TECHNISCHE
UNIVERSITÄT
DRESDEN

Center for Information Services and High Performance Computing (ZIH)

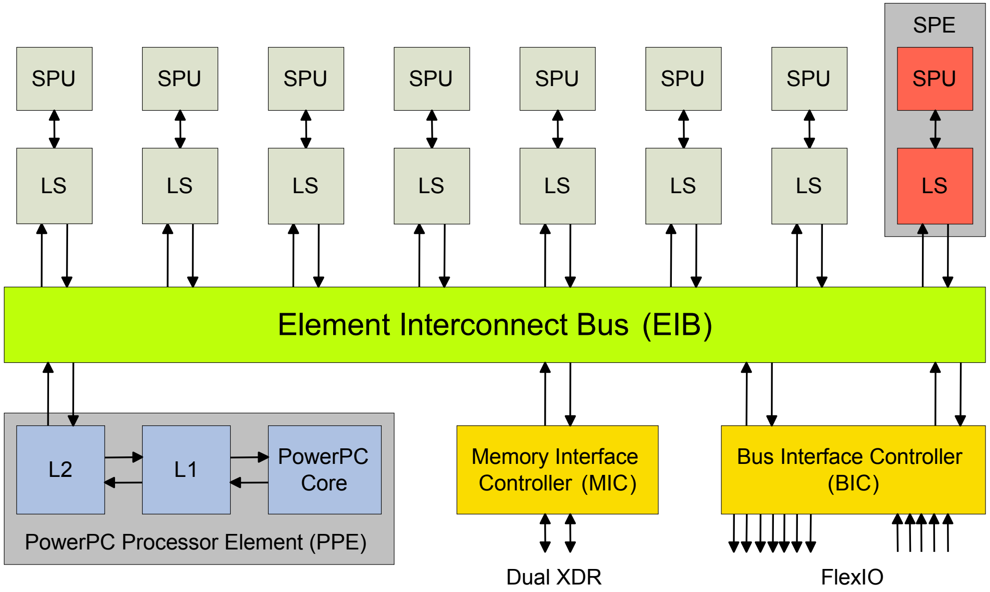
# Event Tracing and Visualization for Cell Broadband Engine Systems

Daniel Hackenberg (daniel.hackenberg@zih.tu-dresden.de)

ZIH
Center for Information Services &
High Performance Computing

# Cell Broadband Engine

- Processor offers vast resources
  - SPEs: SIMD-Cores for fast calculations, 256 KB local store (LS, software controlled), dedicated DMA engine (MFC)
  - PPE: very simple PowerPC Core for OS (Linux) and control tasks


SPE
SPU
SPU
SPU
SPU
SPU
SPU
SPU
SPU
LS
LS
LS
LS
LS
LS
LS
LS
Element Interconnect Bus (EIB)
L2
L1
PowerPC Core
PowerPC Processor Element (PPE)
Memory Interface Controller (MIC)
Bus Interface Controller (BIC)
Dual XDR
FlexIO

- Sophisticated architecture results in complex software development process
  - Different compilers and programs for PPE and SPEs
  - SPEs use DMA commands to access main memory or LS of other SPEs, asynchronous execution by MFC
  - Mailbox communication between PPE and SPEs
- Tool support for software development and performance analysis required

# Software Tracing and Vampir

- Proven method for analysis of complex programs
- Instrumented target application creates events with timestamps at runtime
- Events are stored in traces, trace analysis e.g. by visualization

- VampirTrace: open source trace monitor
  - Supports MPI, OpenMP, regions, hardware counters
  - Creates program traces in Open Trace Format (OTF)

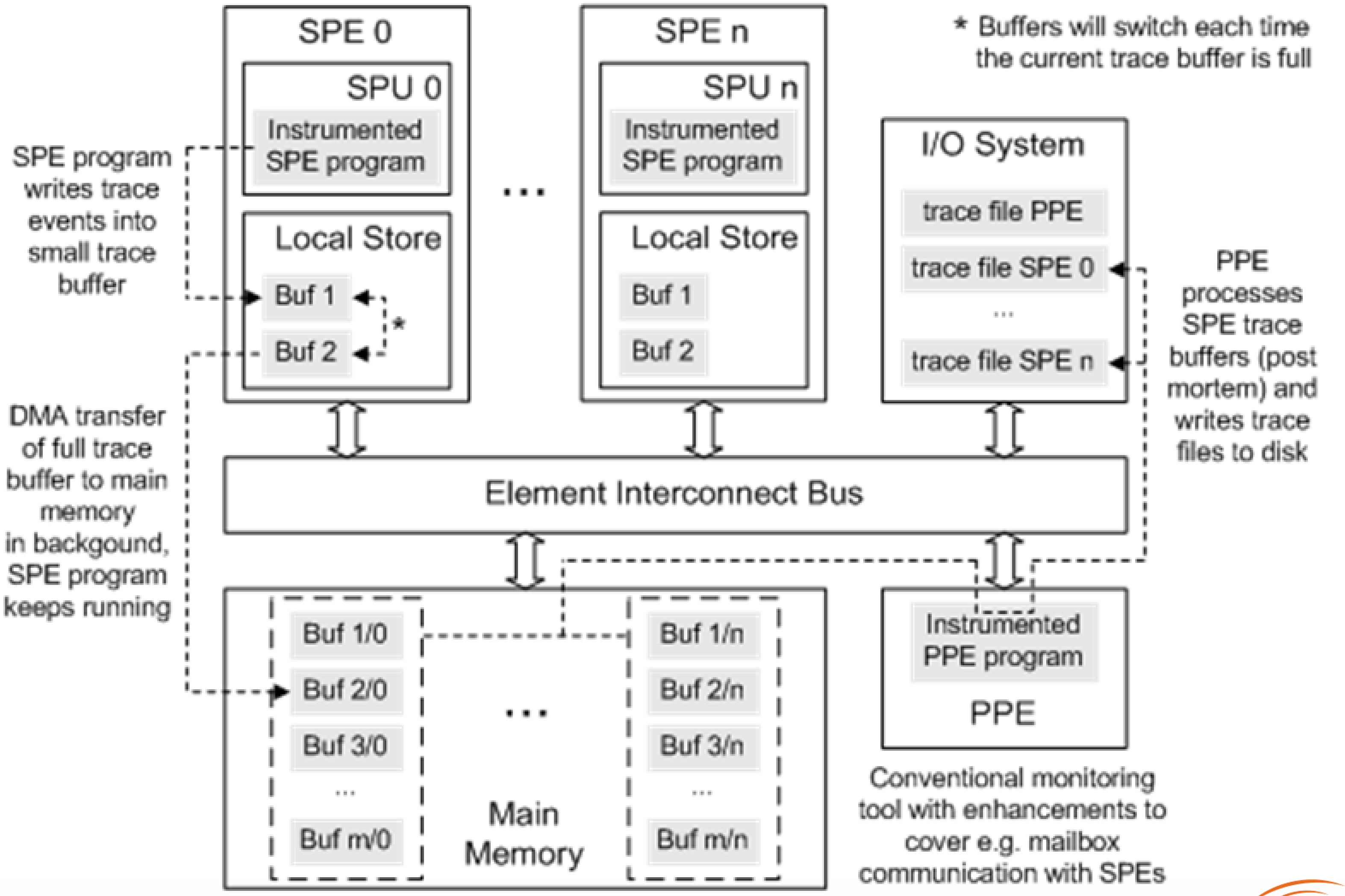
- Vampir: visualization and analysis of trace data
  - Various displays (e.g. timelines) and means for statistical analysis
  - Parallel version supports ultra large program traces

