TECHNISCHE UNIVERSITÄT DRESDEN

Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH)

Über die Cell/B.E.-Architektur: Optionen zur Generierung von Programm-Traces

Diplomverteidigung

Nöthnitzer Straße 46

Informatik, Raum 1027

Tel. +49 351 - 463 - 38537

Verantwortlicher Hochschullehrer:

Prof. Dr. Wolfgang E. Nagel

Betreuer: Dr. Holger Brunst

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

ZIH

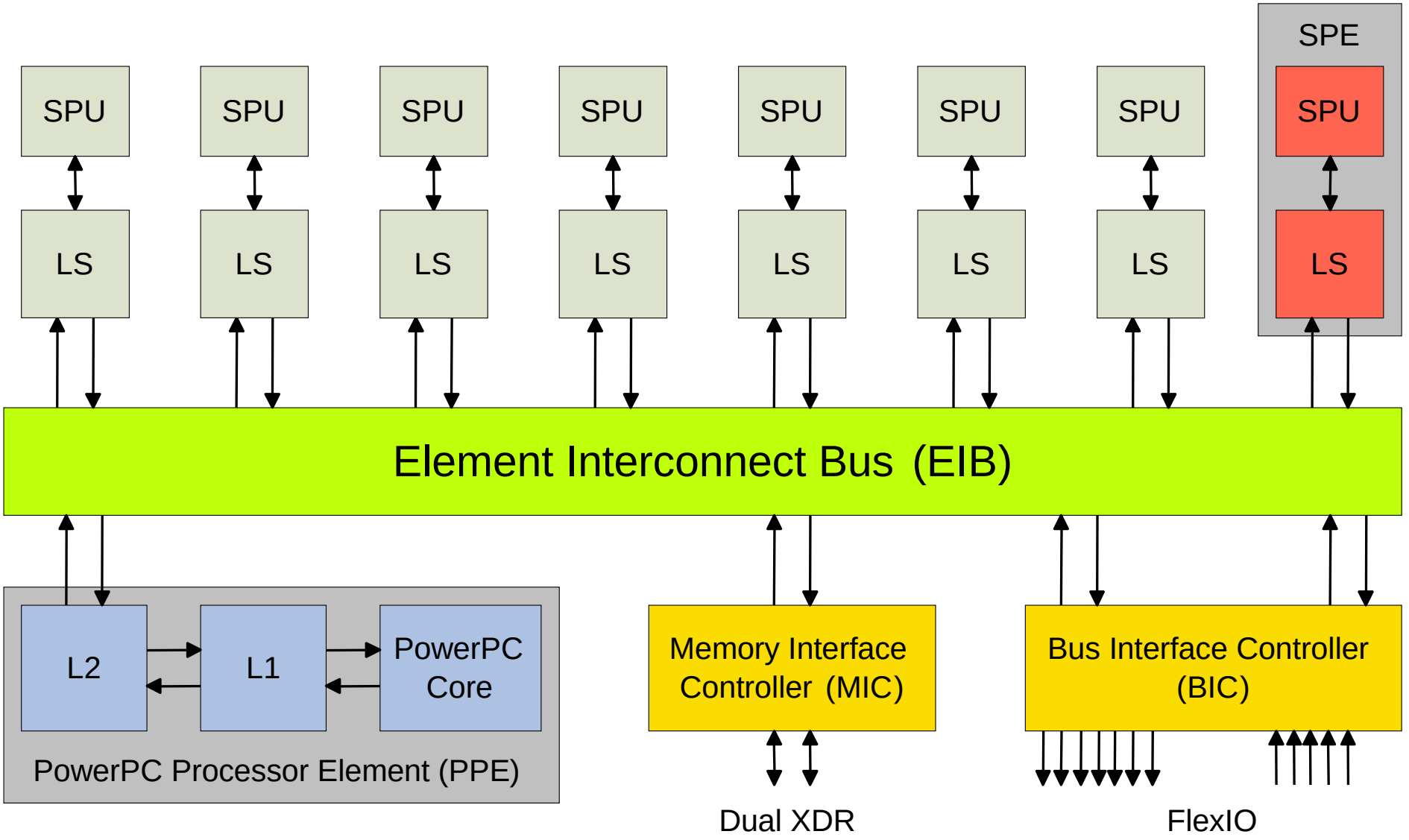
Center for Information Services & High Performance Computing

Gliederung

- Einleitung
- Software-Tracing auf Cell-Prozessoren
- Implementierung und Funktionalität
- Ergebnisse und Overhead-Betrachtung
- Zusammenfassung und Ausblick

- **Einleitung**
- Software-Tracing auf Cell-Prozessoren
- Implementierung und Funktionalität
- Ergebnisse und Overhead-Betrachtung
- Zusammenfassung und Ausblick

Cell Broadband Engine



SPE: Synergistic Processor Element

LS: Local Store

Cell Broadband Engine

- Prozessor stellt erhebliche Ressourcen bereit
  - SPEs: SIMD-Kerne für schnelle Berechnungen, 256 KB lokaler Speicher (LS) unter Software-Kontrolle, dedizierter DMA-Controller (MFC)
  - PPE: sehr einfacher Kern für Betriebssystem (Linux) und Kontroll-Tasks
  - Kohärente SMP-Konfiguration mit zwei CPUs möglich: 2 PPEs und 16 SPEs
- Komplexe Architektur bedingt aufwendige Programmierung
  - Verschiedene Compiler und Programme für PPE und SPEs
  - SPEs greifen mit DMA-Kommandos auf Hauptspeicher oder lokalen Speicher anderer SPEs zu, asynchrone Bearbeitung durch MFC
  - Mailbox-Kommunikation zwischen PPE and SPEs möglich
- Erleichterung der Software-Entwicklung durch geeignete Werkzeuge notwendig

Software-Tracing

- Bewährte Methode zur Analyse komplexer Programme
- Modifizierte Anwendung erzeugt zur Laufzeit Ereignisse mit Zeitstempel
- Typische Ereignisse:
  - Ein-/Austritt von Funktionen bzw. Regionen
  - Nachrichten zwischen parallelen Prozessen, z. B. durch MPI
  - andere Monitoring-Daten, z. B. aus Performance-Countern
- Ereignisse werden in „Traces“ gespeichert
- Auswertung der Traces z. B. durch Visualisierung

Vampir

- VampirTrace: freies Trace-Werkzeug
  - Unterstützt u.a. MPI, OpenMP, Regionen, Hardware-Counter
  - Erzeugt Programm-Traces im Open Trace Format (OTF)
  - Trace-Analyse mit Vampir möglich

- Vampir: Visualisierung und Auswertung von Trace-Daten
  - Verschiedene Displays (z. B. Timelines)
  - Verschiedene statistische Auswertungen
  - Analyse der Effizienz von Berechnungen und Kommunikation
  - Parallele Version unterstützt extrem große Programm-Traces

