Conway's Game of Life in 3D

a cellular automaton exploration

What we are trying to achieve

- core life logic for 3d
- with periodic boundaries
- scalable mpi implementation
- generator of rule sets and primordial soups
- analyzer of evolving populations
- detector for interesting shapes (gliders)
- visualization for interesting outcomes

What we are trying to achieve

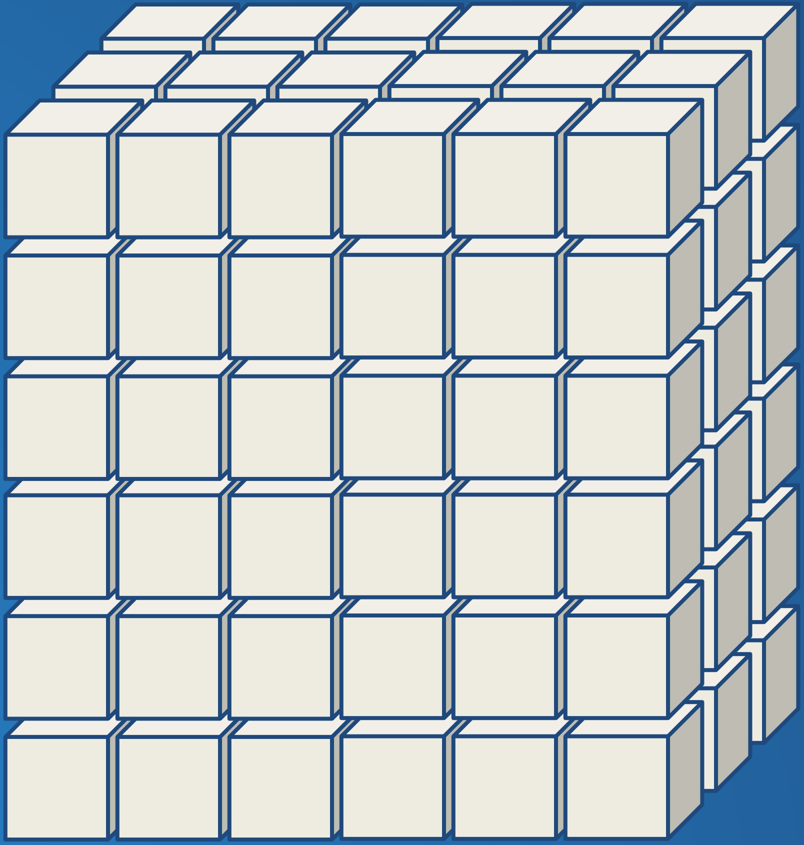
- core life logic for 3d - DONE
- with periodic boundaries - DONE
- scalable mpi implementation - DONE
- generator of rule sets and primordial soups - DONE
- analyzer of evolving populations
- detector for interesting shapes (gliders)
- visualization for interesting outcomes

Parallelization scheme I

Setup:
Master: parse input world
Collective: Scatterv (distribute initial world in chunks of multiple z layers to processes)

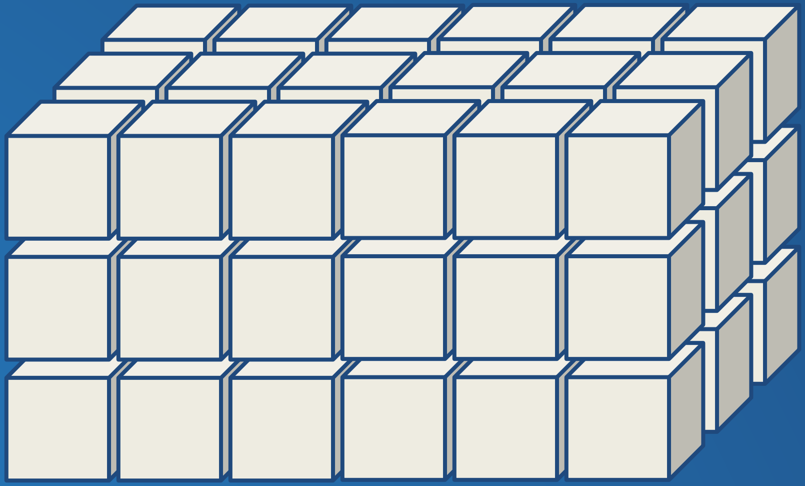
Repeat:
Simultaneously: exchange front and back layer of zlayer-chunk between ‘neighbouring processes’
Each: calculate next generation
Collective: Gather to calculate population

Input

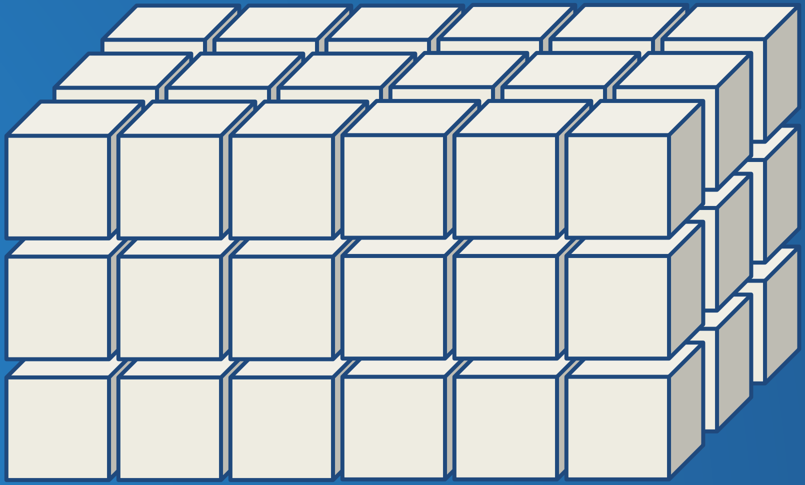


Proc 0

(MASTER)

