Vampir

TECHNISCHE UNIVERSITÄT DARMSTADT

MPI-Performance Analysis

How does one gain insight about an HPC applciation?

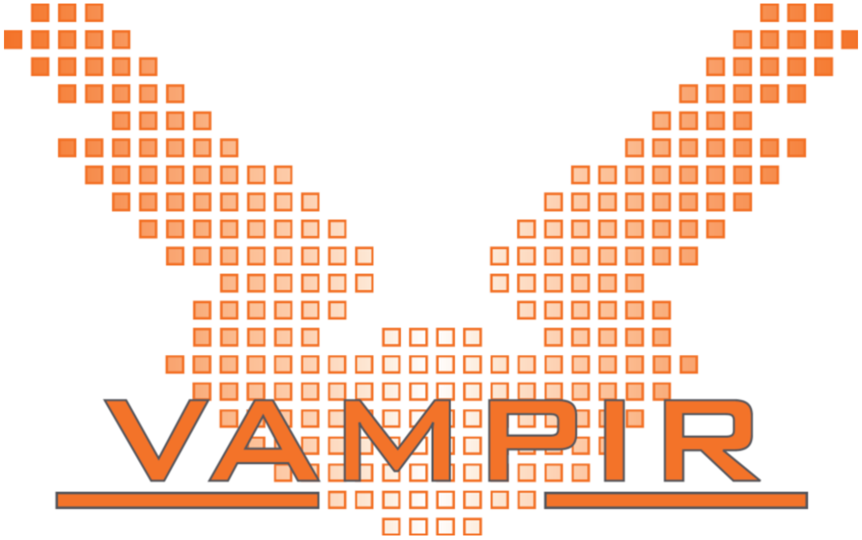
Data has to be presented to the developer in an understandable way in order to gain insight!

- **Data can be broken up**
  - by time
    - Example: Function entries and exits are displayed on a timeline
  - attributed to graphs representing properties of the source code
    - Example: Display function execution times on the call graph
  - etc…

Introduction to Vampir

Overview of Vampir

How to use Vampir



Facts about Vampir

Actually a tool collection

• Vampir Trace	freely available
• Vampir GUI	commercial

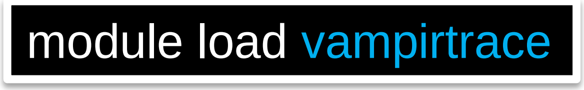
Developed at the TU Dresden

Many additional recording capabilities

- Memory Usage
- I/O Activity Tracing
- CPU-counters

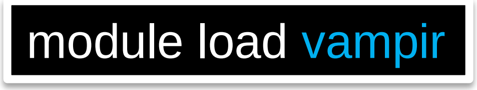
Prepare your environment

Vampir-trace



```
module load vampirtrace
```

Vampir (GUI)



```
module load vampir
```

Note:

- **Vampir-Trace is not necessary for visualization of data**
- **The Vampir module must be loaded for analysis**

How to use

To use Vampir the following steps are necessary:

1. **Select the correct compiler wrapper**

C:

```
vtcc  -vt:cc
```

C++:

```
vtcxx  -vt:cxx
```

Fortran:

```
vtf90  -vt:f90
```

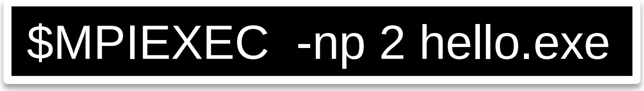
```
vtf77  -vt:f77
```

2. **Instrument your application by recompiling it**

```
vtcc –vt:cc gcc hello.c –o hello.exe
```

How to use

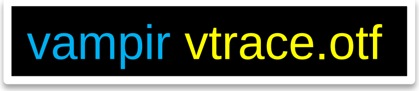
3. Execute as usual



```
$MPIEXEC  -np 2 hello.exe
```

⇨ Results in a OTF-file

4. Analyze application



```
vampir vtrace.otf
```

Interlude #1: Measurement Basics

Two independent decisions:

1. When performance is measurement triggered

- Sampling
  - Triggered by timer interrupt or by hardware counter overflow
  - Can measure unmodified executables, potential low overhead
- Code instrumentation:
  - Triggered by “instrumentation hooks” inserted into the code
  - Insertion can be done manually or automatically

2. How performance is data recorded

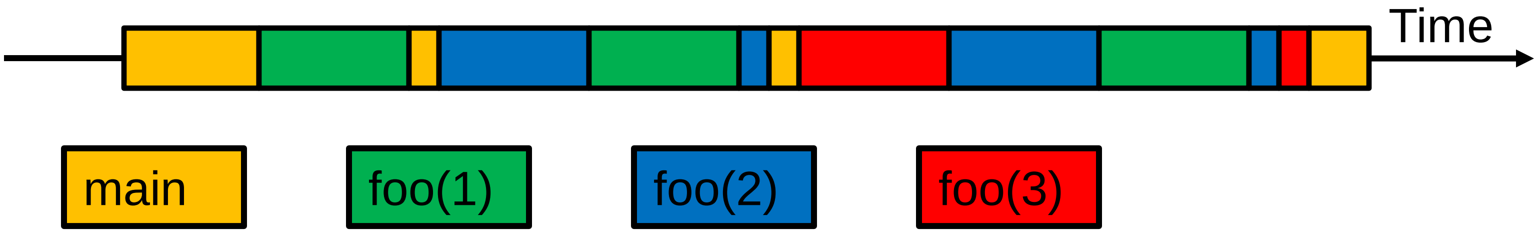
- Profile ::= Summarization of events over time
  - run time summarization (functions, call sites, loops, …)
- Trace file ::= Sequence of events over time