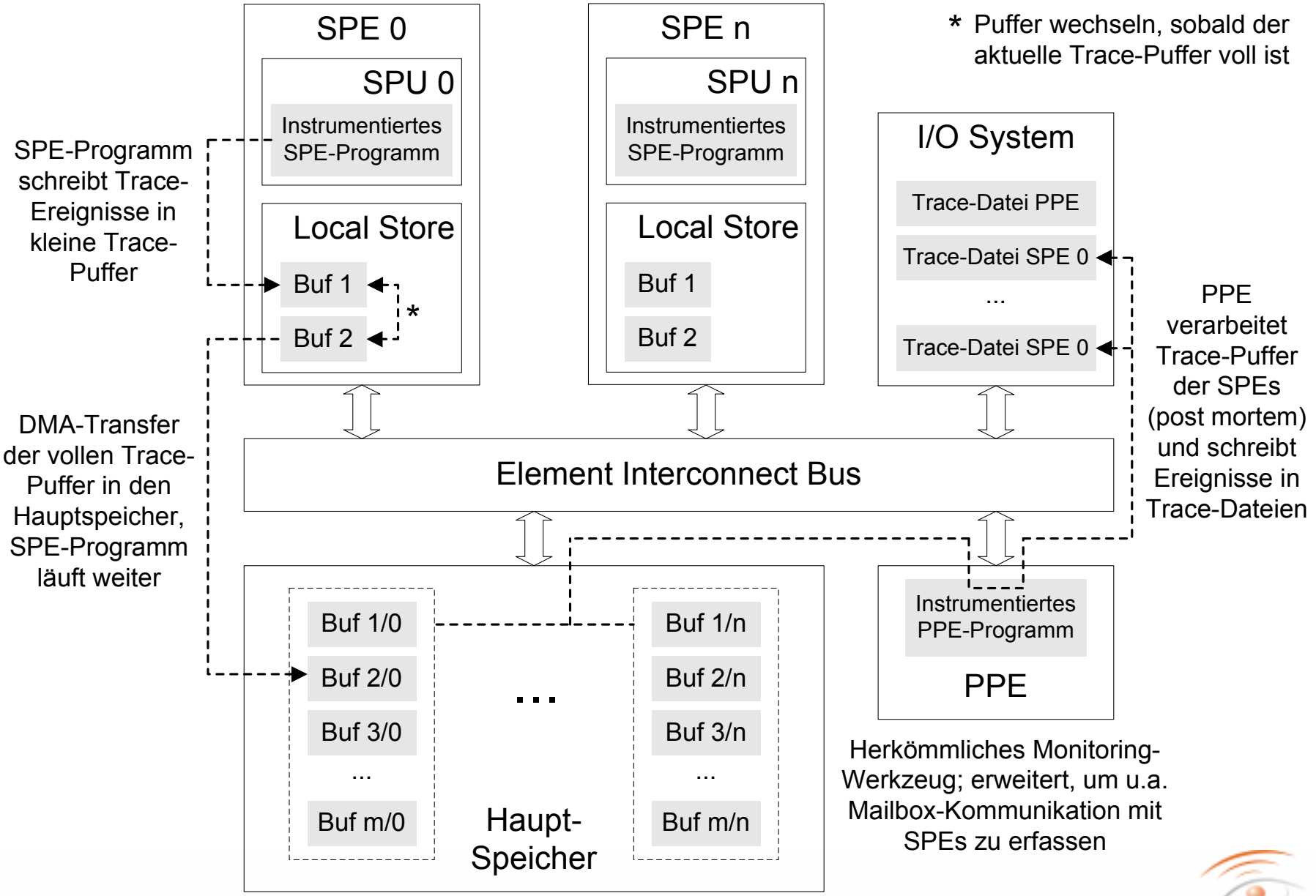
- Einleitung
- **Software-Tracing auf Cell-Prozessoren**
- Implementierung und Funktionalität
- Ergebnisse und Overhead-Betrachtung
- Zusammenfassung und Ausblick

Software-Tracing auf Cell-Prozessoren

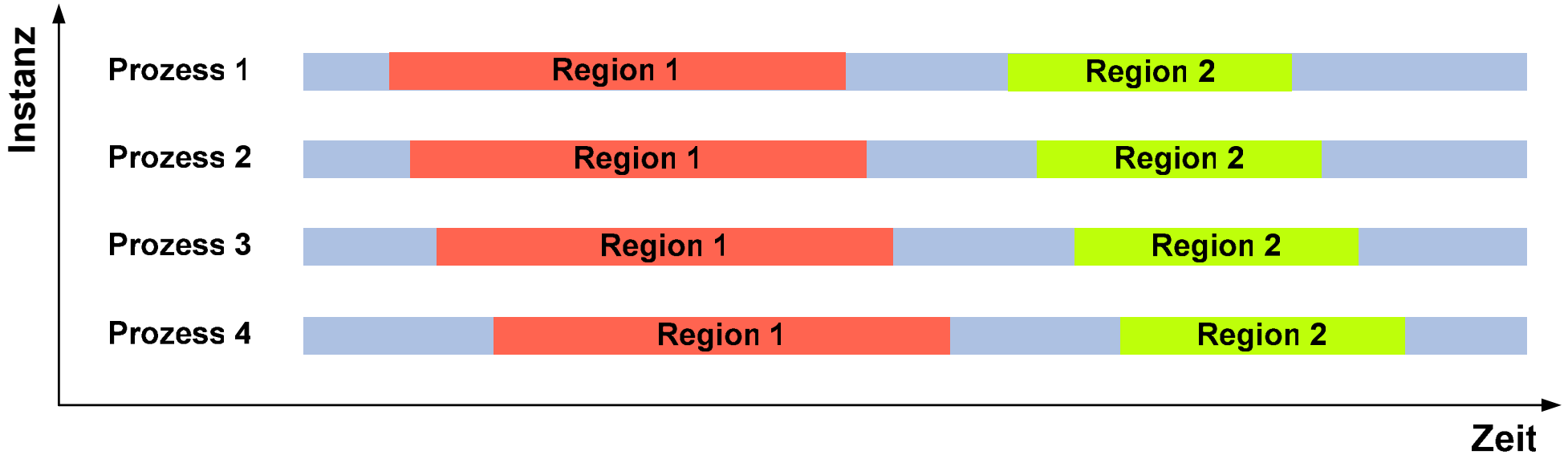
- PPE-seitig
  - Bewährte Werkzeuge mit PowerPC-Unterstützung voll lauffähig
  - Modifikationen zur Unterstützung von SPE-Threads notwendig

- SPE-seitig
  - Neues Konzept benötigt, passend zur Architektur
  - Geeignete Instrumentierung erzeugt Ereignisse
  - Lokaler Speicher zu klein, nur Zwischenspeicherung der Ereignisse
  - Übertragung der Daten in den Hauptspeicher per DMA möglich

Trace-Konzept für die Cell-Architektur

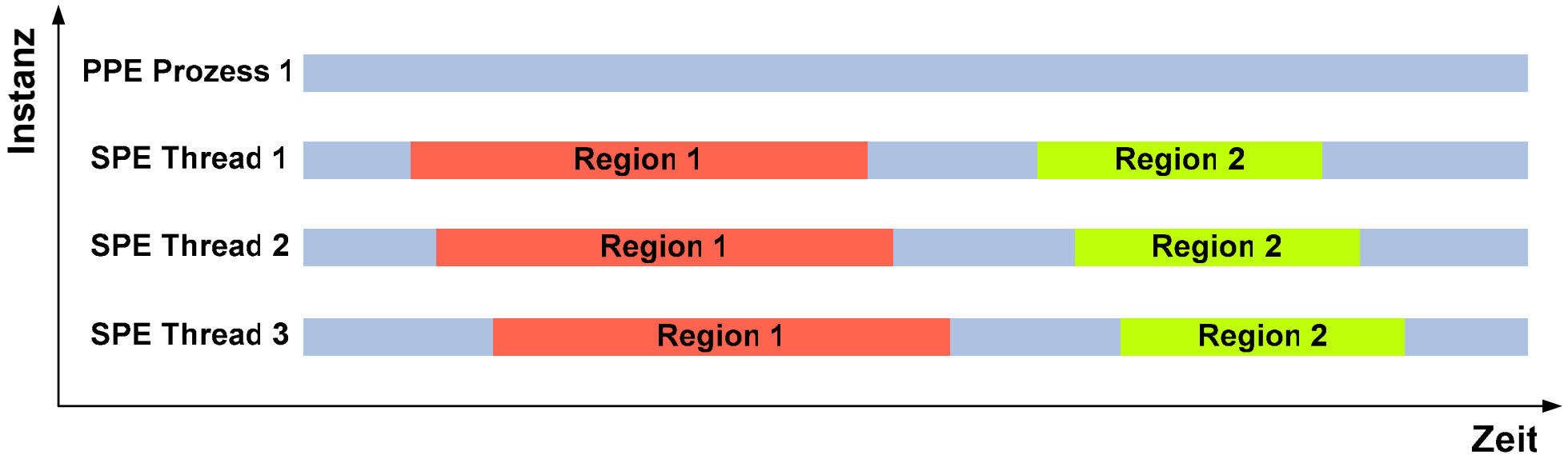


Trace-Visualisierung für die Cell-Architektur (1)