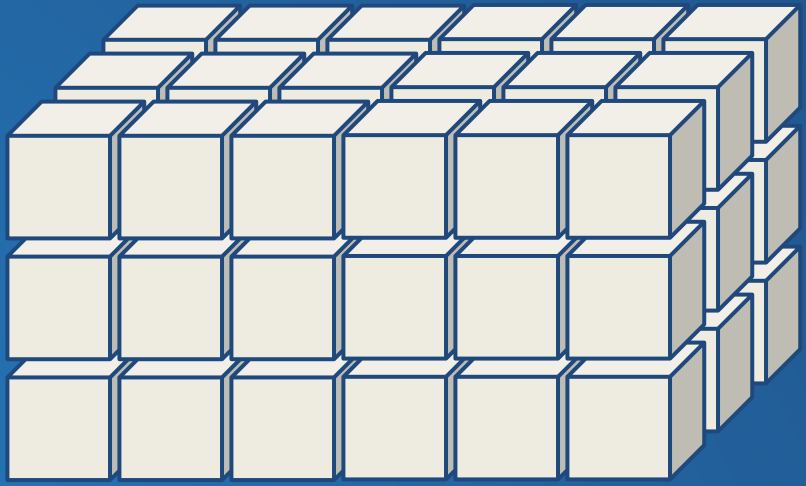


Proc 1

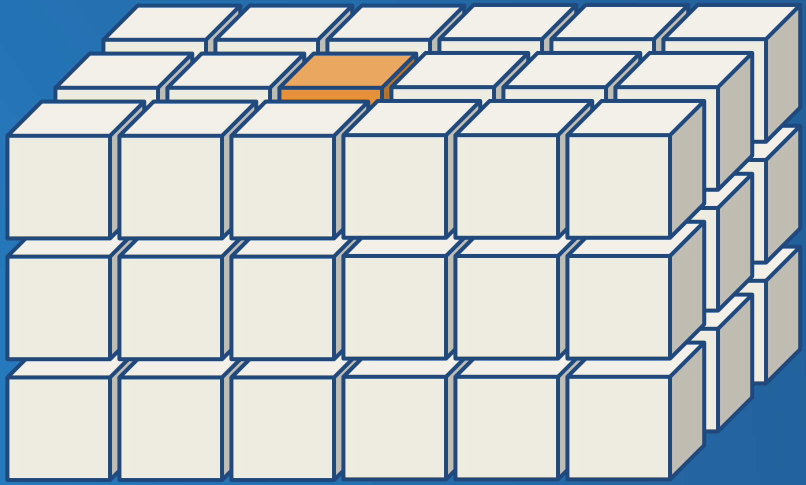


Proc 0

(MASTER)