# Software Tracing on Cell/B.E. Systems

- PPE
  - Conventional tools with PowerPC support run unmodified
  - Modifications necessary to support SPE threads

- SPE
  - New concept needs to be designed, suitable for this architecture
  - New monitor necessary to generate events
  - Local store too small, only temporary storage of events
  - Synchronization of PPE and SPE timers necessary

# Trace Concept for the Cell B./E. Architecture



* Buffers will switch each time the current trace buffer is full

SPE program writes trace events into small trace buffer

DMA transfer of full trace buffer to main memory in backgound, SPE program keeps running

PPE processes SPE trace buffers (post mortem) and writes trace files to disk

Conventional monitoring tool with enhancements to cover e.g. mailbox communication with SPEs

# Trace Visualization for Cell (1)

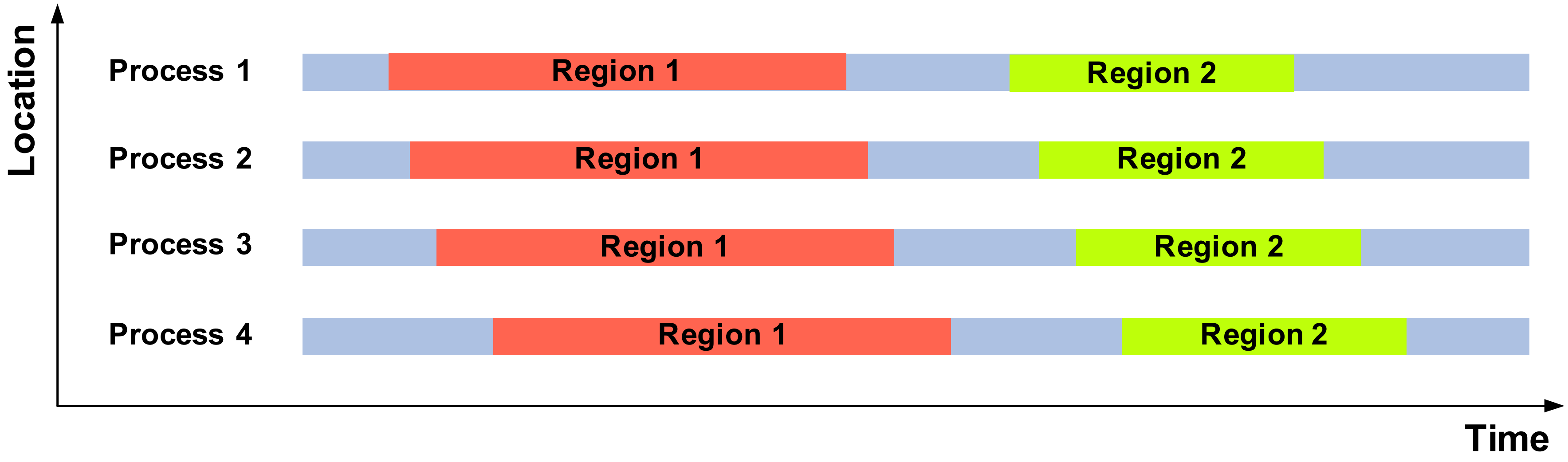


Illustration of parallel processes in a classic timeline display

# Trace Visualization for Cell (2)

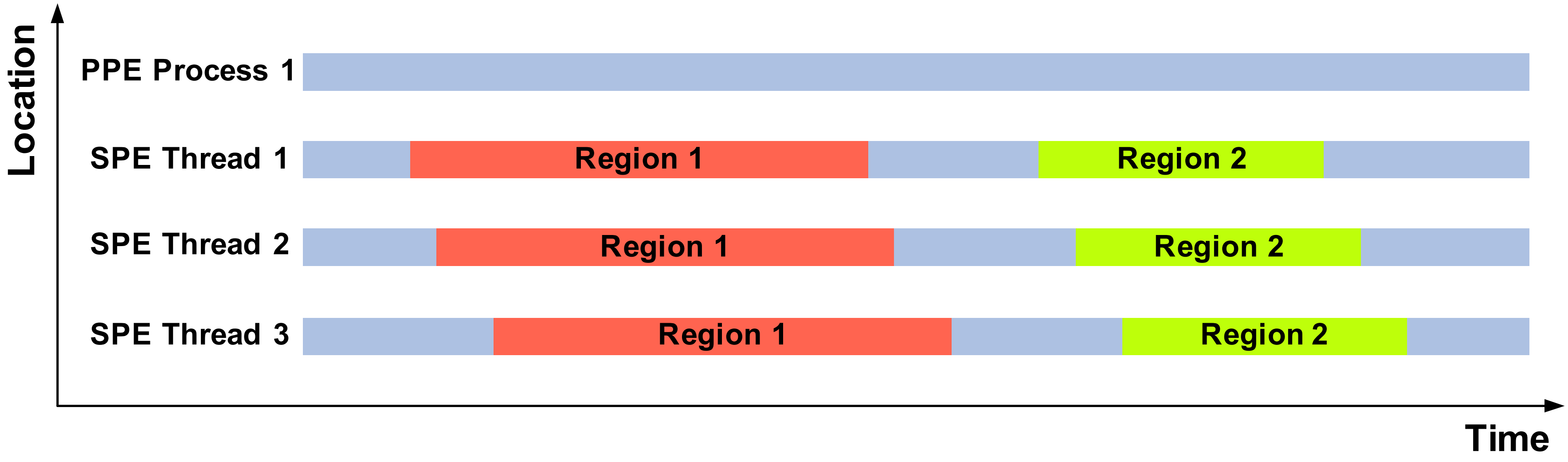


Illustration of SPE threads as children of the PPE process

# Trace Visualization for Cell (3)

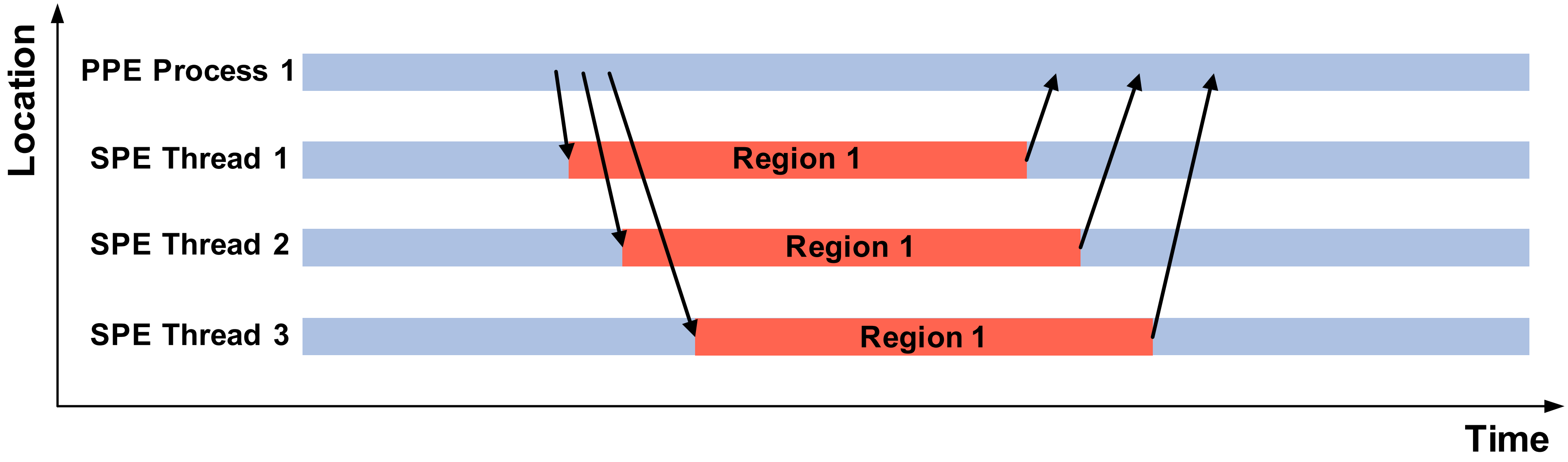


Illustration of mailbox messages

- Classic two-sided communication (send/receive)
- Illustrated by lines similar to MPI messages