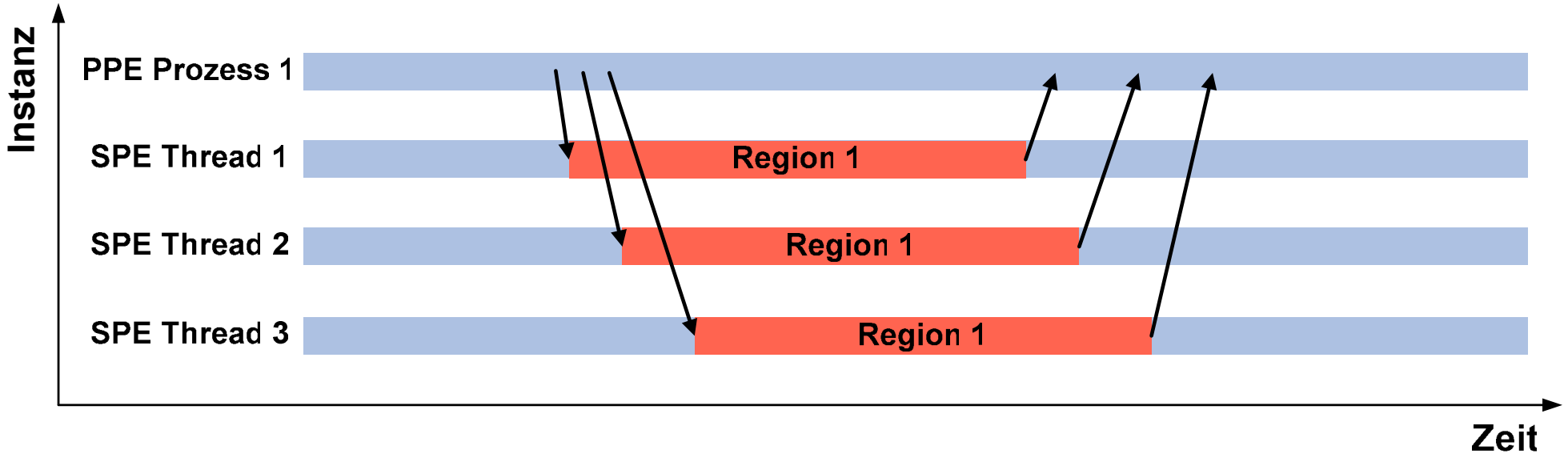
Klassische Timeline-Darstellung von parallelen Programmen

Trace-Visualisierung für die Cell-Architektur (2)



Darstellung der SPE-Threads als „Kinder“ des PPE-Prozesses

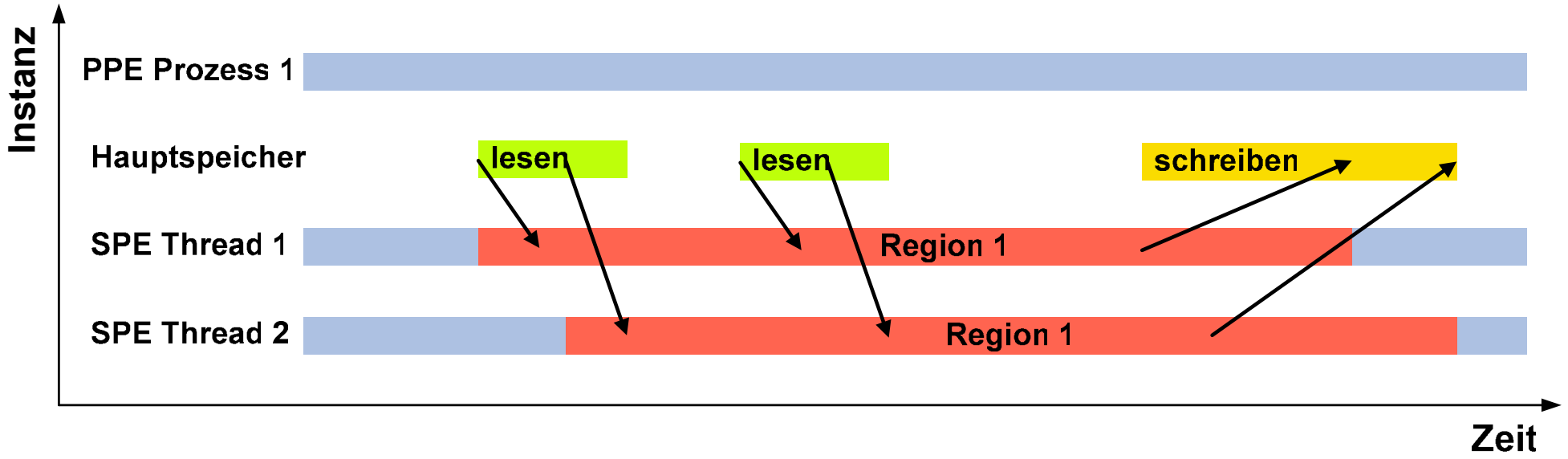
Trace-Visualisierung für die Cell-Architektur (3)



Darstellung von Mailbox-Übertragungen

- Klassische zweiseitige Kommunikation (send/receive)
- Analog zu MPI-Nachrichten durch Linien/Pfeile abgebildet

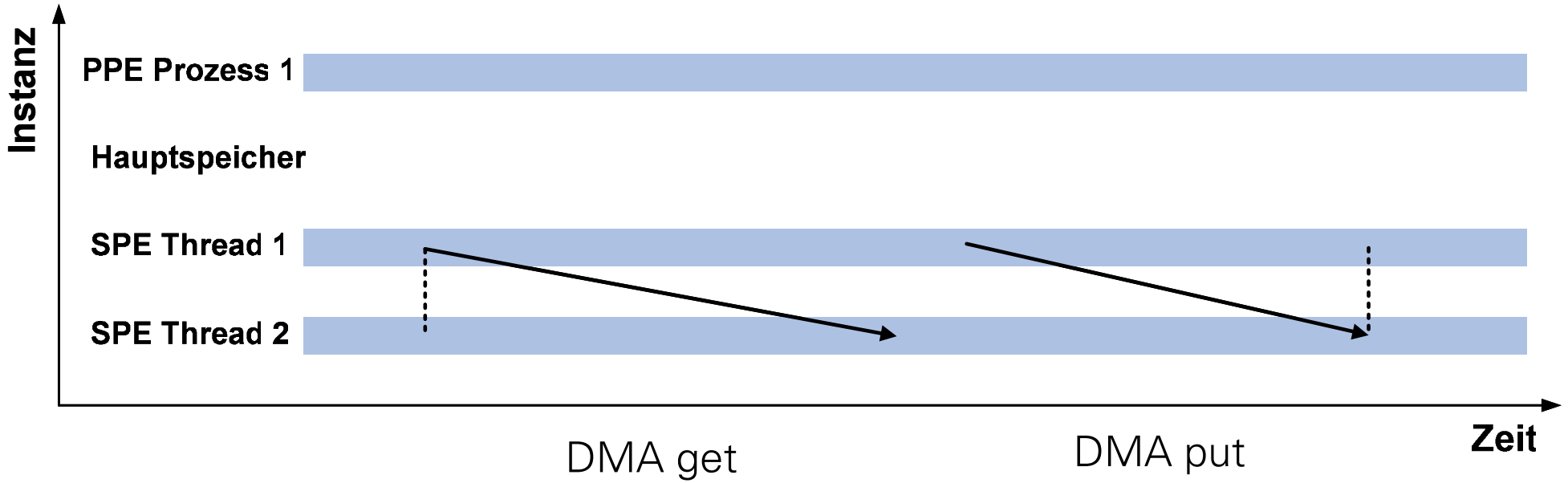
Trace-Visualisierung für die Cell-Architektur (4)



Darstellung von DMA-Zugriffen der SPEs auf den Hauptspeicher

- Virtueller Prozess-Balken repräsentiert den Hauptspeicher
- Spezielle Markierung der Hauptspeicherzugriffe möglich