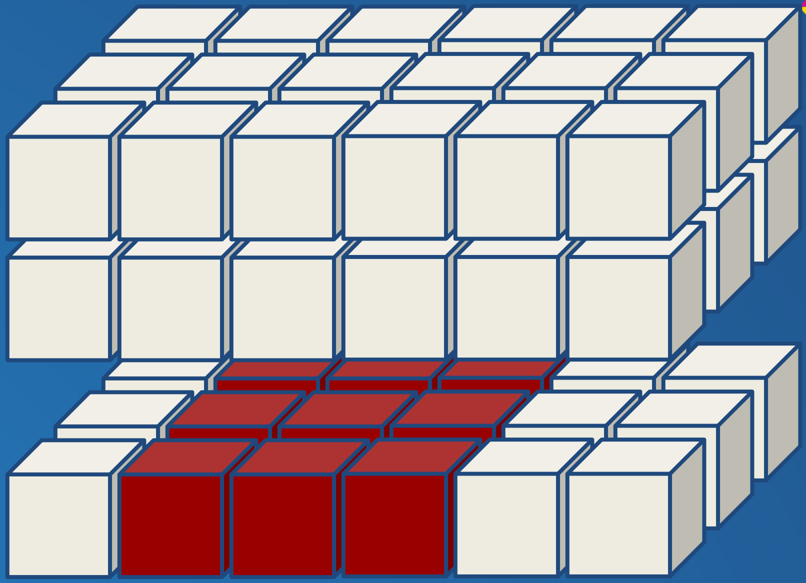


Proc 1

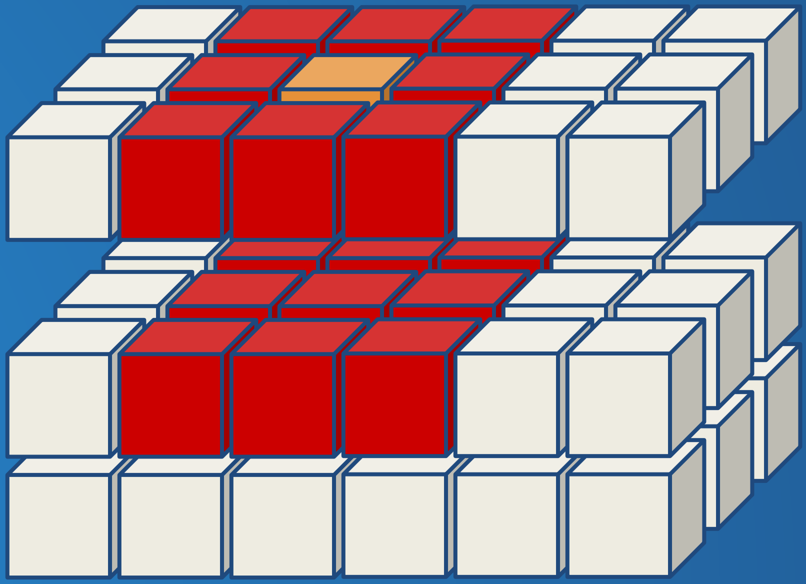


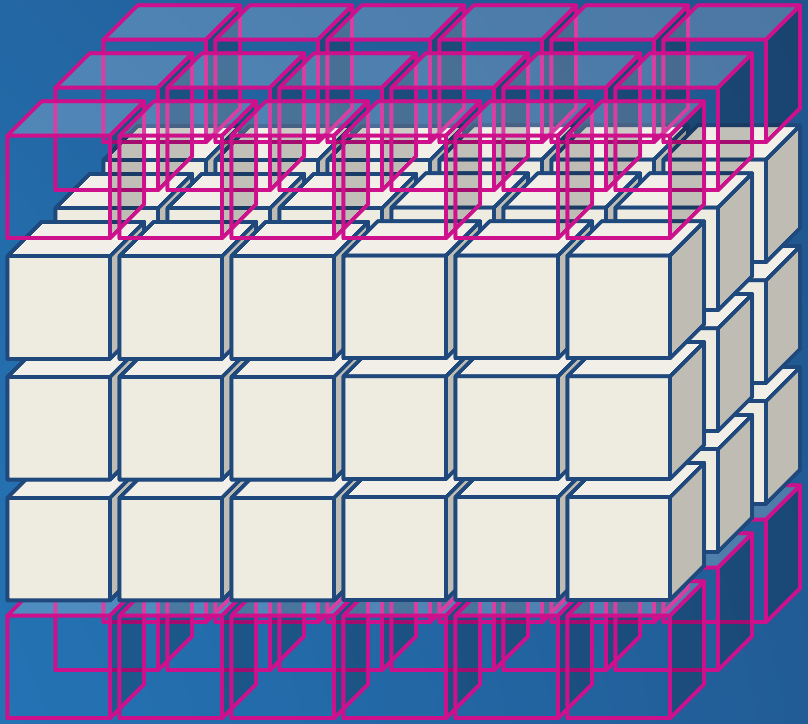
Proc 0

(MASTER)



Proc 1





Buffer for Neighbour Layer

Border Layer (Send)

Internal Layer

Border Layer (Send)

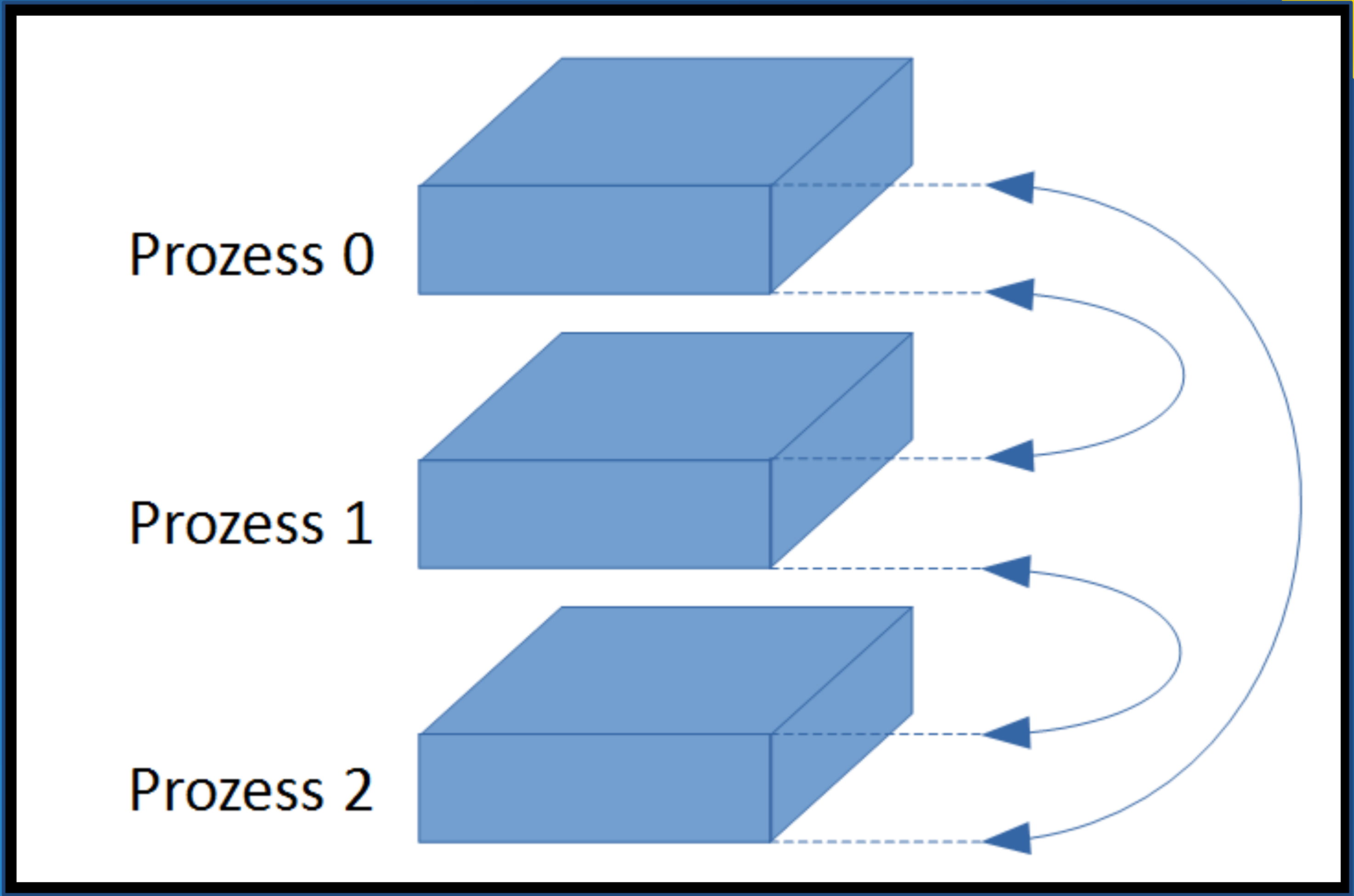
Buffer for Neighbour Layer

Parallelization scheme II

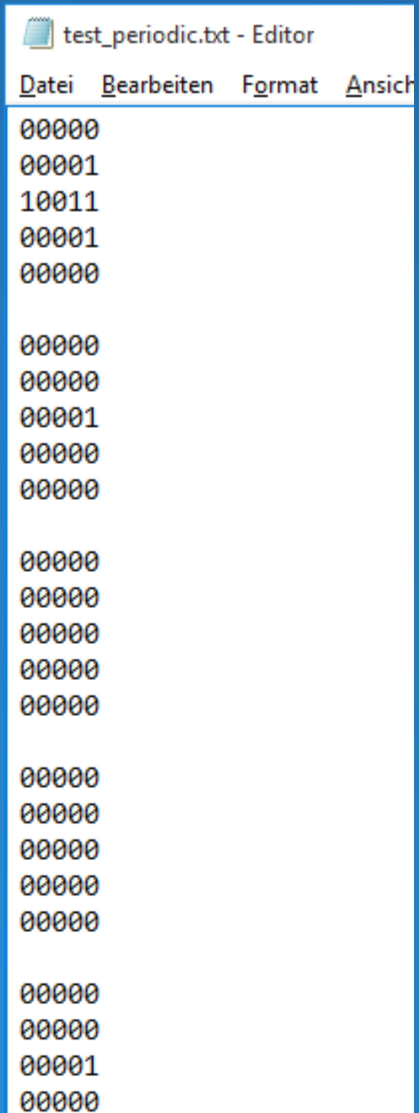
The exchange (simple version):

```
if (procId % 2 == 0)
    send back layer to next process
    recv last layer as front layer from previos process
    send front layer to prev process
    ...
else
    recv back layer as front layer from previos process
    send back layer to next process
    recv front layer as back layer from next process
    …
```