A study of a short example program

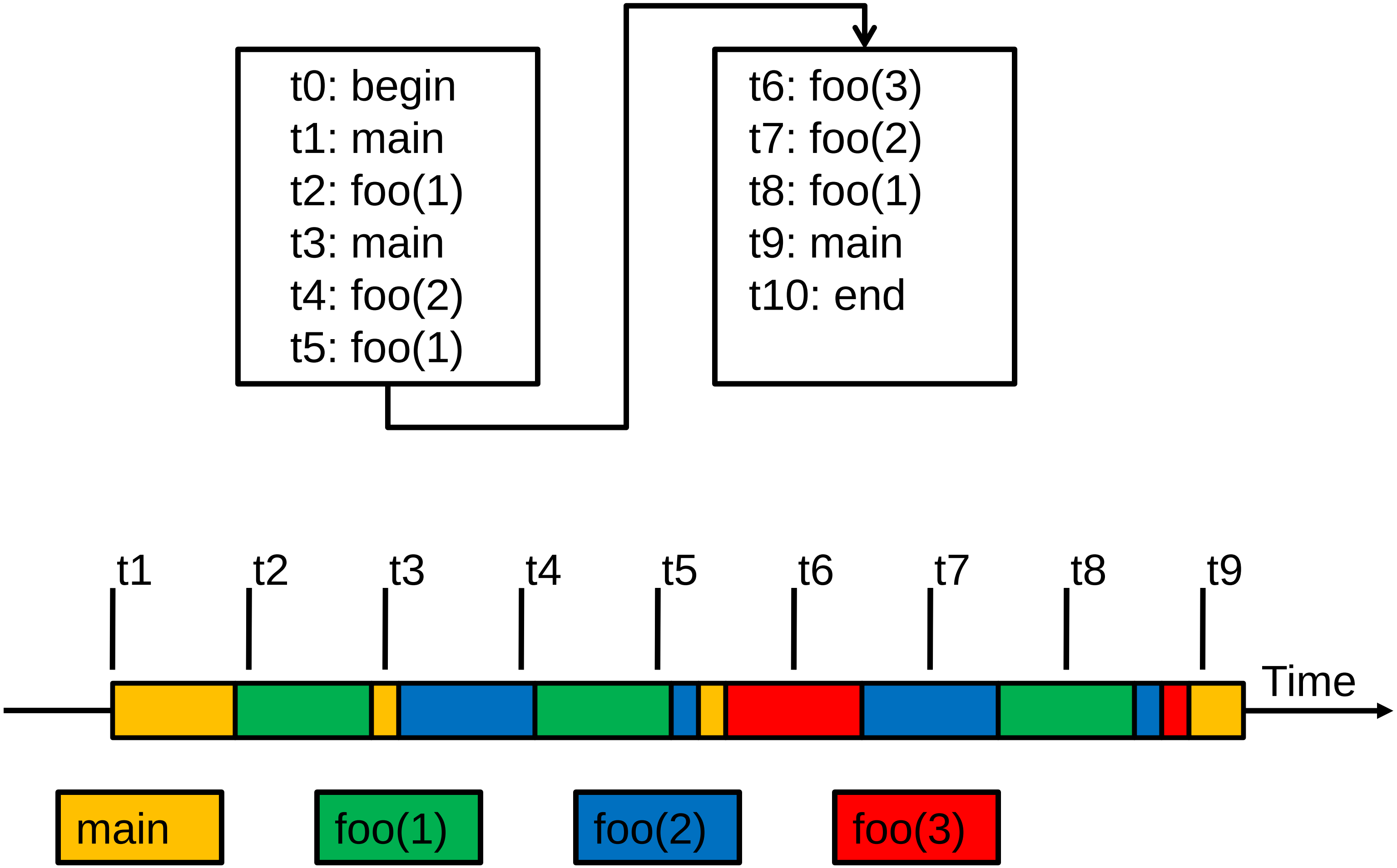
To explain the differences:

```
main(..)
{
  for (i=1..3)
  {
    foo(i)
  }
}

foo(i)
{
  if (i>0)  foo(i-1)
}
```



Example: Sampling



t0: begin
t1: main
t2: foo(1)
t3: main
t4: foo(2)
t5: foo(1)

t6: foo(3)
t7: foo(2)
t8: foo(1)
t9: main
t10: end

Summary: Sampling

Sampling:

The application is probed at specific times and a set of interesting metrics is gathered

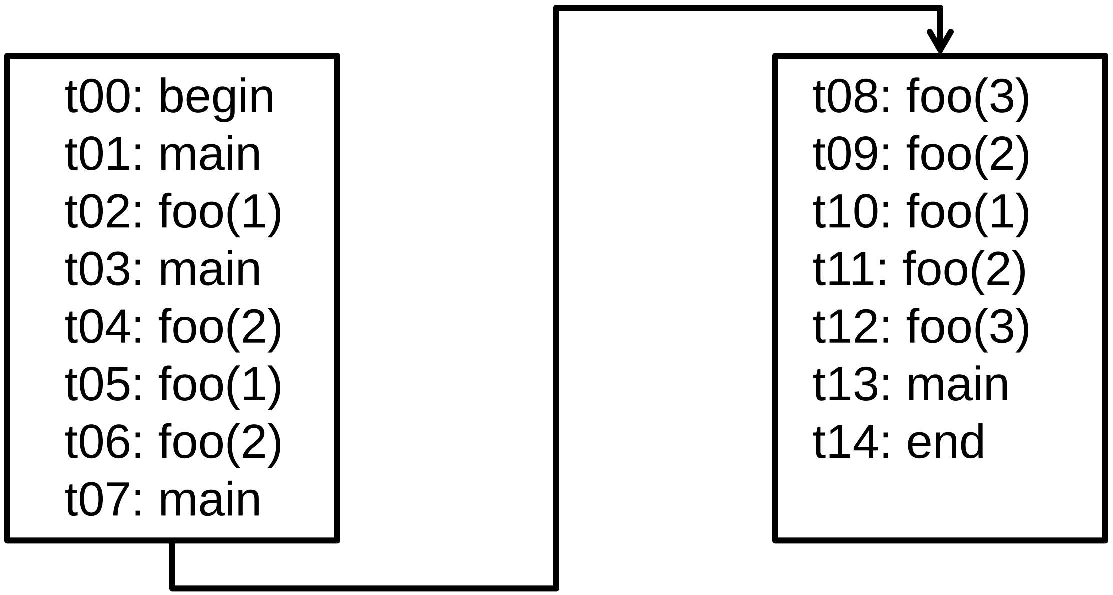
Advantages:

+ **Low perturbance of the application**
+ **Application does not have to be recompiled**
+ **Works well with large and long running applications**

Disadvantages:

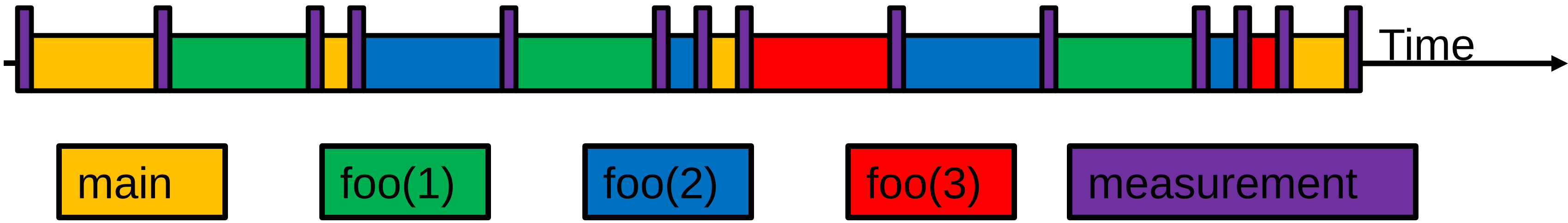
- **Not very detailed information on high frequency metrics**

Example: Instrumentation



t00: begin
t01: main
t02: foo(1)
t03: main
t04: foo(2)
t05: foo(1)
t06: foo(2)
t07: main

t08: foo(3)
t09: foo(2)
t10: foo(1)
t11: foo(2)
t12: foo(3)
t13: main
t14: end



Summary: Instrumentation

Idea:

The code is instrumented such that every interesting event is recorded as it occurs.