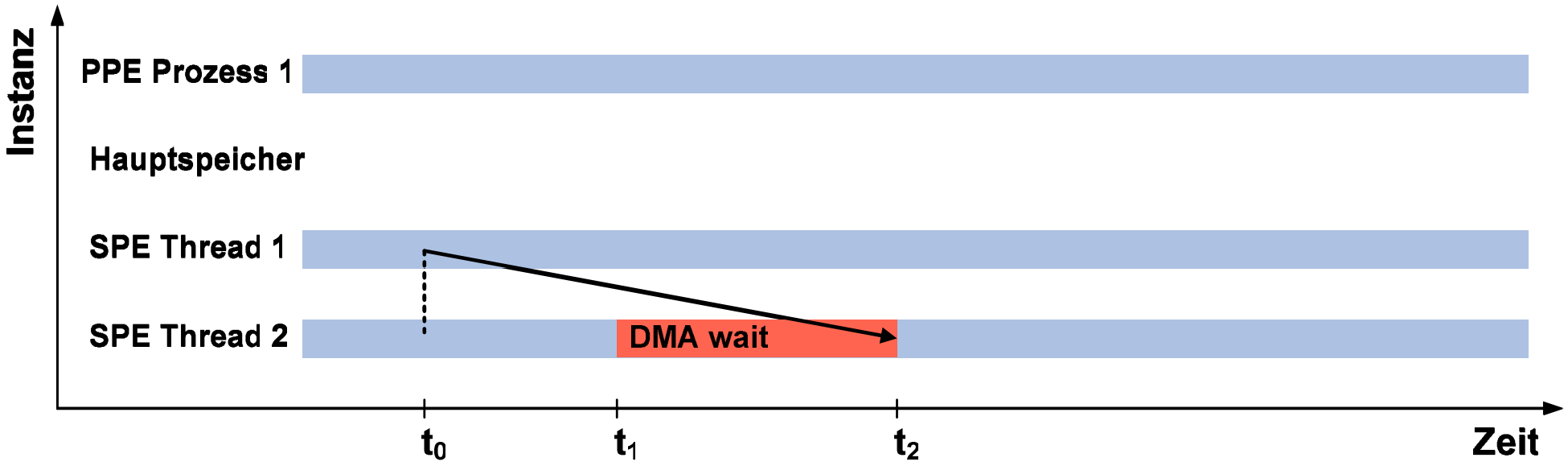
Trace-Visualisierung für die Cell-Architektur (5)



DMA-Transfers zwischen SPEs

- Klassische Send/Receive-Darstellung ungeeignet
- Zusätzliche Linie ermöglicht Unterscheidung von aktivem und passivem Partner

Trace-Visualisierung für die Cell-Architektur (6)




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

- Warte-Operation bei DMA-Transfers erzeugt zwei Ereignisse (bei t_1 und t_2)
- Ermöglicht Markierung der Wartezeit
- Analoge Vorgehensweise bei Mailbox-Nachrichten

- Einleitung
- Software-Tracing auf Cell-Prozessoren
- **Implementierung und Funktionalität**
- Ergebnisse und Overhead-Betrachtung
- Zusammenfassung und Ausblick

Implementierung (1)

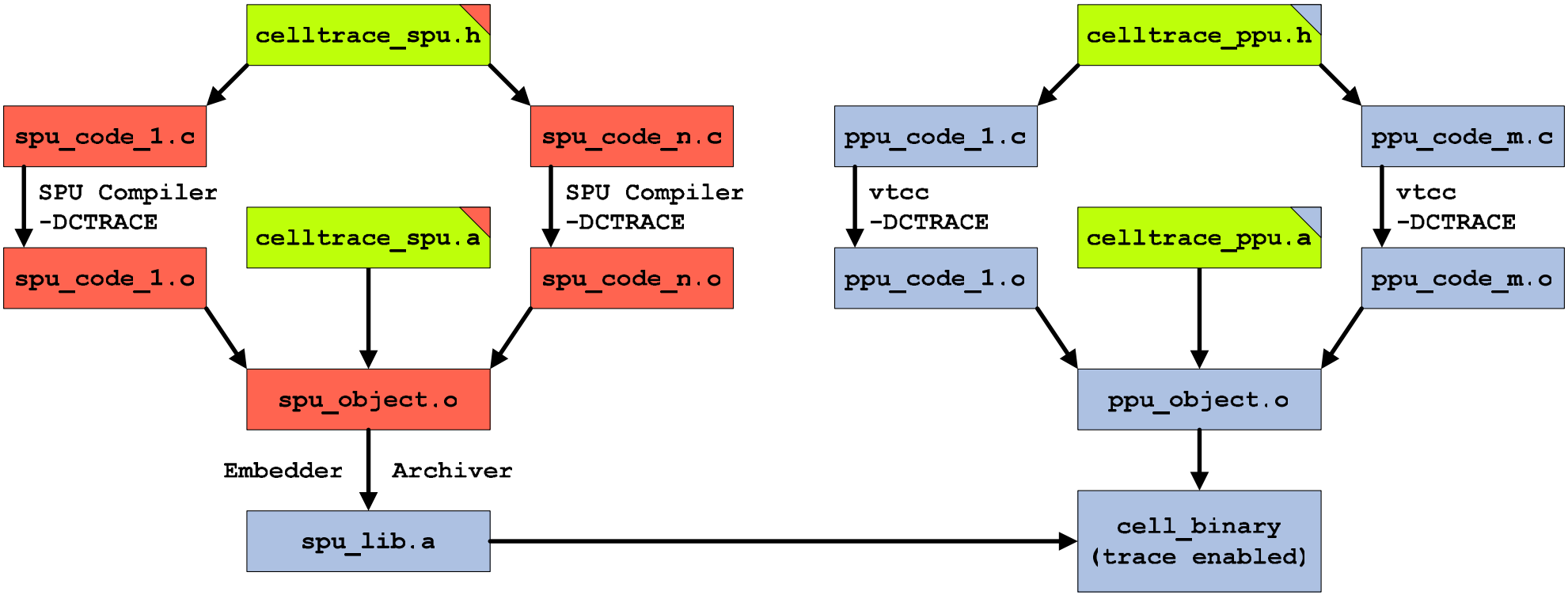
- Beispielhafte Implementierung des Konzepts basiert auf VampirTrace (VT)
- VT-Modifikationen erforderlich, beispielsweise:
  - Schnittstellen zur SPE-Thread-Erzeugung und Initialisierung
  - Schnittstellen für PPE-seitige Mailbox-Kommunikation
  - Infrastruktur zur Auswertung der SPE-Ereignisdaten und Erzeugung der Trace-Files
- Keine Abhängigkeiten zum Cell-SDK
- Initialisierung erfolgt vor Start der SPE-Threads, Auswertung nach Ende der SPE-Threads → Vermeidung von PPE-Overhead zur SPE-Laufzeit

Implementierung (2)

Zusätzliches Werkzeug: CellTrace

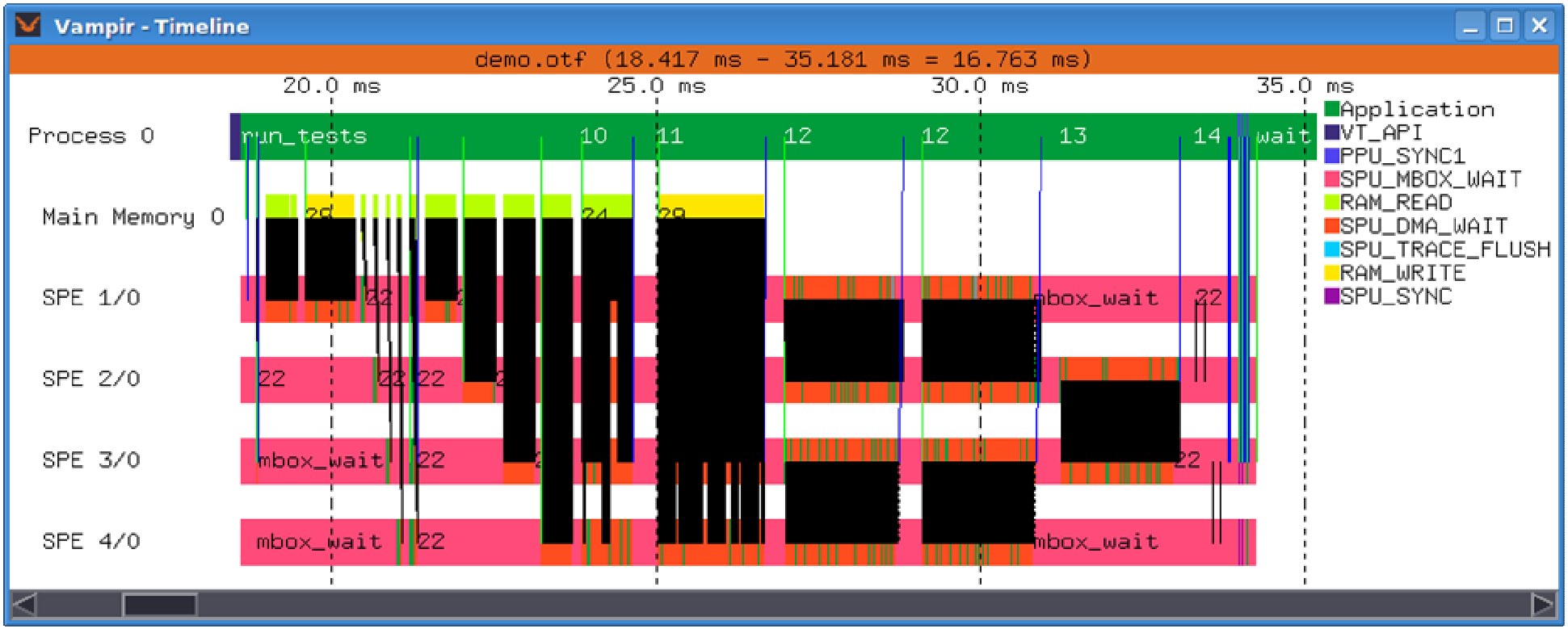
- Header-Dateien für PPE- und SPE-Programme: Instrumentierung von Inline-Funktionen der Header-Dateien aus dem Cell-SDK
- Bibliothek für PPE-Programme
  - Enthält alle Cell-spezifischen Funktionen für das Tracing
  - Modifikation des PPE-Programms: Makro nach Erzeugung der SPE-Threads aufrufen
- Bibliothek für SPE-Programme
  - Erzeugt SPE-Ereignisse im lokalen Speicher
  - Verwaltet Trace-Puffer im lokalen Speicher
  - Modifikation des SPE-Programms: Makro zu Beginn von main() aufrufen
  - Manuelle Instrumentierung von SPE-Funktionen