Order is important, so that no deadlocks happen, and the application scales nicely with even or uneven number of processes


Prozess 0
Prozess 1
Prozess 2

Example of input

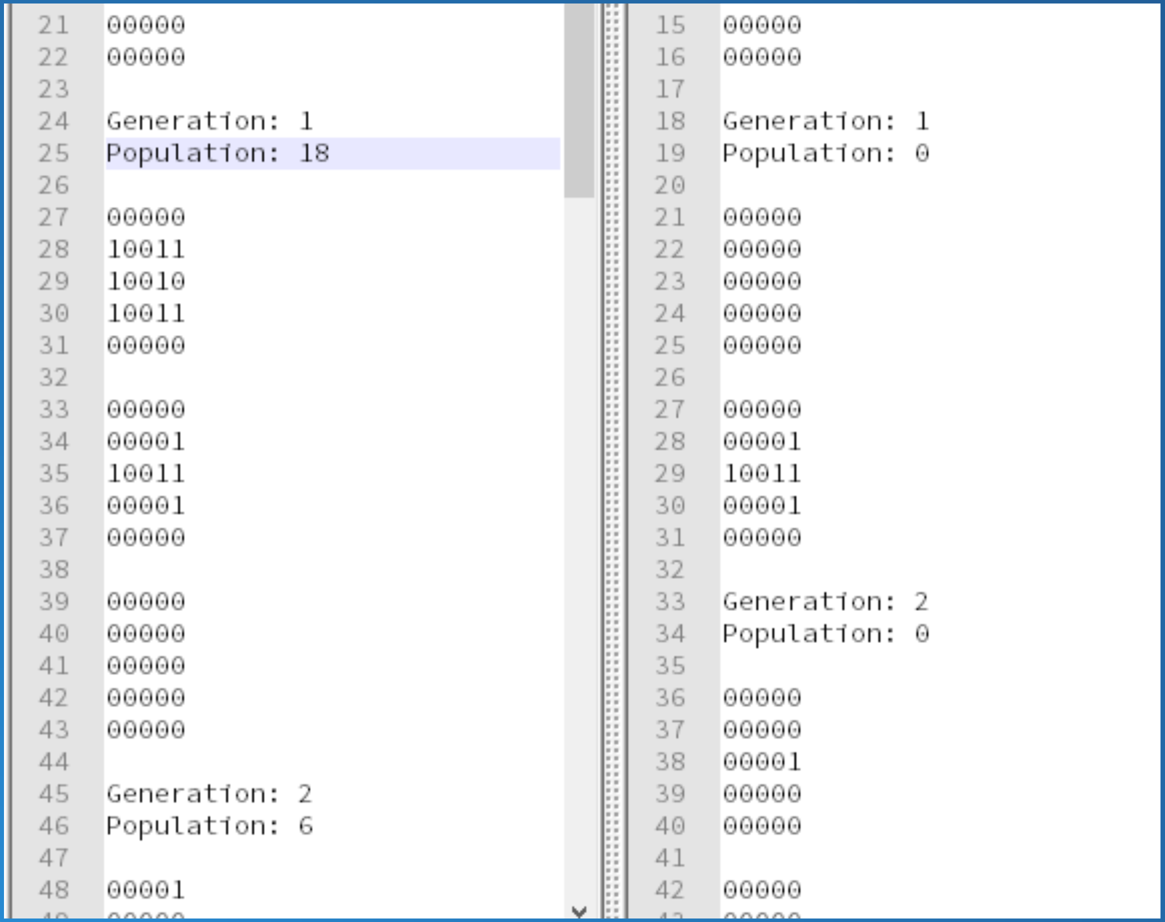


```
00000
00001
10011
00001
00000

00000
00000
00001
00000
00000

00000
00000
00000
00000
00000

00000
00000
00000
00000
00000

00000
00000
00001
00000
```

example command to execute program:

```
mpiexec -np 2 ./pargol test_periodic.txt
-xlen 5 -ylen 5 -zlen 5
```

Example of output



```
00000
00000

Generation: 1
Population: 18

00000
10011
10010
10011
00000

00000
00001
10011
00001
00000

00000
00000
00000
00000
00000

Generation: 2
Population: 6

00001
```