Advantages:

- + **Every event of interest can be captured**
- + **Much more detailed information possible**

Disadvantages:

- - **Preprocessing of the program necessary**
- - **Probably expensive at runtime**

Critical issues

Accuracy

- Perturbation

  Measurement alters program behaviour
- Intrusion overhead

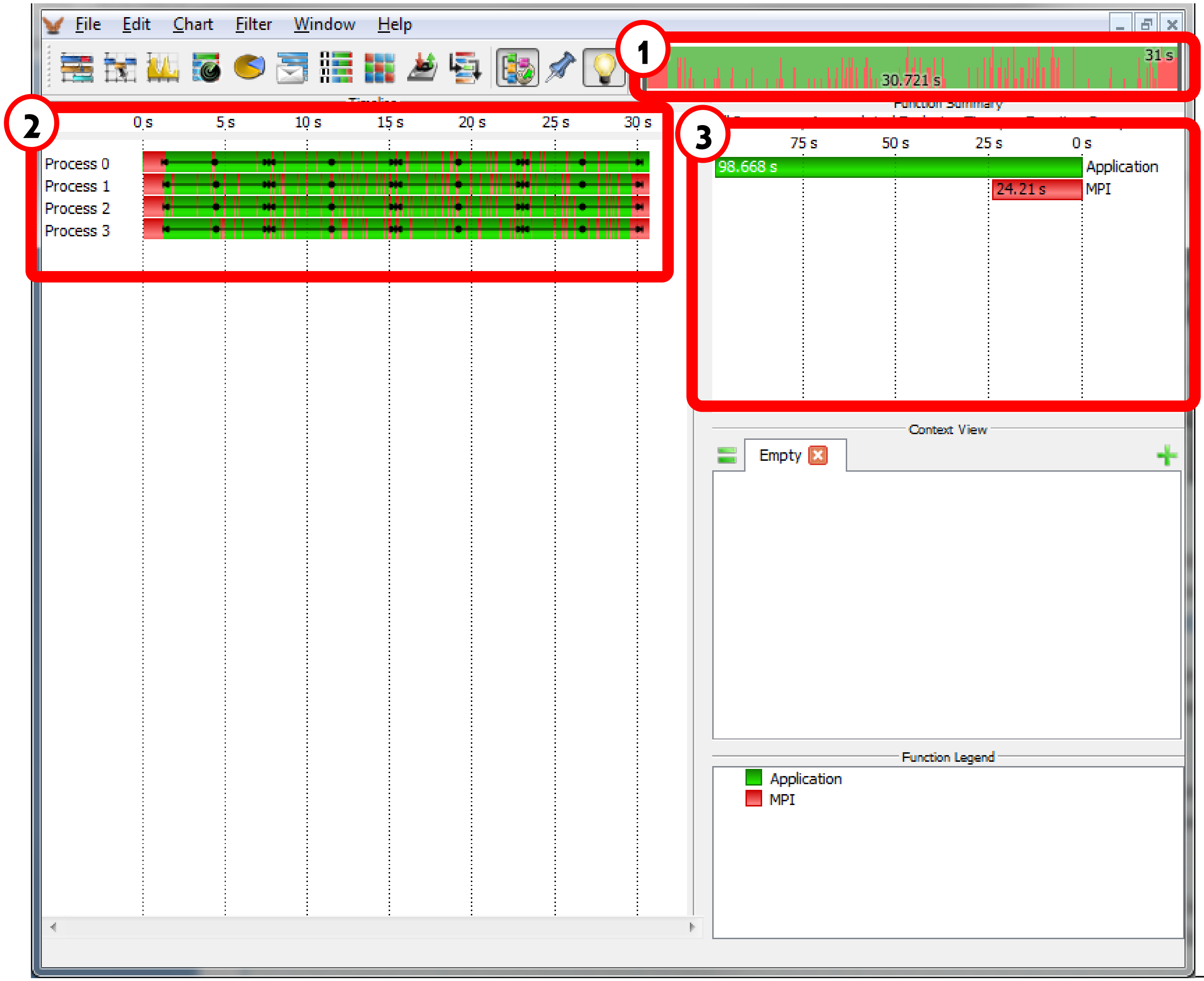
  Measurement itself needs time and thus lowers performance
- Accuracy of timers, counters

Granularity

- How many measurements
- How much information / work during each measurement

Trade-off: Accuracy ⇔ Expressiveness of data

Vampir - Main Window



1) **Overview Timeline**
2) **„Master“ Timline**
3) **Function Summary**

Function Summary:

- all data is grouped
- aggregated for all processes

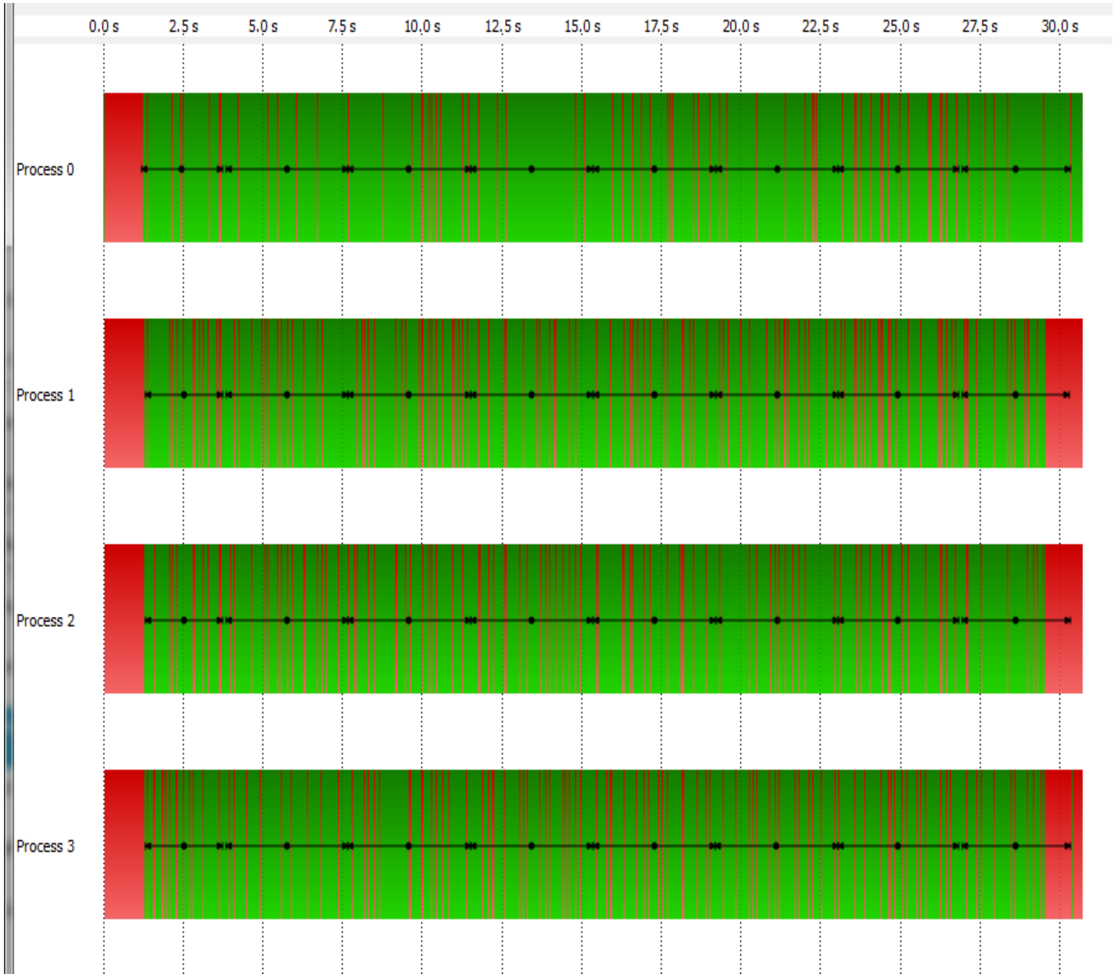
To Access a pannels options: **Right Click in the Window**