Implementierung (3)



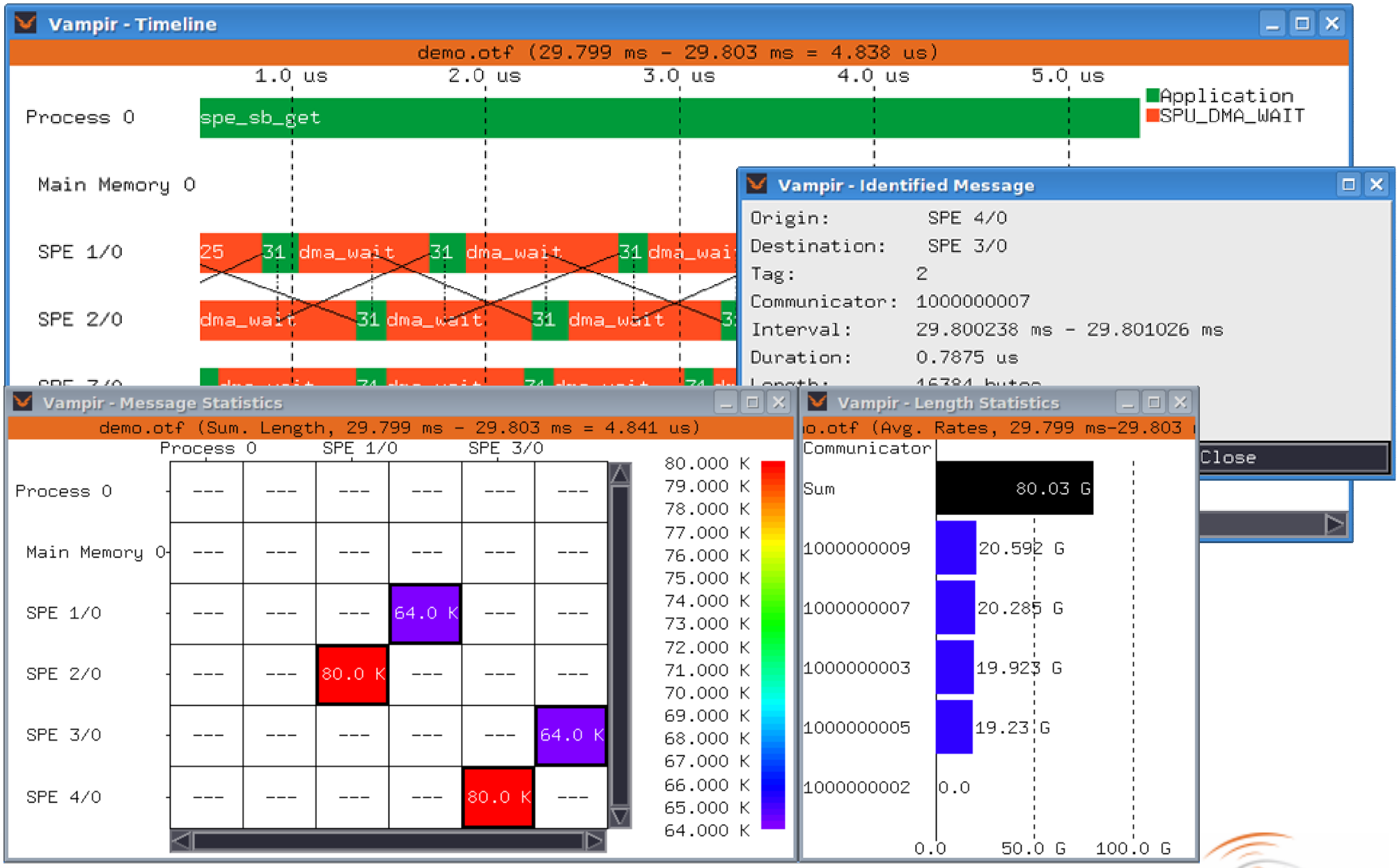
Programm-Übersetzung mit CellTrace und VampirTrace

Trace-Visualisierung mit Vampir (1)



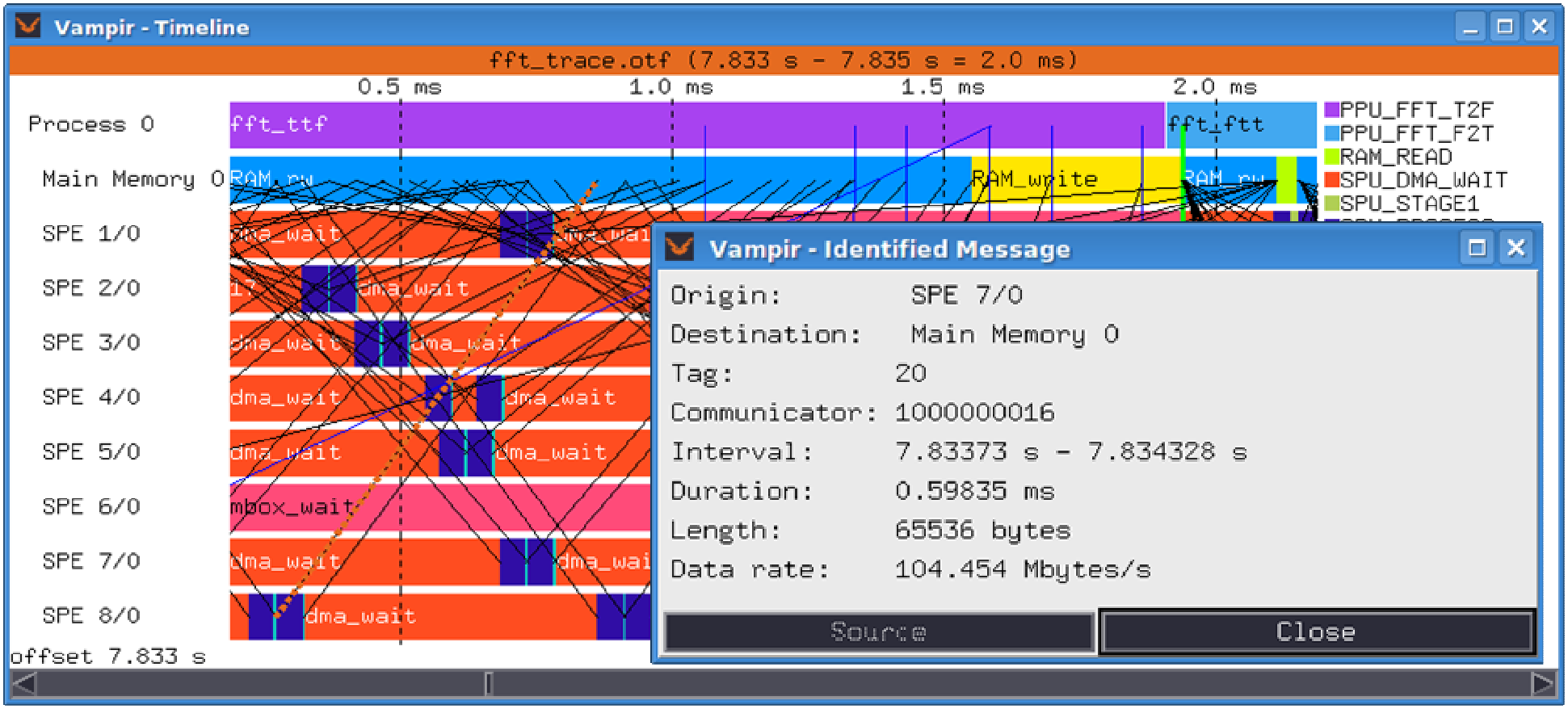
- Erzeugte OTF-Traces können mit Vampir analysiert werden
- Keine Vampir-Modifikation erfolgt, bestehende Funktionen zur Darstellung von MPI-Kommunikation werden genutzt