# Trace Visualization for Cell (4)

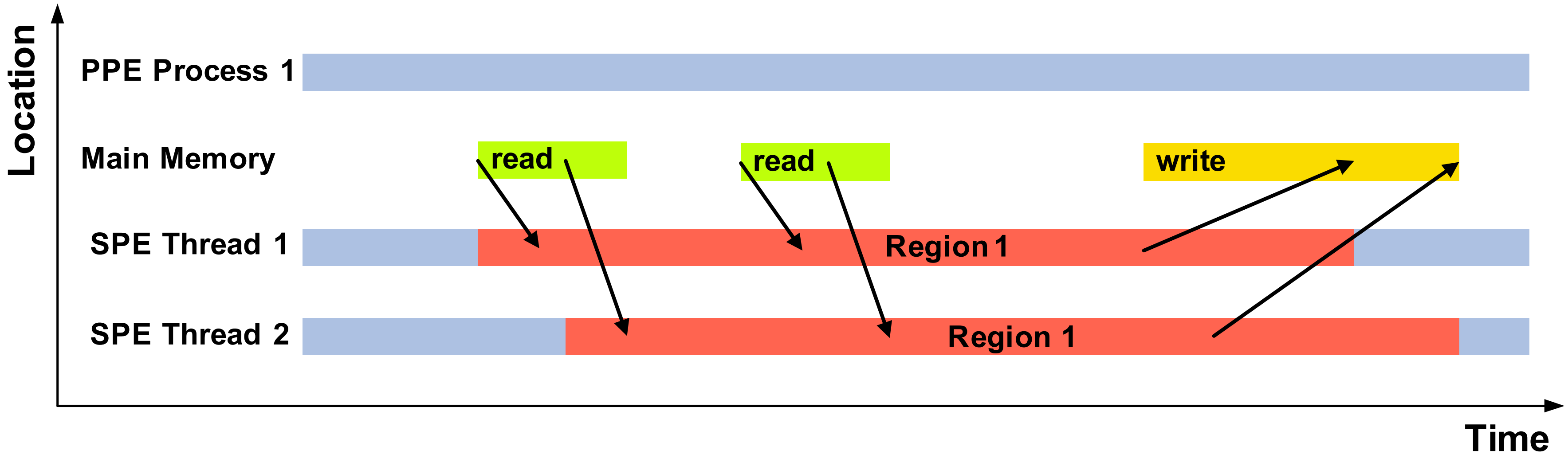
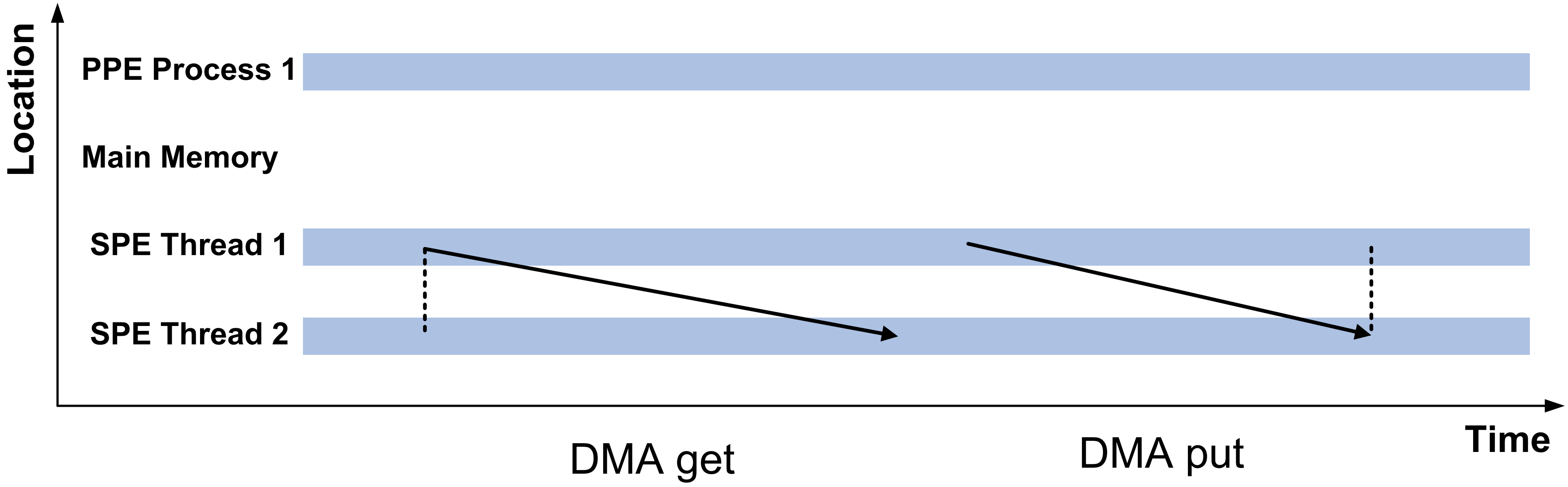


Illustration of DMA transfers between SPEs and main memory

- Virtual process bar represents the main memory
- Illustration of main memory state possible (read/write)

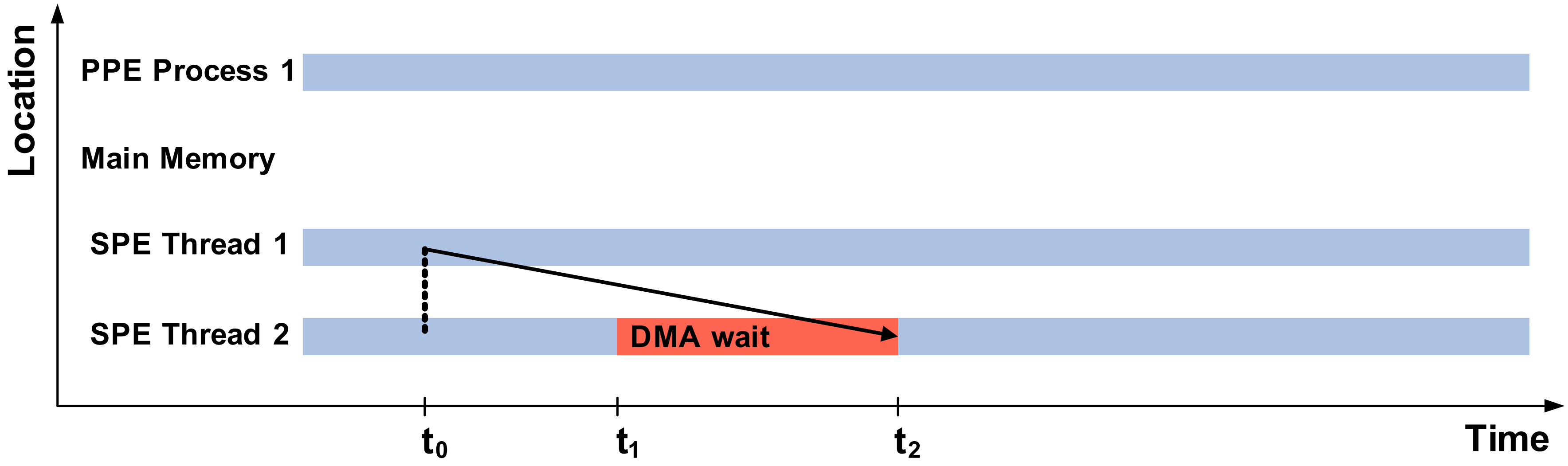
# Trace Visualization for Cell (5)