Important Vampir Charts

Main Charts

1) Master Timeline
2) Process Timeline
3) Counter Timeline
4) Function Summary
5) Message Summary
6) Process Summary
7) Communication Matrix
8) Call Tree

Master Timeline

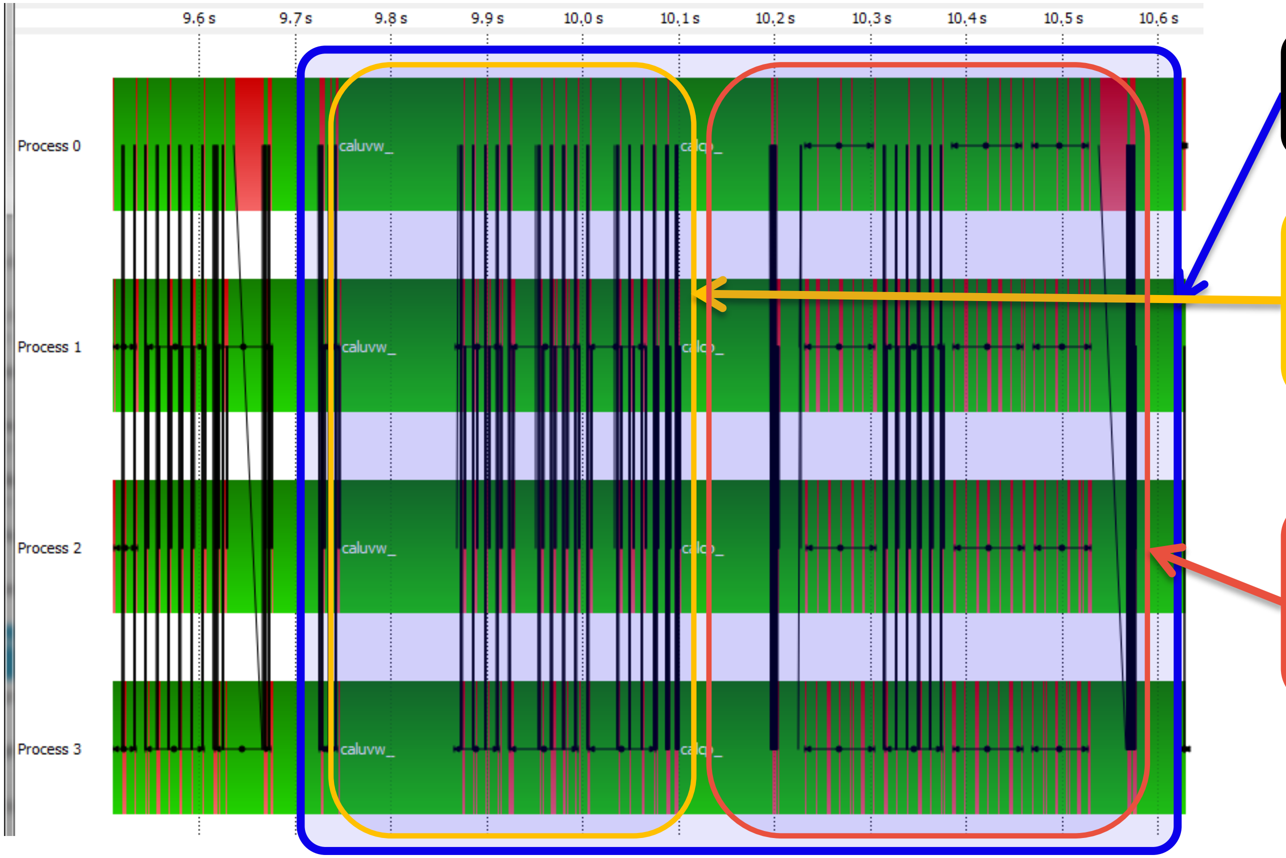


- Timeline for every process (and thread)

Possible Options:

- Vertical Zooming
- Horizontal Zooming
- Vertical / Horizontal Panning

Master Timeline (more detail)

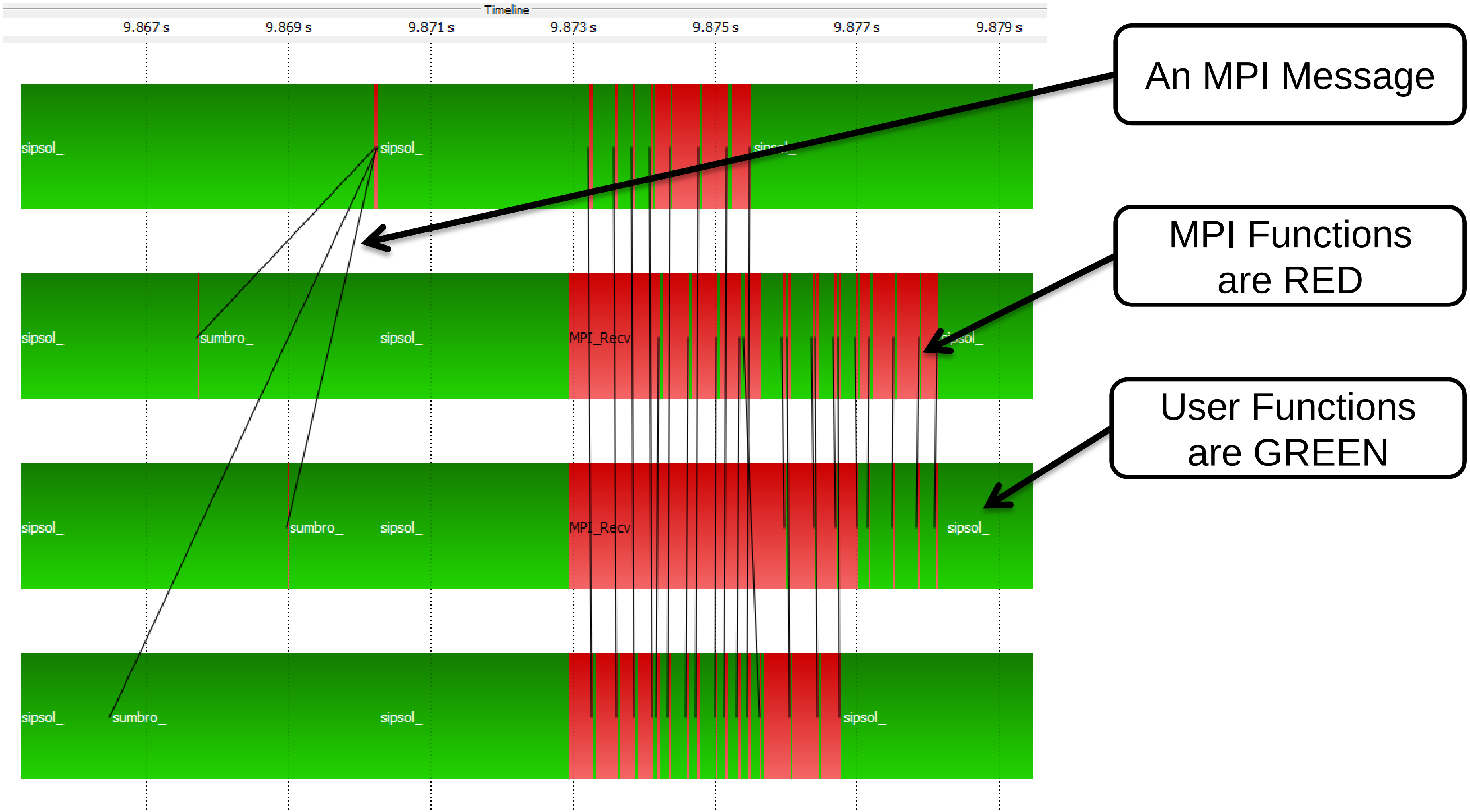


Single FASTEST Iteration

Solving of Mass Transport/ Flow Equation

Solving of Mass Transport/ Flow Equation

Master Timeline (even more detail)