Trace-Visualisierung mit Vampir (2)



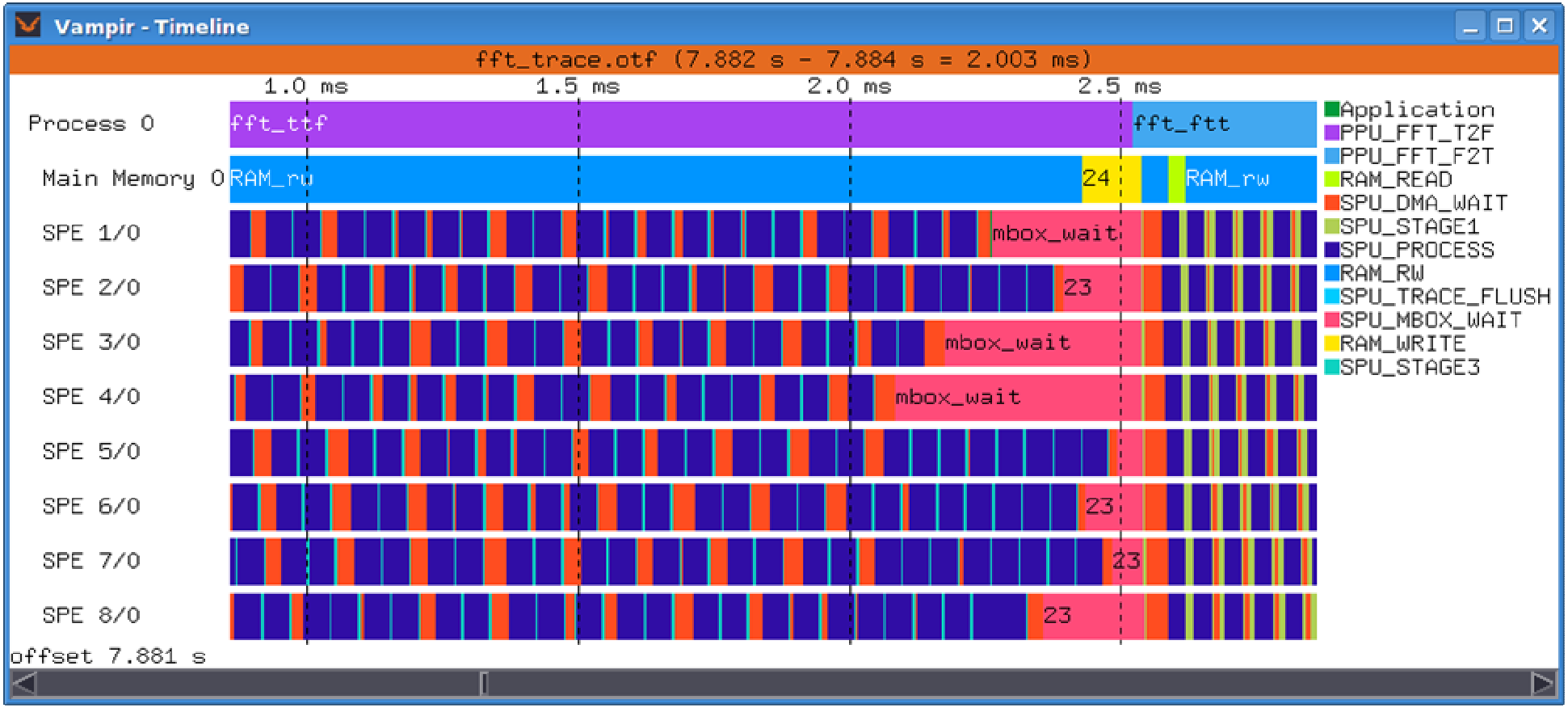
- Einleitung
- Software-Tracing auf Cell-Prozessoren
- Implementierung und Funktionalität
- **Ergebnisse und Overhead-Betrachtung**
- Zusammenfassung und Ausblick

Komplexe Cell-Anwendungen: FFT (1)



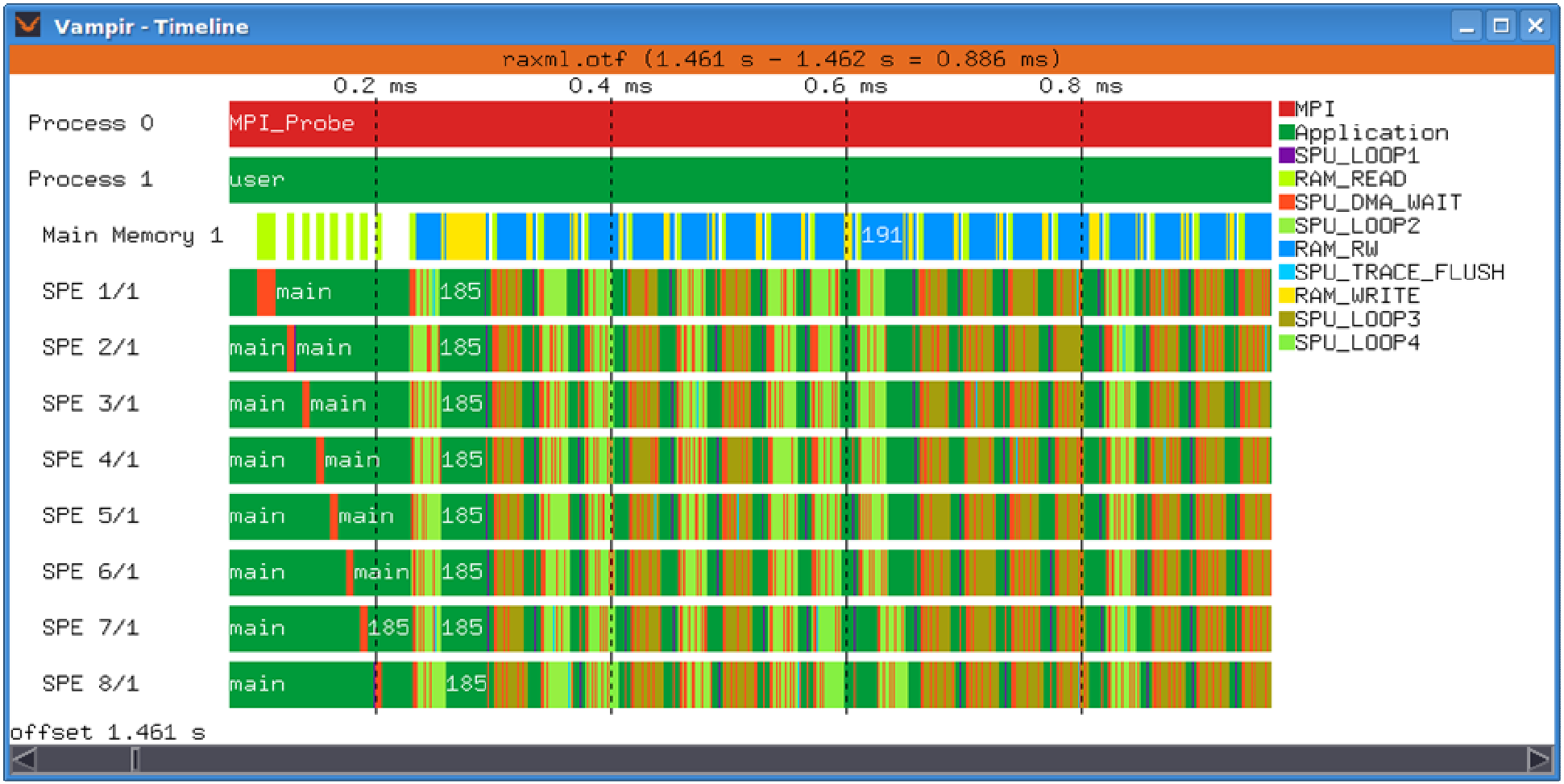
FFT an Synchronisationspunkt
acht SPEs, 64 KByte Seitengröße, 11,9 GFLOPS