DMA transfers between SPEs

- Classic send/receive representation unsuitable
- Additional line allows distinction of active and passive partner

# Trace Visualization for Cell (6)



```
t_0 = get_timestamp();
mfc_get();
[...]
t_1 = get_timestamp();
wait_for_dma_tag();
t_2 = get_timestamp();
```

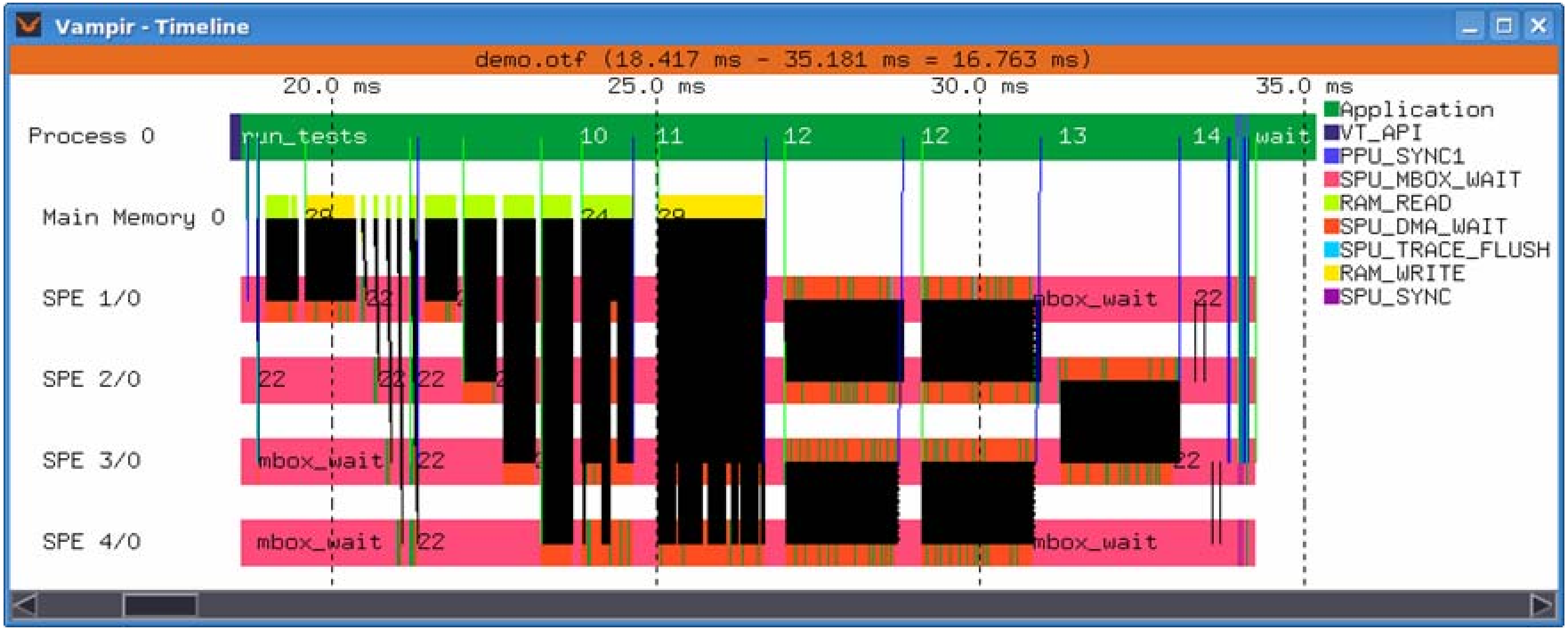
- DMA wait operation creates two events (at $t_1$ and $t_2$)
- Allows illustration of DMA wait time
- Similar for mailbox messages

# Implementation

- Beta version VampirTrace (VT) with Cell support
  - Open Source trace monitor
  - Compiler wrappers (vtcc and vtspucc) will do most of the work for you
  - Header files for PPE and SPE programs: Instrumentation of inline functions provided by the Cell SDK
  - Manual instrumentation of important SPE code regions for low overhead
  - Tracing of hybrid Cell/MPI parallel applications supported

- Trace analyzer Vampir: Technology study with support for Cell traces available

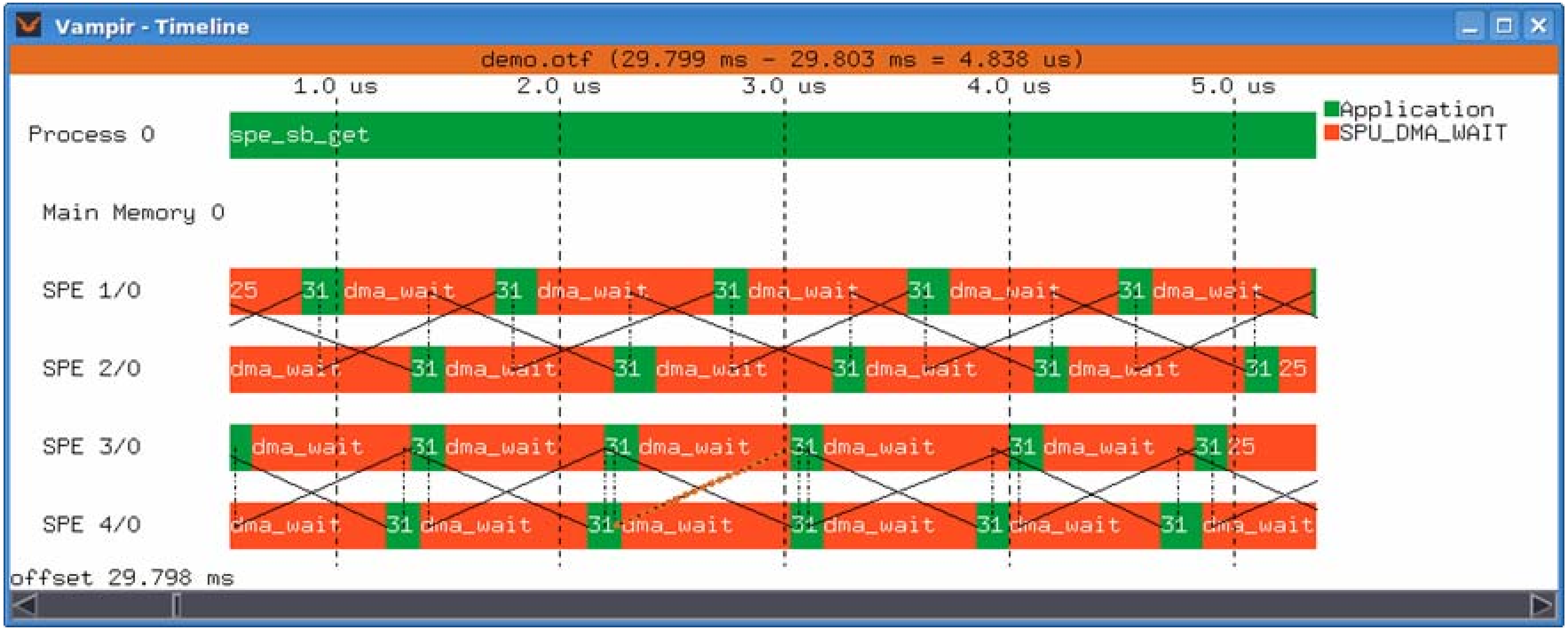
# Trace Visualization with Vampir (1)