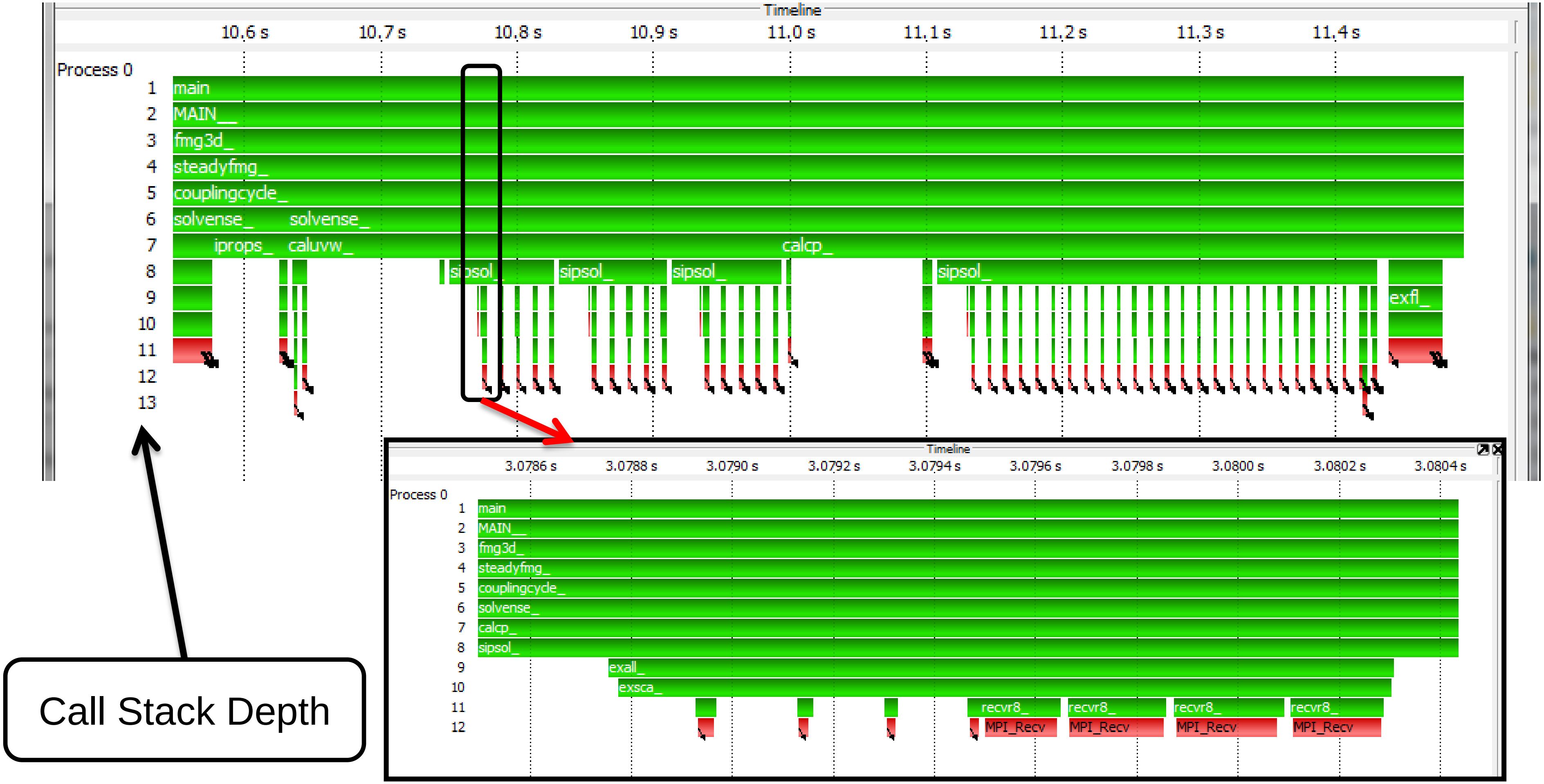


An MPI Message

MPI Functions are RED

User Functions are GREEN

Process Timeline



Call Stack Depth

Interlude #2:
What are Hardware Performance Counter?

- **Hardware Performance Counter (Wikipedia):**
  … a set of (programmable) special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems …

- **What do they do?**
  Example MEGAFLOPS
  - MEGA = Million (10^6)
  - FLOPS = FLoatingpoint Operations Per Second
  - Tell the CPU to count the number of floating point operations

Excerpt of available counters Nehalem X5570

- Level 1 data cache misses
- Level 1 instruction cache misses
- Level 2 data cache misses
- Level 2 instruction cache misses
- Level 3 data cache misses
- Level 3 instruction cache misses
- Level 1 cache misses
- Level 2 cache misses
- Level 3 cache misses
- Requests for a snoop
- Requests for exclusive access to shared cache line
- Requests for exclusive access to clean cache line
- Requests for cache line invalidation
- Requests for cache line intervention
- Level 3 load misses
- Level 3 store misses
- Cycles branch units are idle
- Cycles integer units are idle
- Cycles floating point units are idle
- Cycles load/store units are idle
- Data translation lookaside buffer misses
- Instruction translation lookaside buffer misses
- Total translation lookaside buffer misses
- Level 1 store misses
- Level 1 load misses
- Level 2 load misses
- Level 2 store misses
- Branch target address cache misses
- Data prefetch cache misses
- Level 3 data cache hits
- Translation lookaside buffer shootdowns
- Failed store conditional instructions
- Successful store conditional instructions
- Total store conditional instructions
- Cycles Stalled Waiting for memory accesses
- Cycles Stalled Waiting for memory Reads
- Cycles Stalled Waiting for memory writes
- Cycles with no instruction issue
- Cycles with maximum instruction issue
- Cycles with no instructions completed
- Cycles with maximum instructions completed
- Hardware interrupts
- Unconditional branch instructions
- Conditional branch instructions
- Conditional branch instructions taken
- Conditional branch instructions not taken
- Conditional branch instructions mispredicted
- Conditional branch instructions correctly predicted
- FMA instructions completed