Komplexe Cell-Anwendungen: FFT (2)



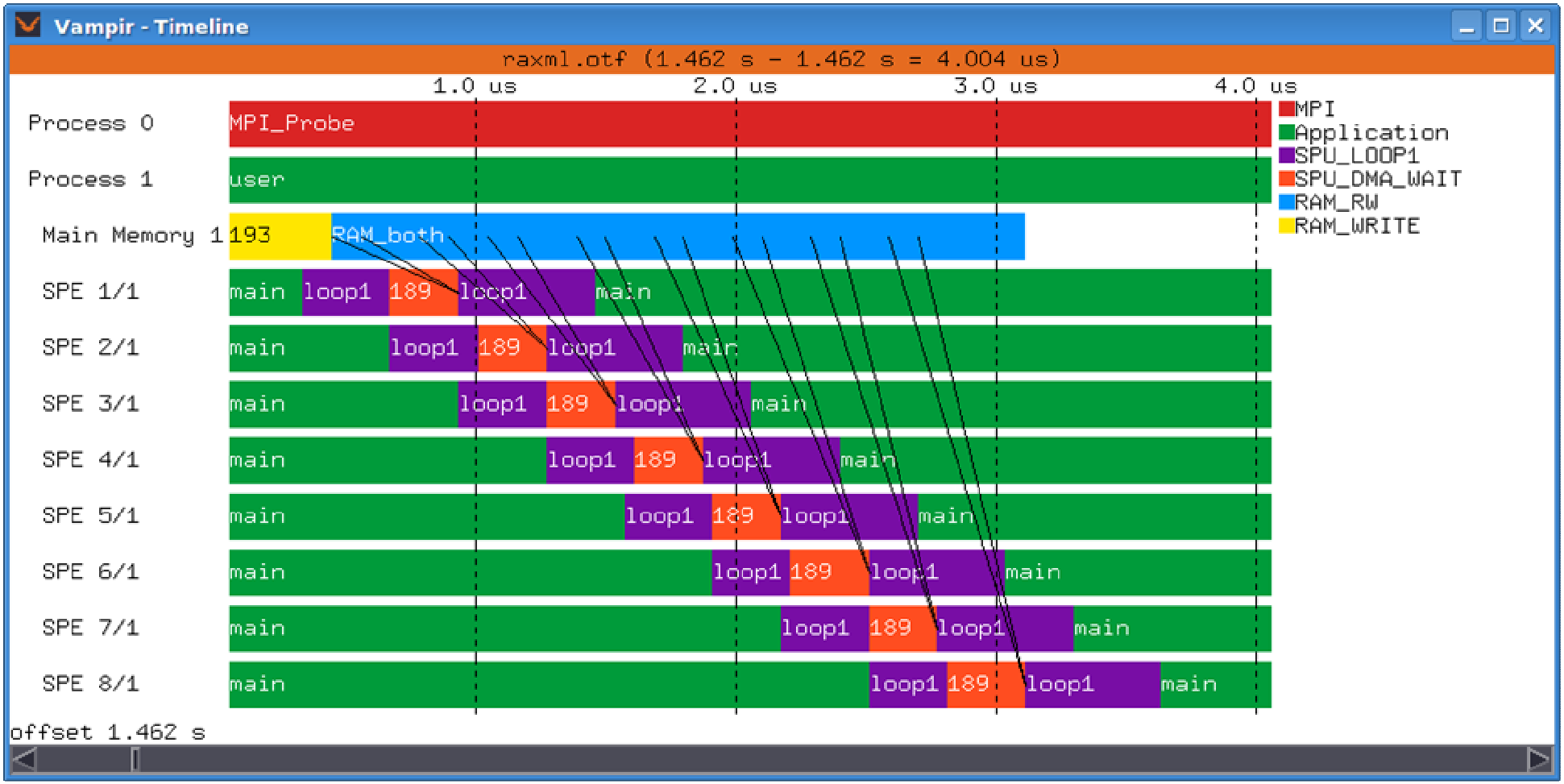
FFT an Synchronisationspunkt
acht SPEs, 16 MByte Seitengröße, 42,9 GFLOPS

Komplexe Cell-Anwendungen: RAxML (1)



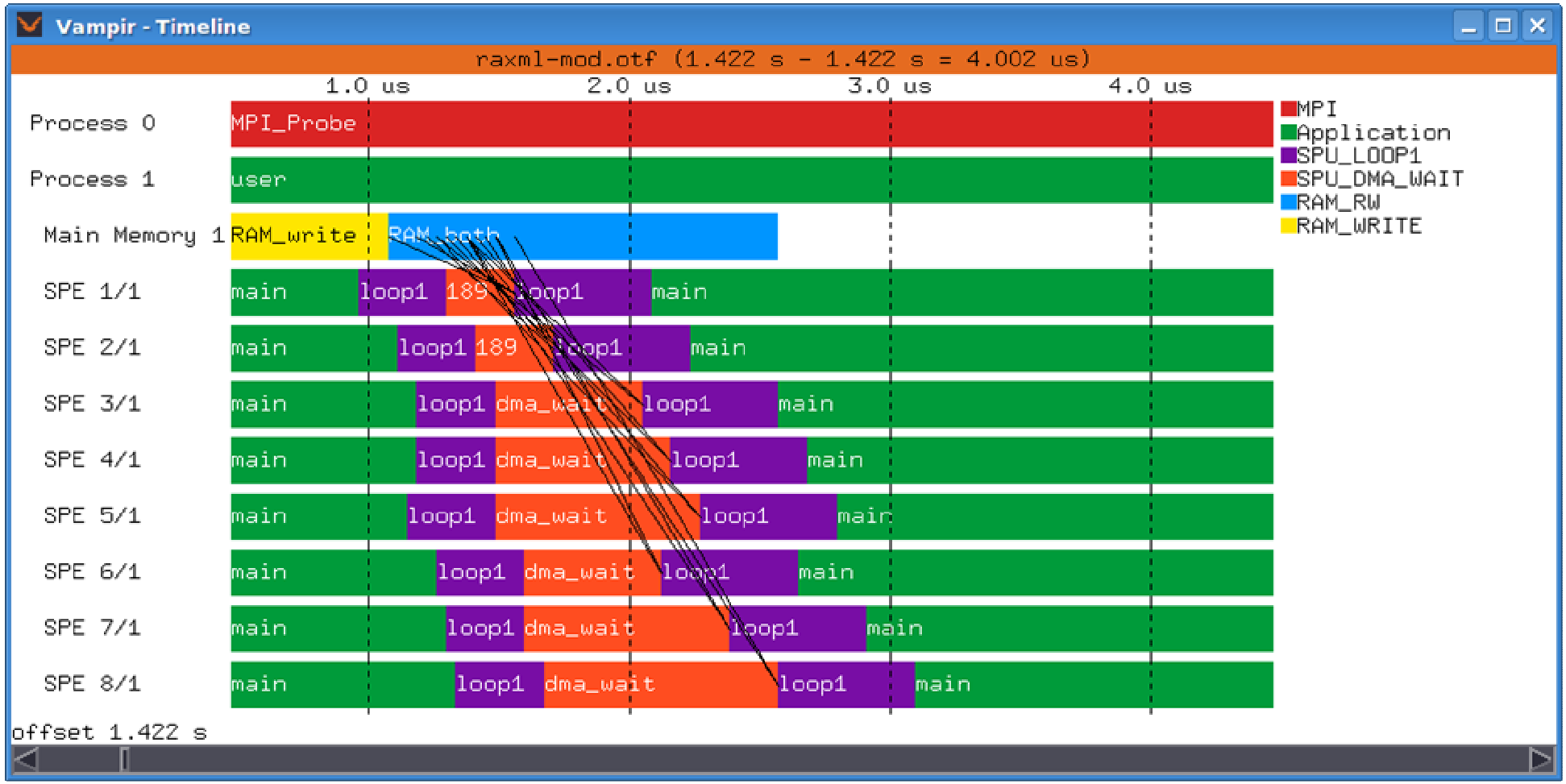
RAxML (Randomized Accelerated Maximum Likelihood)
mit acht SPEs, Startphase