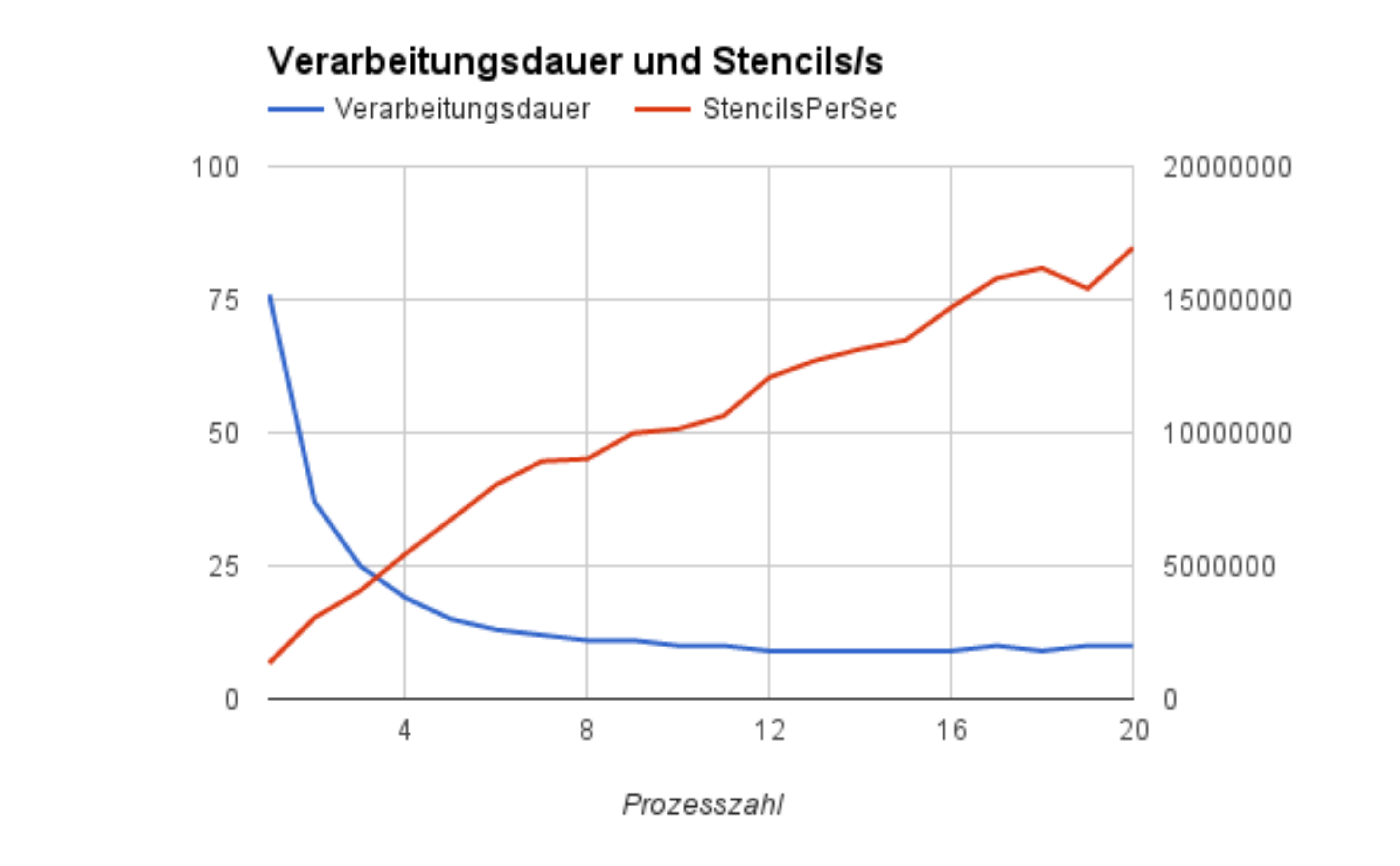
```
00000
00000

Generation: 1
Population: 0

00000
00000
00000
00000
00000

00000
00001
10011
00001
00000

Generation: 2
Population: 0

00000
00000
00001
00000
00000

00000
```