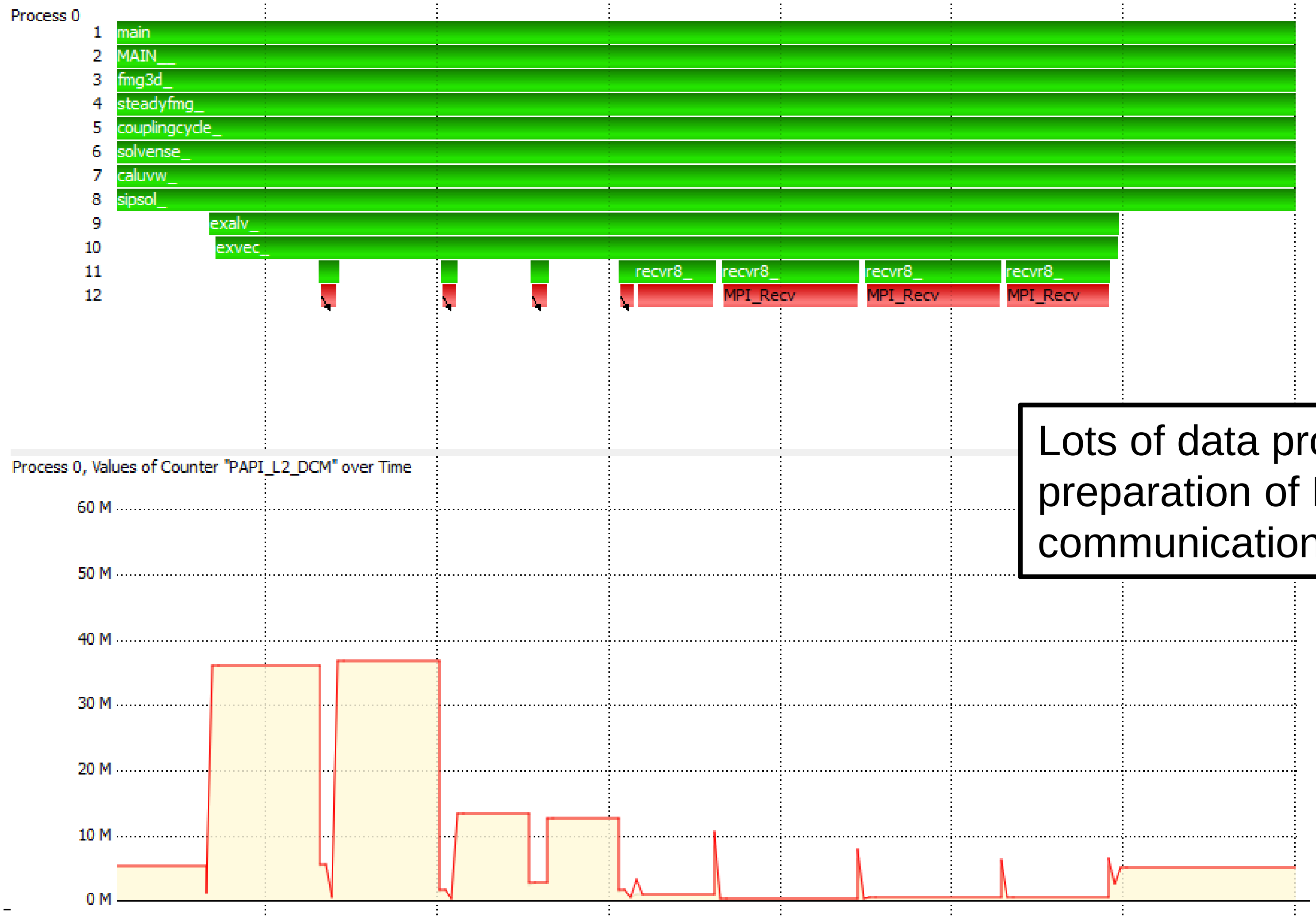
Excerpt of available counters
Nehalem X5570

- Instructions issued
- Instructions completed
- Integer instructions
- Floating point instructions
- Load instructions
- Store instructions
- Branch instructions
- Vector/SIMD instructions (could include integer)
- Cycles stalled on any resource
- Cycles the FP unit(s) are stalled
- Total cycles
- Load/store instructions completed
- Synchronization instructions completed
- Level 1 data cache hits
- Level 2 data cache hits
- Level 1 data cache accesses
- Level 2 data cache accesses
- Level 3 data cache accesses
- Level 1 data cache reads
- Level 2 data cache reads
- Level 3 data cache reads
- Level 1 data cache writes
- Level 2 data cache writes
- Level 3 data cache writes
- Level 1 instruction cache hits
- Level 2 instruction cache hits
- Level 3 instruction cache hits
- Level 1 instruction cache accesses
- Level 2 instruction cache accesses
- Level 3 instruction cache accesses
- Level 1 instruction cache reads
- Level 2 instruction cache reads
- Level 3 instruction cache reads
- Level 1 instruction cache writes
- Level 2 instruction cache writes
- Level 3 instruction cache writes
- Level 1 total cache hits
- Level 2 total cache hits
- Level 3 total cache hits
- Level 1 total cache accesses
- Level 2 total cache accesses
- Level 3 total cache accesses
- Level 1 total cache reads
- Level 2 total cache reads
- Level 3 total cache reads
- Level 1 total cache writes
- Level 2 total cache writes
- Level 3 total cache writes
- Floating point multiply instructions
- Floating point add instructions
- Floating point divide instructions