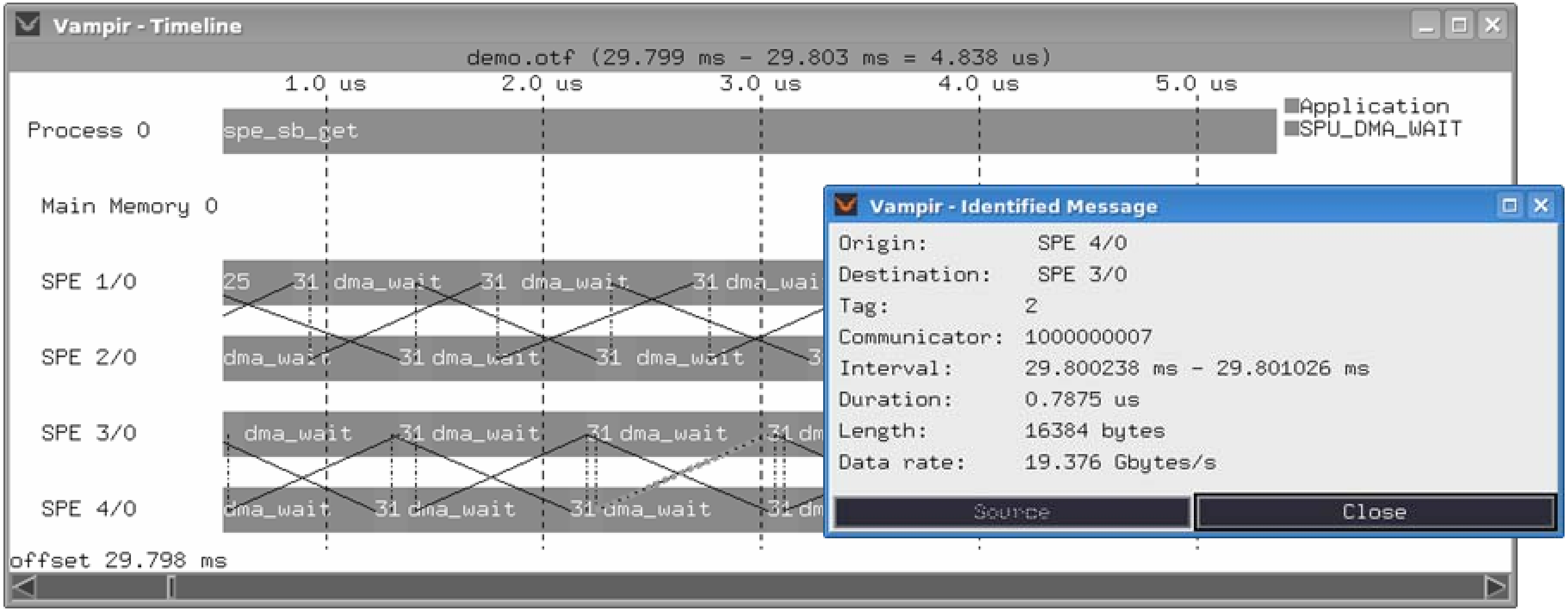
Visualization of a Cell trace using Vampir

Demo program using 4 SPEs

# Trace Visualization with Vampir (2)

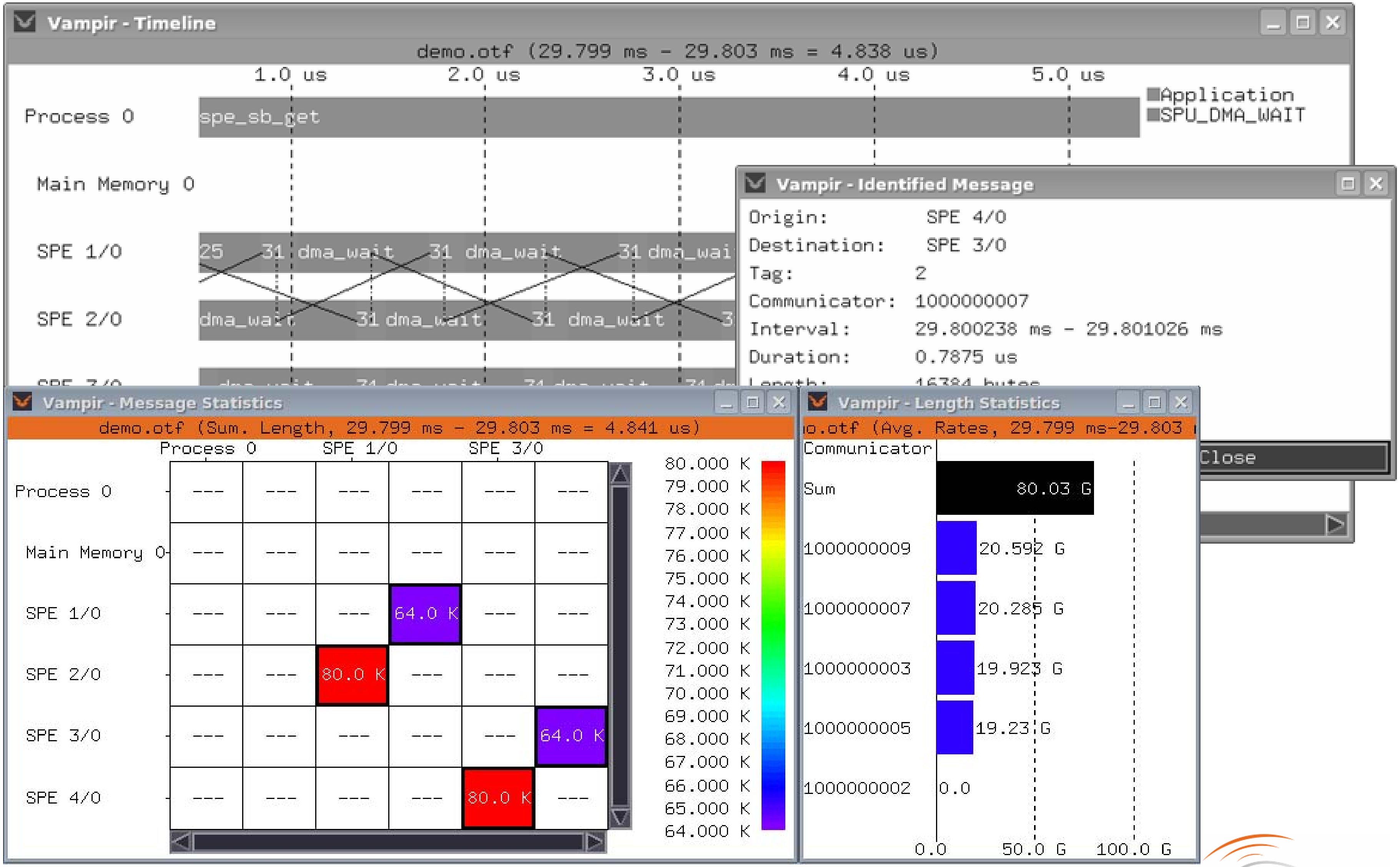


# Trace Visualization with Vampir (3)



Origin: SPE 4/0
Destination: SPE 3/0
Tag: 2
Communicator: 1000000007
Interval: 29.800238 ms - 29.801026 ms
Duration: 0.7875 us
Length: 16384 bytes
Data rate: 19.376 Gbytes/s

# Trace Visualization with Vampir (4)



Vampir - Identified Message

Origin: SPE 4/0
Destination: SPE 3/0
Tag: 2
Communicator: 1000000007
Interval: 29.800238 ms - 29.801026 ms
Duration: 0.7875 us
Length: 16384 bytes