2 processes: output divided by zlayer-chunks

rules are hardcoded at the moment

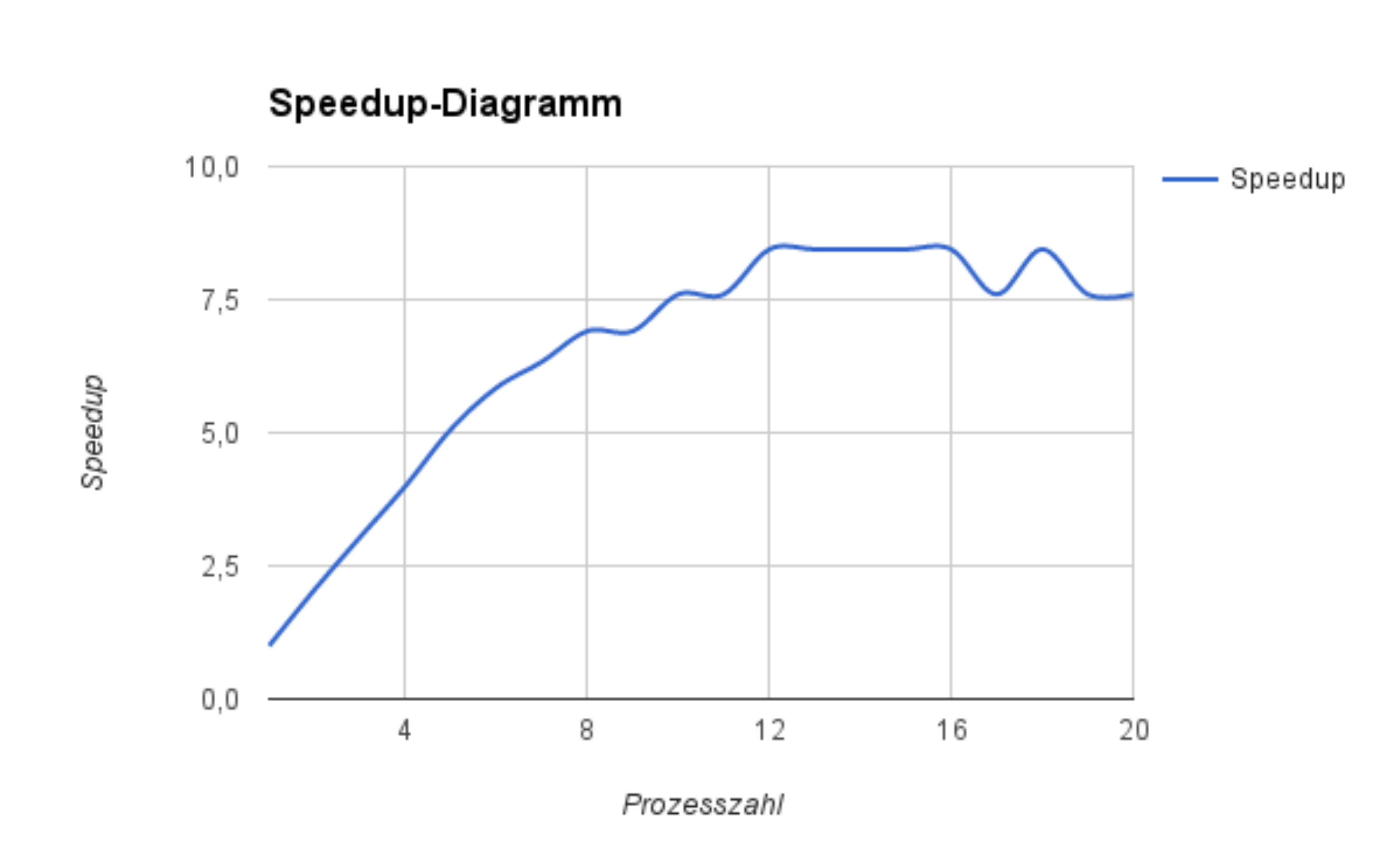
for this example LIFE 4555 was used

Laufzeitmessung



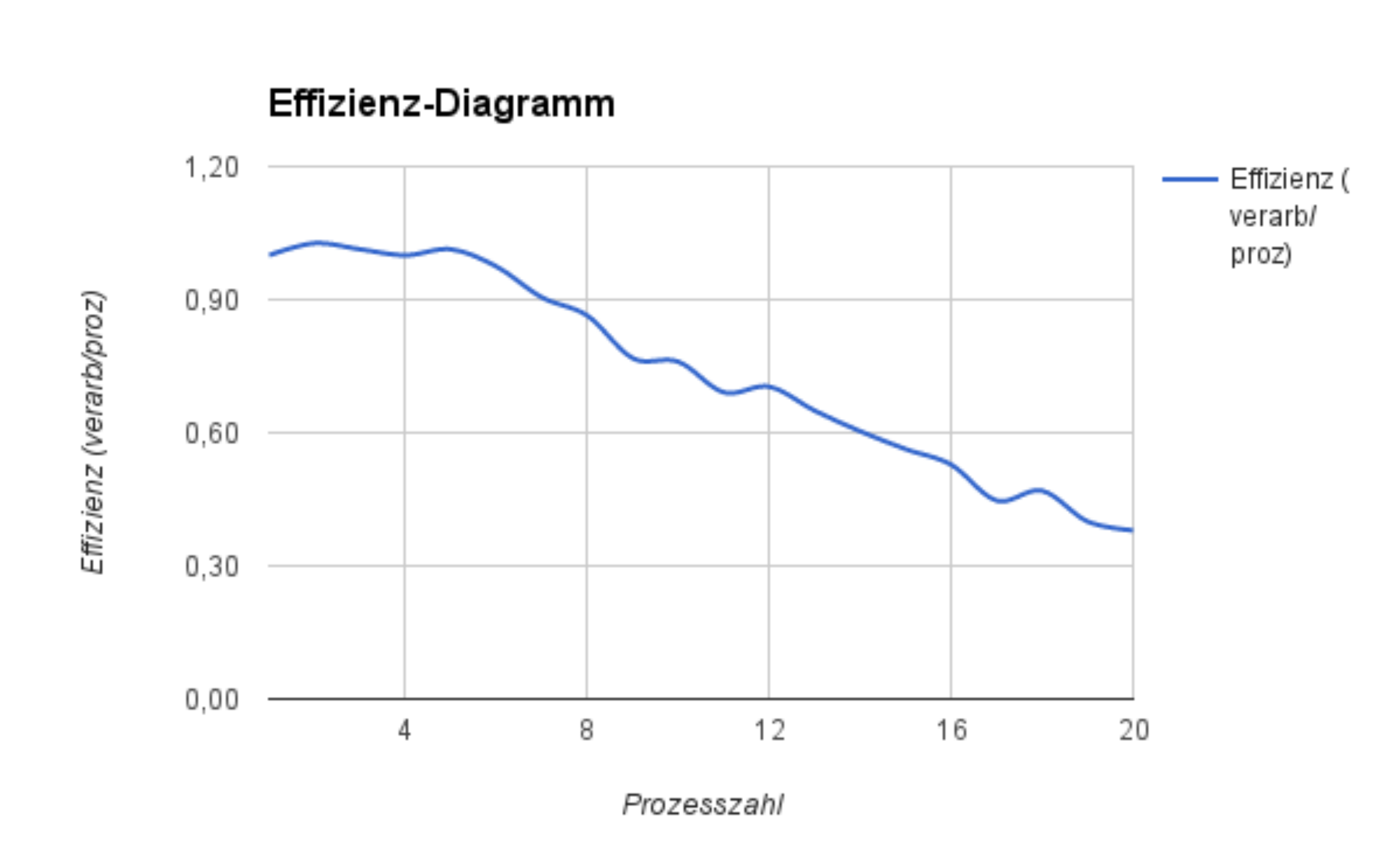
1 Stencil = 1 Ausführung von countNeighbours

Parallele Beschleunigung



Umbruch bei 17 / 18 Prozessen

Parallele Effizienz

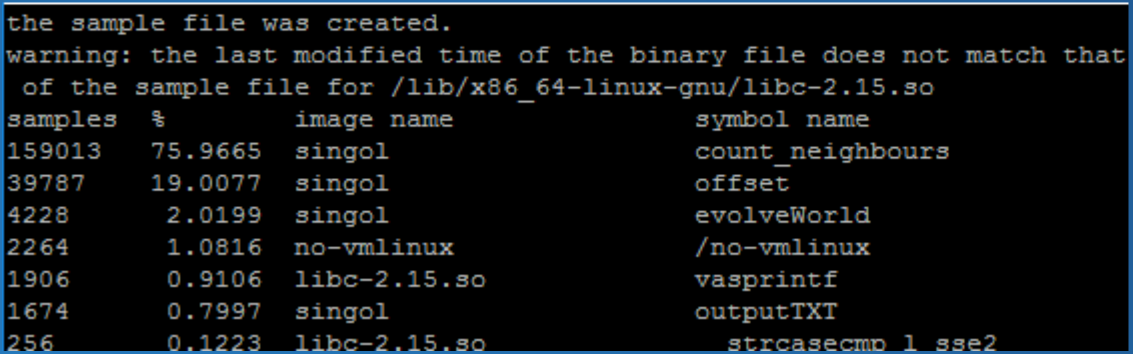


Bis zu 6 Prozesse arbeiten effizient, am gestellten Problem

Auswertungsergebnisse I:

- nicht sehr gut im strong scaling (kommt aber auf die Problemgröße und -form an)
- viel Potenzial für weak-scaling

OProfile

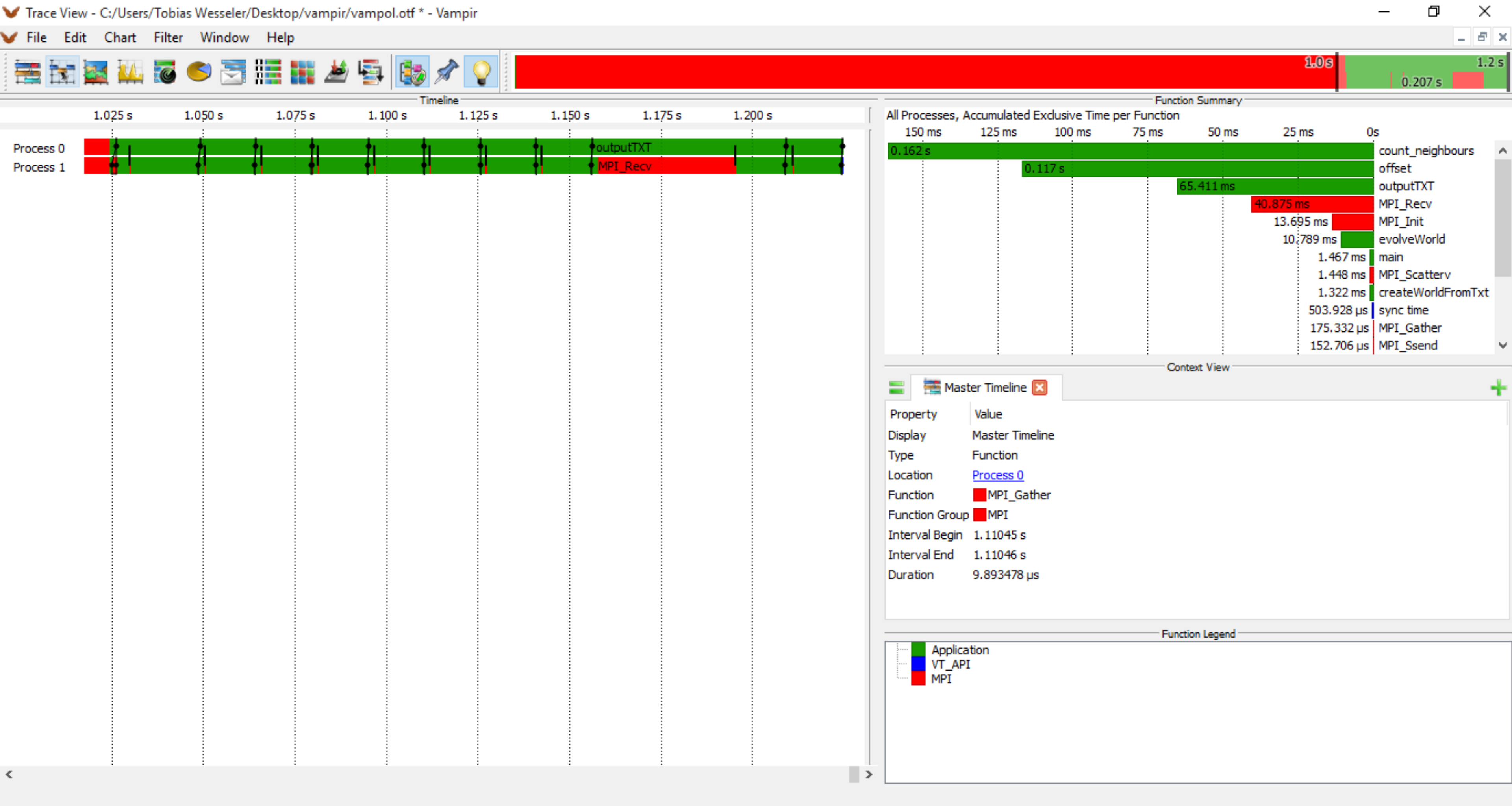


```
the sample file was created.
warning: the last modified time of the binary file does not match that
 of the sample file for /lib/x86_64-linux-gnu/libc-2.15.so
samples  %        image name               symbol name
159013   75.9665  singol                   count_neighbours
39787    19.0077  singol                   offset
4228      2.0199  singol                   evolveWorld
2264      1.0816  no-vmlinux               /no-vmlinux
1906      0.9106  libc-2.15.so             vasprintf
1674      0.7997  singol                   outputTXT
256       0.1223  libc-2.15.so             __strcasecmp_l_sse2
```