Excerpt of available counters Nehalem X5570

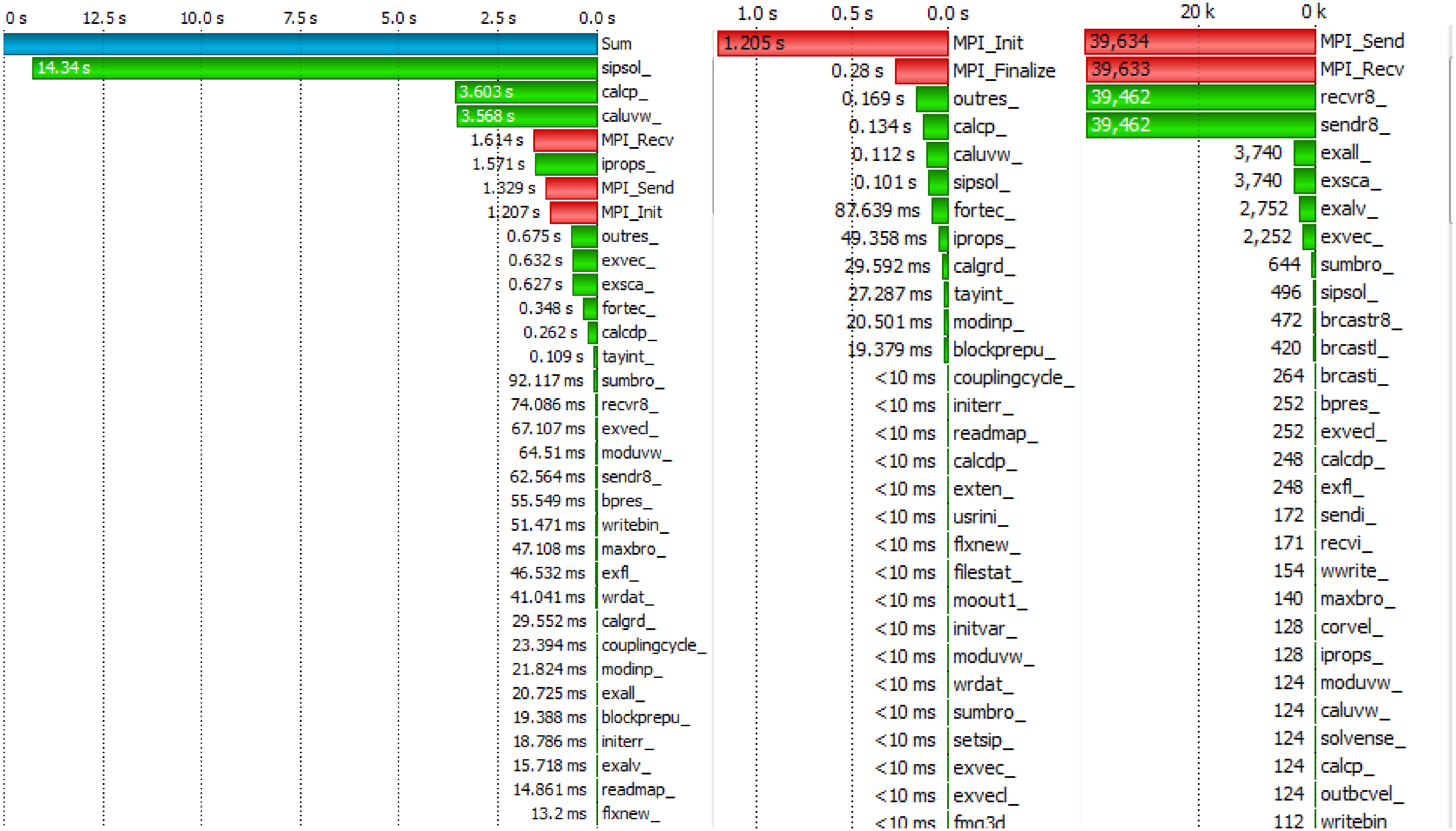
- Floating point square root instructions
- Floating point inverse instructions
- Floating point operations
- Floating point operations; optimized to count scaled single precision vector operations
- Floating point operations; optimized to count scaled double precision vector operations
- Single precision vector/SIMD instructions
- Double precision vector/SIMD instructions