Vampir - Message Statistics

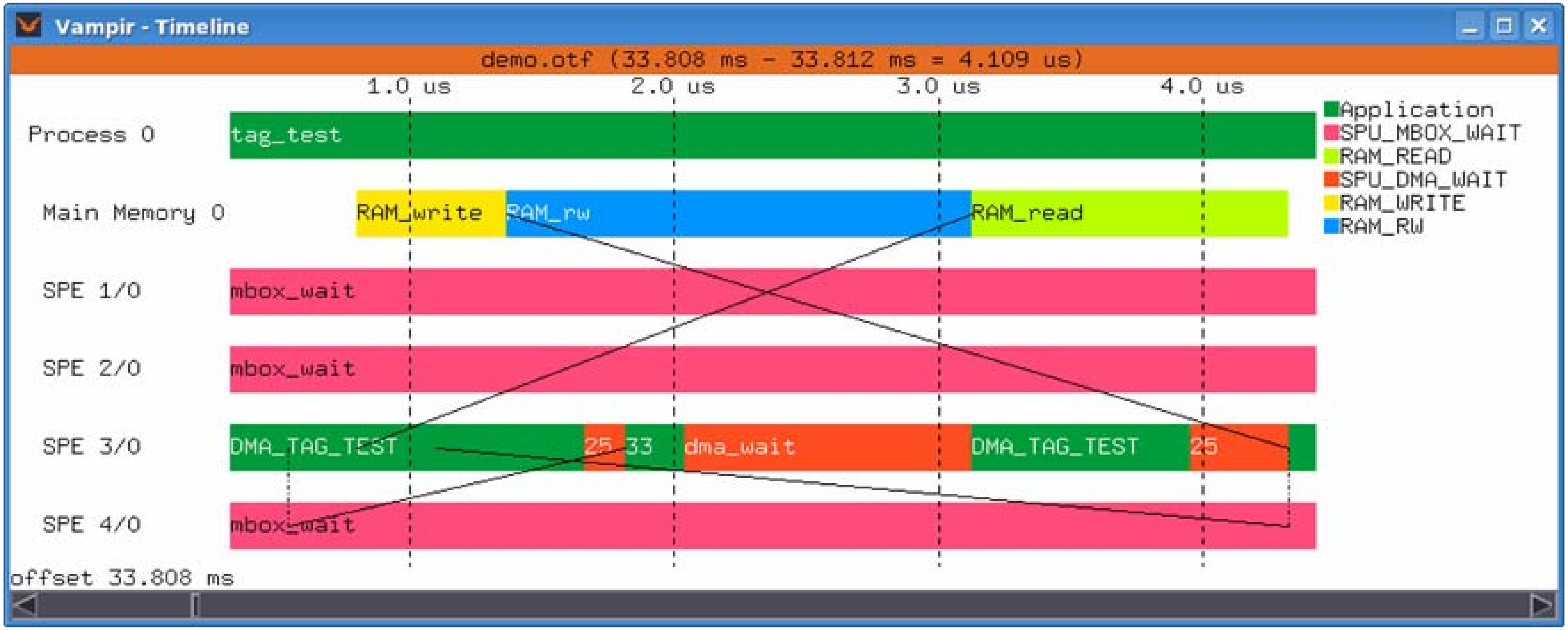
demo.otf (Sum. Length, 29.799 ms - 29.803 ms = 4.841 us)

|  | Process 0 |  | SPE 1/0 |  | SPE 3/0 |  |
|---|---|---|---|---|---|---|
| Process 0 | --- | --- | --- | --- | --- | --- |
| Main Memory 0 | --- | --- | --- | --- | --- | --- |
| SPE 1/0 | --- | --- | --- | 64.0 K | --- | --- |
| SPE 2/0 | --- | --- | 80.0 K | --- | --- | --- |
| SPE 3/0 | --- | --- | --- | --- | --- | 64.0 K |
| SPE 4/0 | --- | --- | --- | --- | 80.0 K | --- |