- 76% der CPU-Zeit in countNeighbours
- 19% der Zeit in offset

- entspricht den Erwartungen

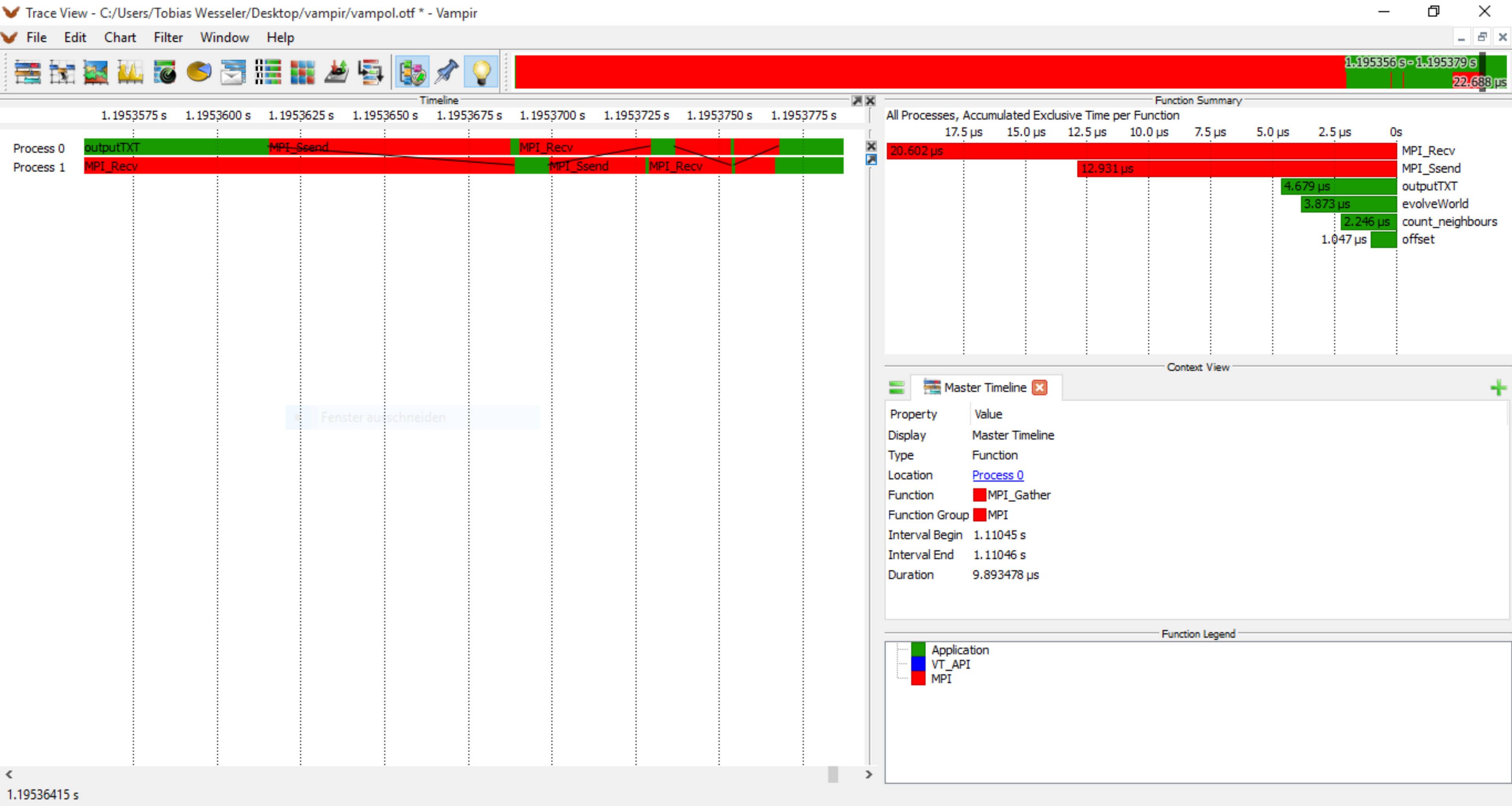
VampirTrace



Property	Value
Display	Master Timeline
Type	Function
Location	Process 0
Function	🟥 MPI_Gather
Function Group	🟥 MPI
Interval Begin	1.11045 s
Interval End	1.11046 s
Duration	9.893478 µs

- 🟩 Application
- 🟦 VT_API
- 🟥 MPI

VampirTrace

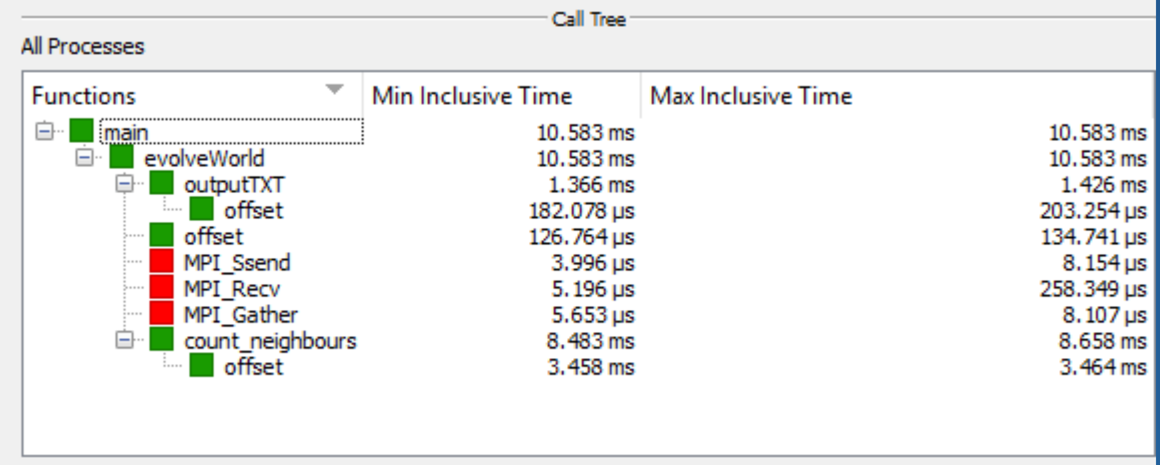


Trace View - C:/Users/Tobias Wesseler/Desktop/vampir/vampol.otf * - Vampir

Property	Value
Display	Master Timeline
Type	Function
Location	Process 0
Function	🟥 MPI_Gather
Function Group	🟥 MPI
Interval Begin	1.11045 s
Interval End	1.11046 s
Duration	9.893478 µs

🟩 Application
🟦 VT_API
🟥 MPI

VampirTrace



Call Tree

All Processes

Functions	Min Inclusive Time	Max Inclusive Time
⊟ 🟩 main	10.583 ms	10.583 ms
⊟ 🟩 evolveWorld	10.583 ms	10.583 ms
⊟ 🟩 outputTXT	1.366 ms	1.426 ms
🟩 offset	182.078 µs	203.254 µs
🟩 offset	126.764 µs	134.741 µs
🟥 MPI_Ssend	3.996 µs	8.154 µs
🟥 MPI_Recv	5.196 µs	258.349 µs
🟥 MPI_Gather	5.653 µs	8.107 µs
⊟ 🟩 count_neighbours	8.483 ms	8.658 ms
🟩 offset	3.458 ms	3.464 ms

- “offset”-Aufrufe möglicherweise reduzierbar/optimierbar

Auswertungsergebnisse II:

- Das Programm verschickt nur so wenig Daten wie möglich
- Hauptzeit wird mit Entwicklung der Welten verbracht
- Verhält sich wie erwünscht

- Aber: großes Potenzial für weitere Features und Optimierungen

Thank you

and happy coding… :)