Counter Timeline

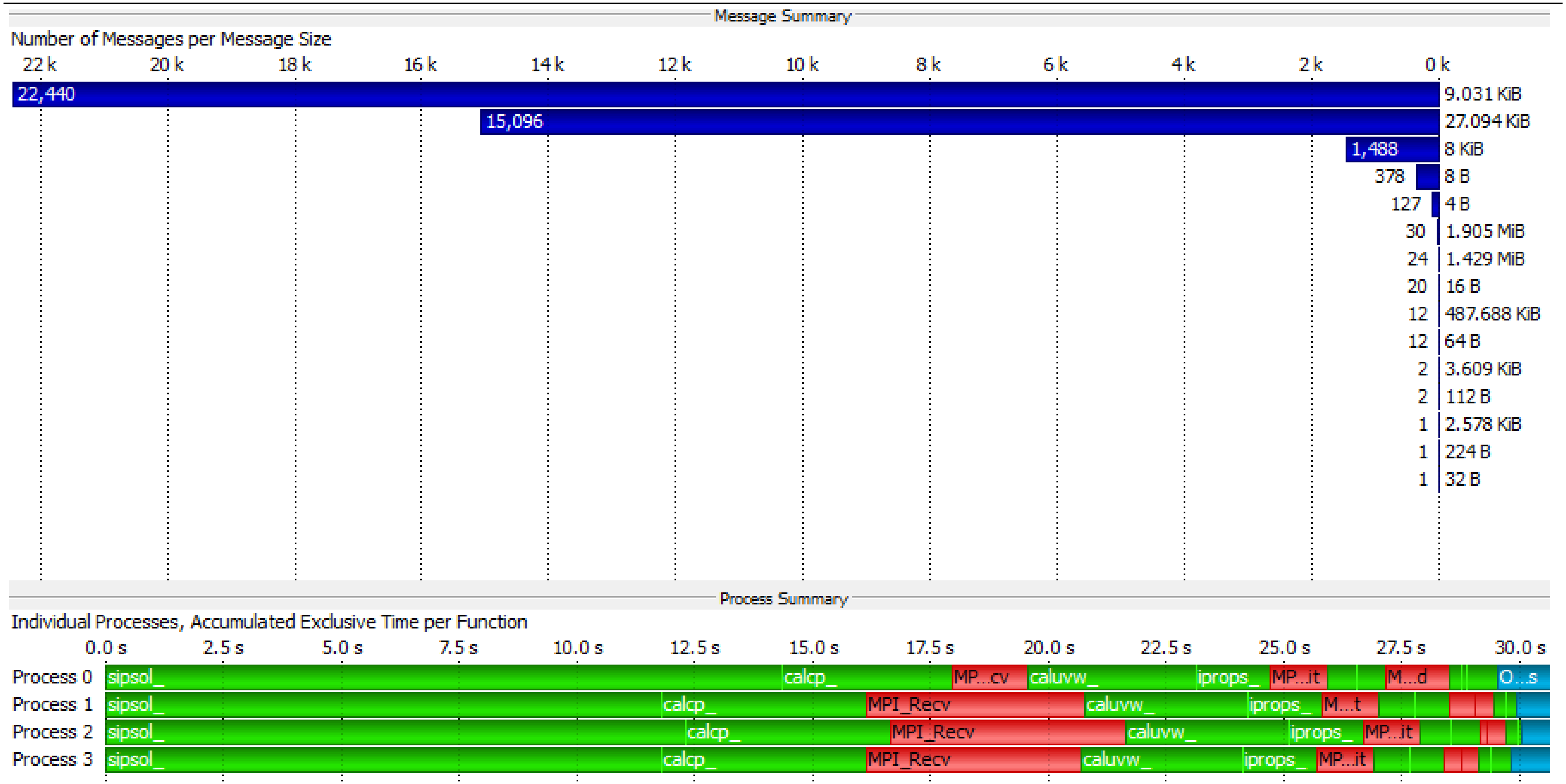


Lots of data processing in preparation of MPI communication

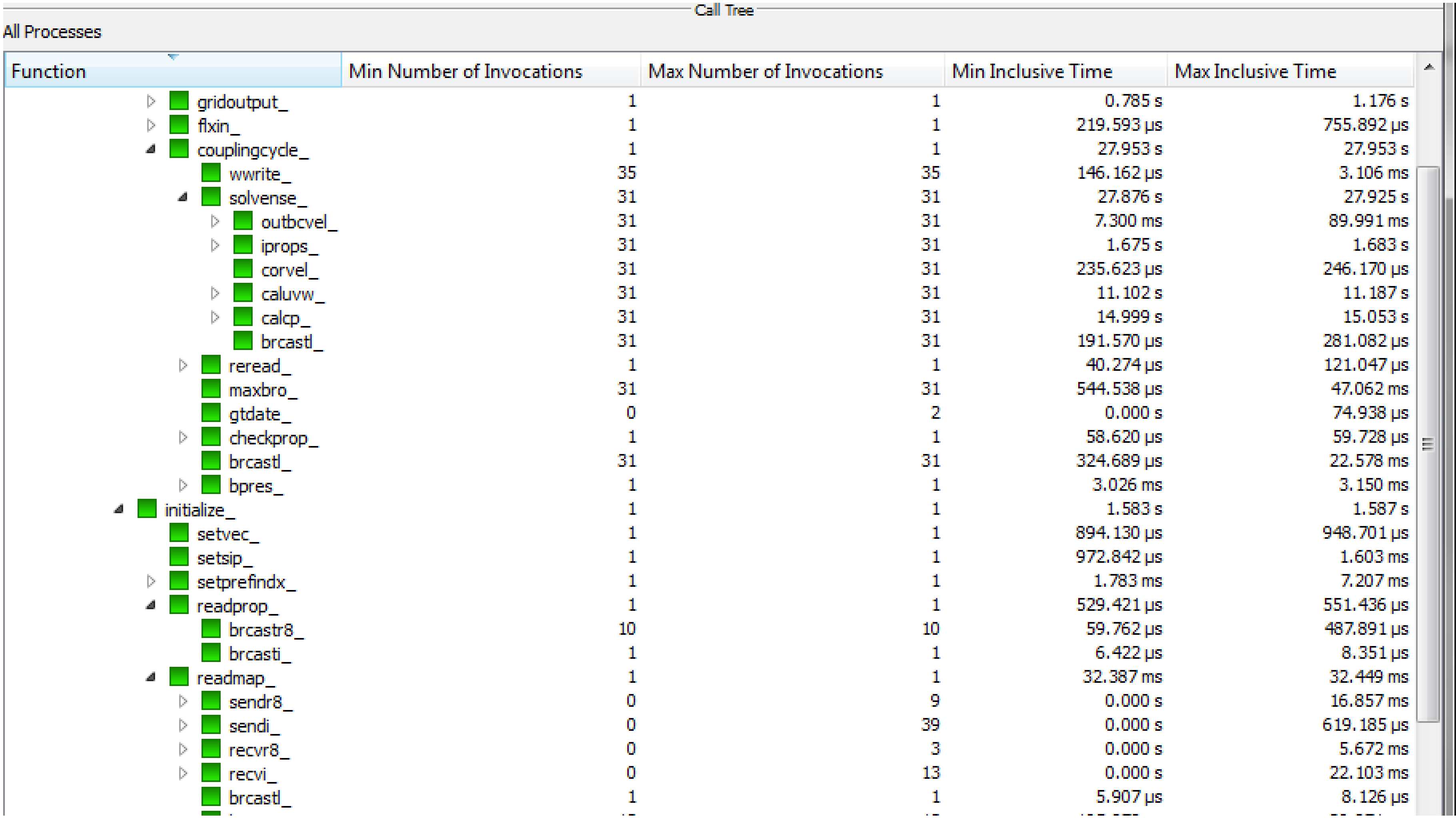
Function Summary



Message Summary
& Process Summary



Call Tree



Call Tree

All Processes

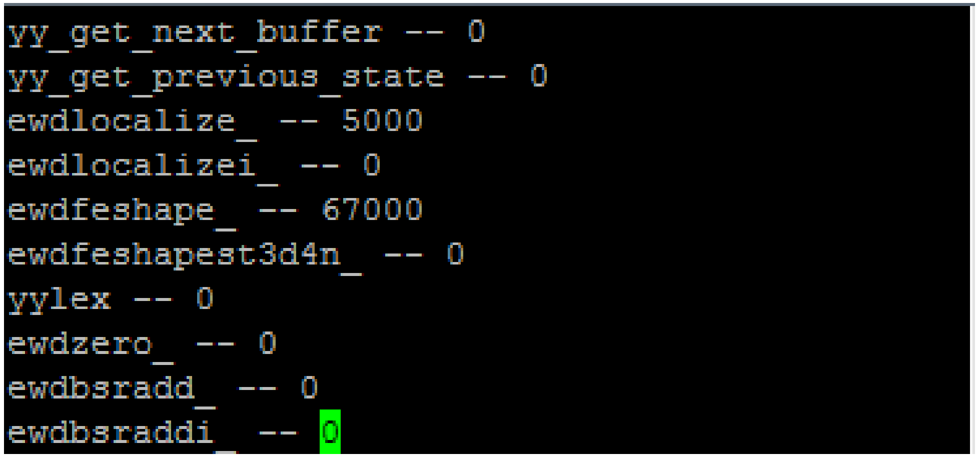
Function	Min Number of Invocations	Max Number of Invocations	Min Inclusive Time	Max Inclusive Time
▷ 🟩 gridoutput_	1	1	0.785 s	1.176 s
▷ 🟩 flxin_	1	1	219.593 µs	755.892 µs
◢ 🟩 couplingcycle_	1	1	27.953 s	27.953 s
🟩 wwrite_	35	35	146.162 µs	3.106 ms
◢ 🟩 solvense_	31	31	27.876 s	27.925 s
▷ 🟩 outbcvel_	31	31	7.300 ms	89.991 ms
▷ 🟩 iprops_	31	31	1.675 s	1.683 s
🟩 corvel_	31	31	235.623 µs	246.170 µs
▷ 🟩 caluvw_	31	31	11.102 s	11.187 s
▷ 🟩 calcp_	31	31	14.999 s	15.053 s
🟩 brcastl_	31	31	191.570 µs	281.082 µs
▷ 🟩 reread_	1	1	40.274 µs	121.047 µs
🟩 maxbro_	31	31	544.538 µs	47.062 ms
🟩 gtdate_	0	2	0.000 s	74.938 µs
▷ 🟩 checkprop_	1	1	58.620 µs	59.728 µs
🟩 brcastl_	31	31	324.689 µs	22.578 ms
▷ 🟩 bpres_	1	1	3.026 ms	3.150 ms
◢ 🟩 initialize_	1	1	1.583 s	1.587 s
🟩 setvec_	1	1	894.130 µs	948.701 µs
🟩 setsip_	1	1	972.842 µs	1.603 ms
▷ 🟩 setprefindx_	1	1	1.783 ms	7.207 ms
◢ 🟩 readprop_	1	1	529.421 µs	551.436 µs
🟩 brcastr8_	10	10	59.762 µs	487.891 µs
🟩 brcasti_	1	1	6.422 µs	8.351 µs
◢ 🟩 readmap_	1	1	32.387 ms	32.449 ms
▷ 🟩 sendr8_	0	9	0.000 s	16.857 ms
▷ 🟩 sendi_	0	39	0.000 s	619.185 µs
▷ 🟩 recvr8_	0	3	0.000 s	5.672 ms
▷ 🟩 recvi_	0	13	0.000 s	22.103 ms
🟩 brcastl_	1	1	5.907 µs	8.126 µs

Vampir also provides runtime filtering capabilities

1. Define a filter file in the environment variable

   export VT_FILTER_SPEC=~/myFilter

2. The filter file contains a list of functions names with a limit
3. Execute your application as usual



```
yy_get_next_buffer -- 0
yy_get_previous_state -- 0
ewdlocalize_ -- 5000
ewdlocalizei_ -- 0
ewdfeshape_ -- 67000
ewdfeshapest3d4n_ -- 0
yylex -- 0
ewdzero_ -- 0
ewdbsradd_ -- 0
ewdbsraddi_ -- 0
```