Vampir - Length Statistics

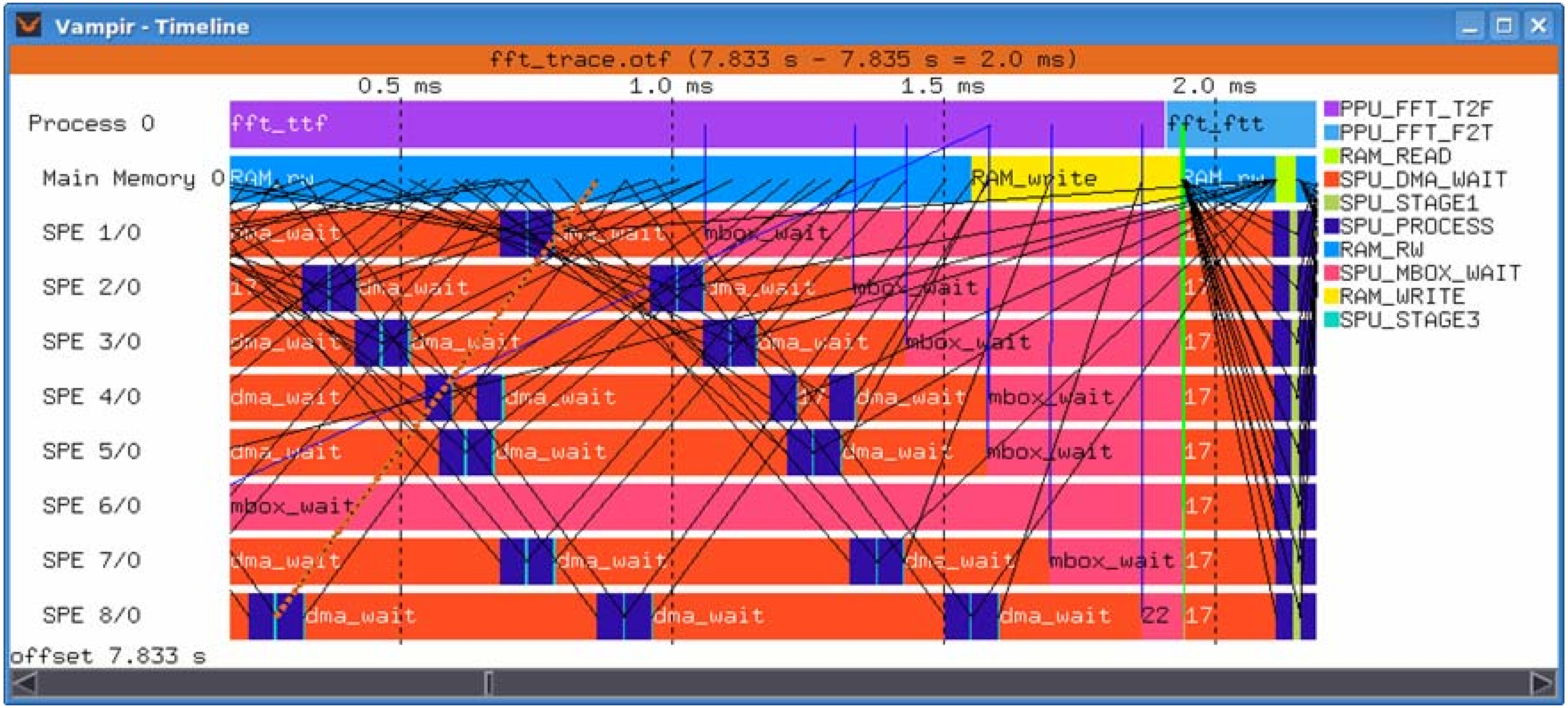
o.otf (Avg. Rates, 29.799 ms-29.803

# Trace Visualization with Vampir (5)



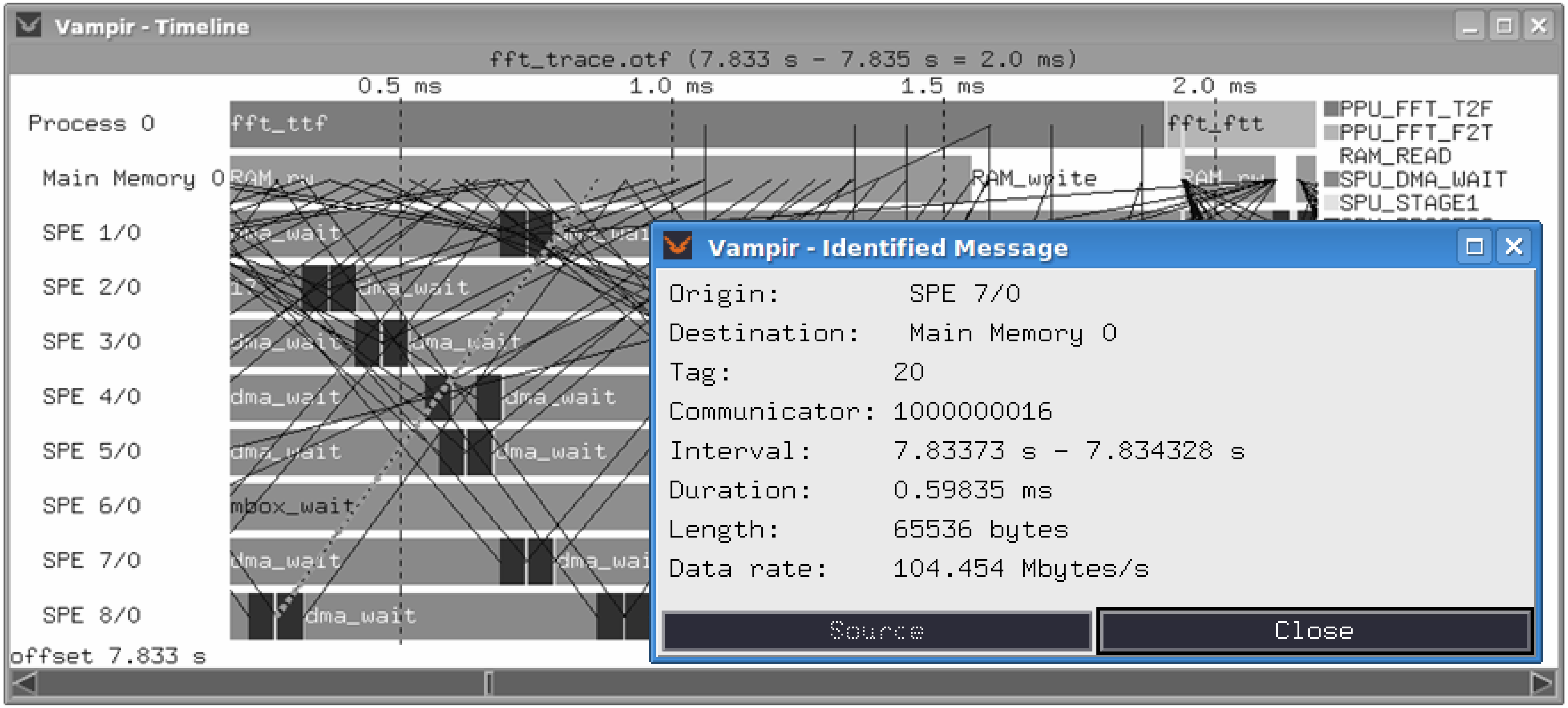
Complex DMA transfers of SPE 3

# Tracing Complex Cell Applications: FFT (1)



FFT at synchronization point
8 SPEs, 64 KByte page size, 11.9 GFLOPS

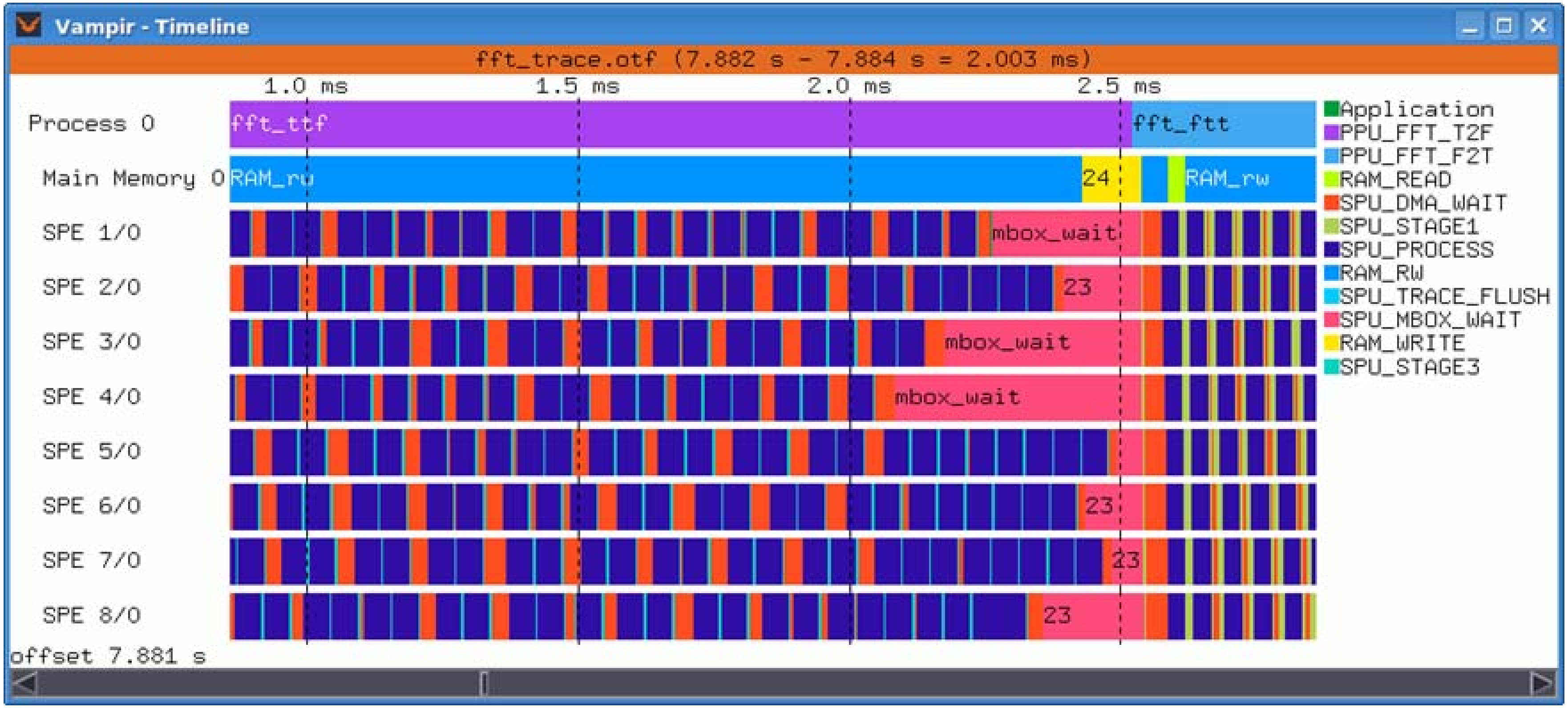
# Tracing Complex Cell Applications: FFT (2)



Origin: SPE 7/0
Destination: Main Memory 0
Tag: 20
Communicator: 1000000016
Interval: 7.83373 s - 7.834328 s
Duration: 0.59835 ms
Length: 65536 bytes
Data rate: 104.454 Mbytes/s

FFT at synchronization point
8 SPEs, 64 KByte page size, 11.9 GFLOPS

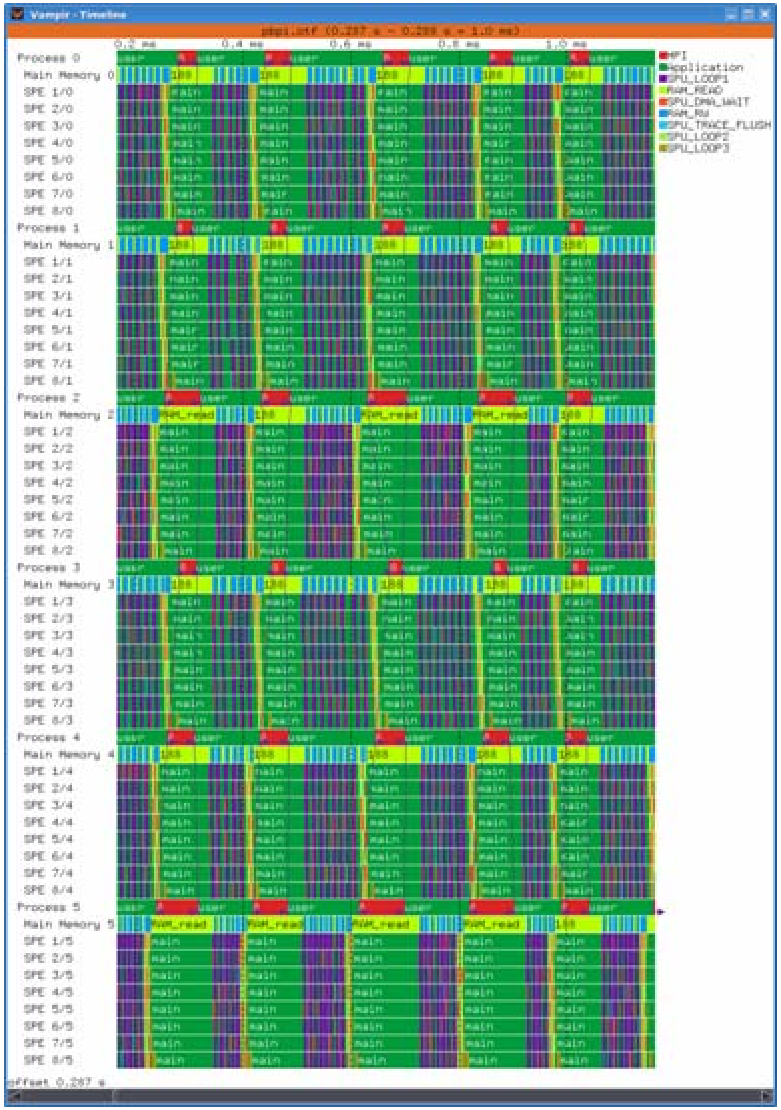
# Tracing Complex Cell Applications: FFT (3)

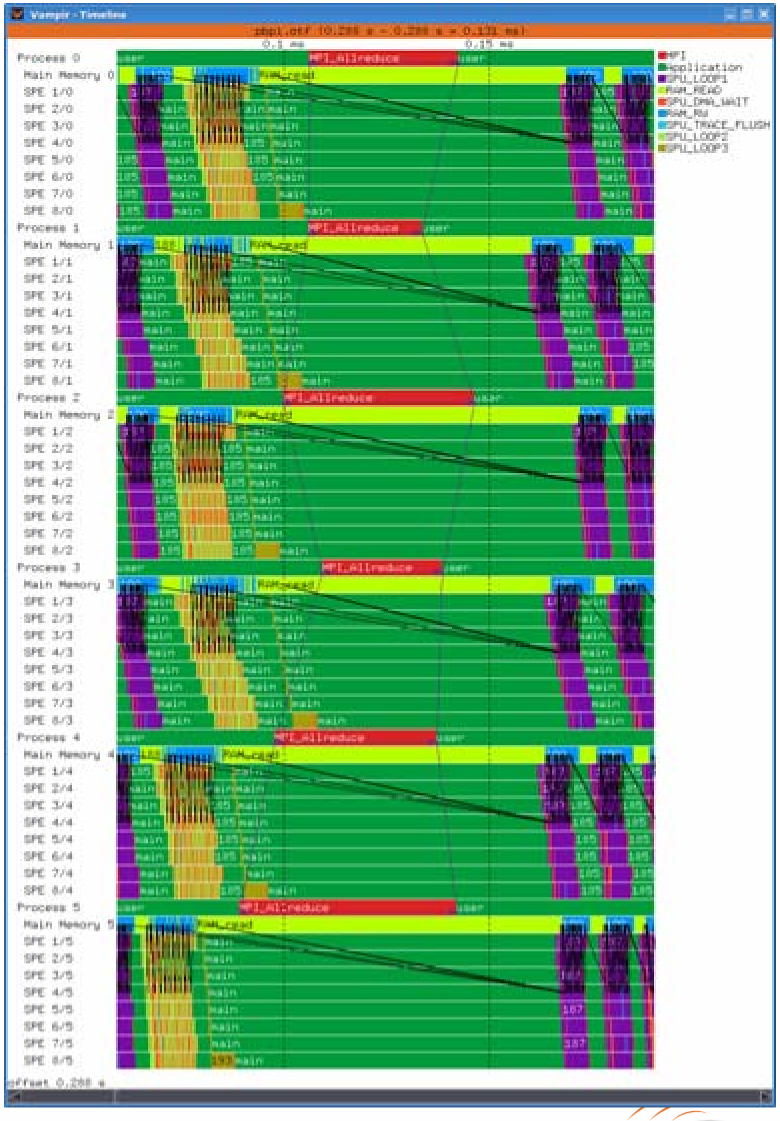


FFT at synchronization point

8 SPEs, 16 MByte page size, 42.9 GFLOPS

# Tracing Hybrid Cell/MPI Applications: PBPI





PBPI (Parallel Bayesian Phylogenetic Inference)
on 3 QS21 blades (6 Cell processors)

# Cell Tracing Overhead

- Overhead sources
  - Creating events
  - Transferring trace data from the SPEs to main memory
  - Trace buffer und trace library use space in local store (< 12 KByte)
- Additional overhead (VampirTrace initialization and processing of SPE event data) outside of SPE runtime → Analysis unaffected
- Experimental overhead measurements (QS21, 8 SPEs)

| | Original (GFLOPS) | Tracing (GFLOPS) | Overhead |
|---|---|---|---|
| SGEMM | 203,25 | 200,73 | 1,3 % |
| FFT | 11,93 | 11,85 | 0,7 % |
| Cholesky, SPOTRF | 143,17 | 139,32 | 2,8 % |
| Cholesky, DGEMM | 4,48 | 4,10 | 9,2 % (*) |
| Cholesky, STRSM | 5,73 | 5,64 | 1,7 % |

(*) Increased overhead due to intense usage of DMA lists
Trace overhead without DMAs: 1,4 %

# Summary & Future Work

- Concept for software tracing on Cell systems presented
- VampirTrace Beta with Cell support, typical overhead < 5 percent
- Visualization of traces with Vampir
  - Creates invaluable insight into the runtime behavior of Cell applications
  - Intuitive performance analysis and optimization
- Support for large, hybrid Cell/MPI applications
- Future work may include:
  - Improved tracing, e.g. by providing additional analysis features such as alignment checks
  - Improved visualization, e.g. by colorizing DMA messages (tag, size or bandwidth), displaying intensity of main memory accesses