Komplexe Cell-Anwendungen: RAxML (2)



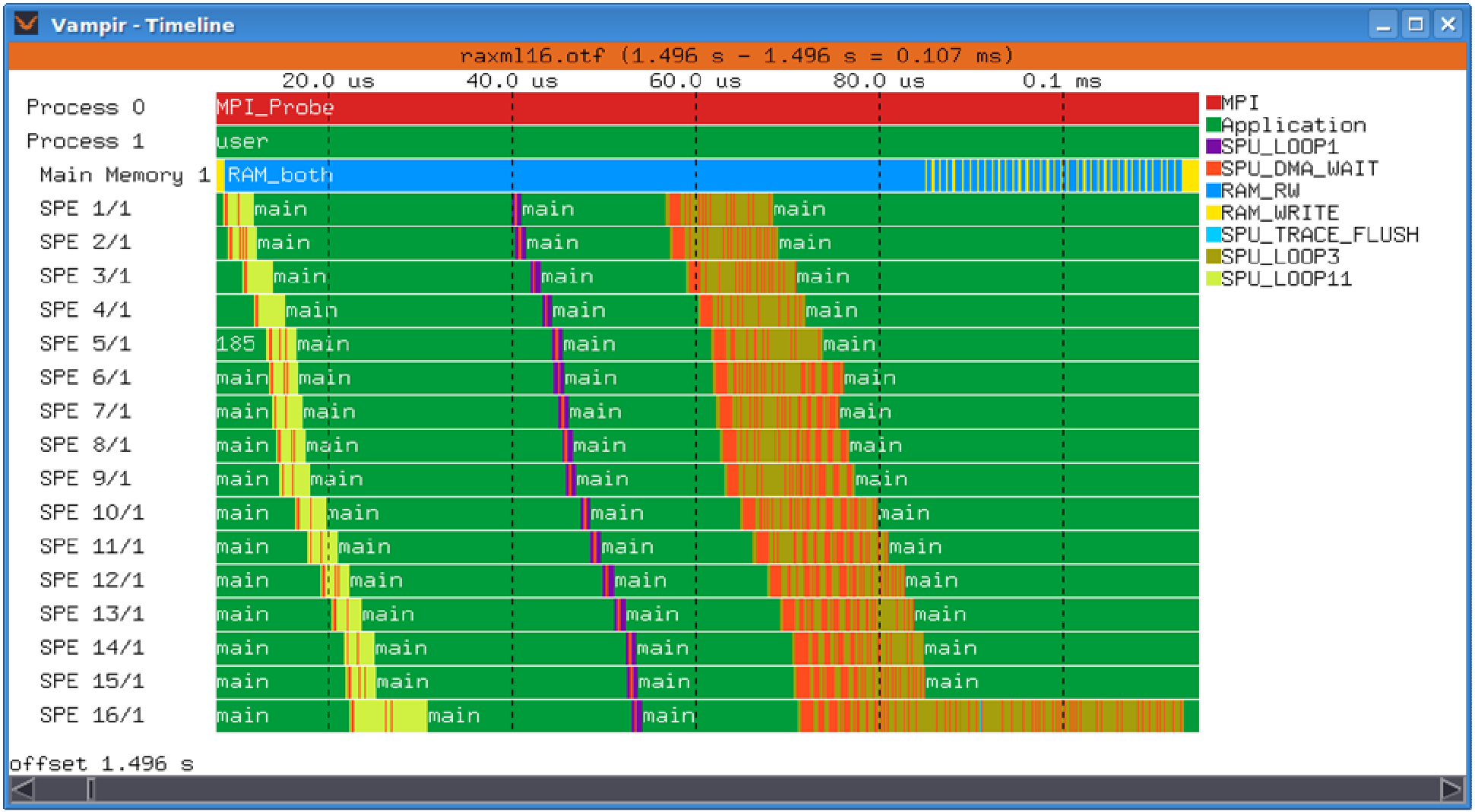
RAxML mit acht SPEs, 4000ns Fenster
Vergrößerung einer kurzen Schleife
versetzter Start, gleichmäßige Laufzeit

Komplexe Cell-Anwendungen: RAxML (3)



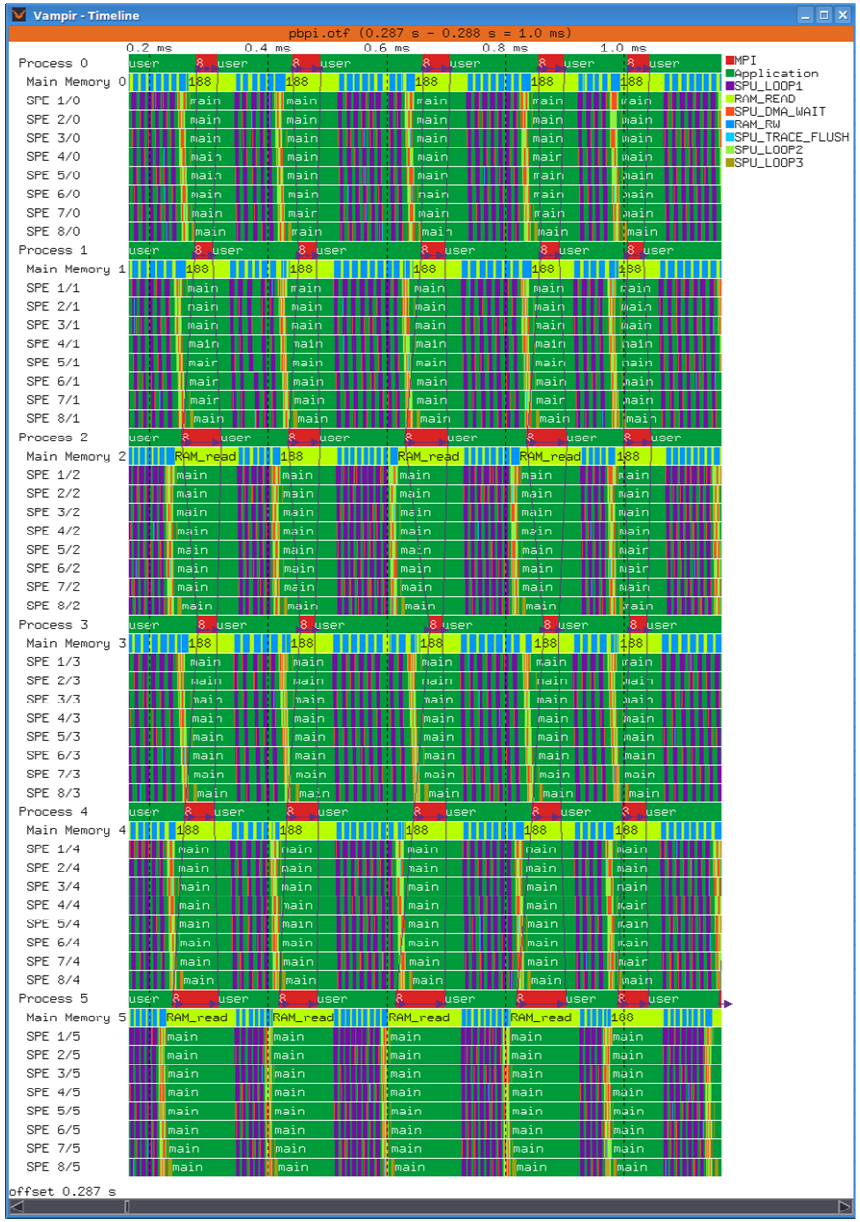
RAxML mit acht SPEs, 4000ns Fenster
Vergrößerung einer kurzen Schleife (modifiziert)
synchroner Start, Konkurrenz um Hauptspeicher

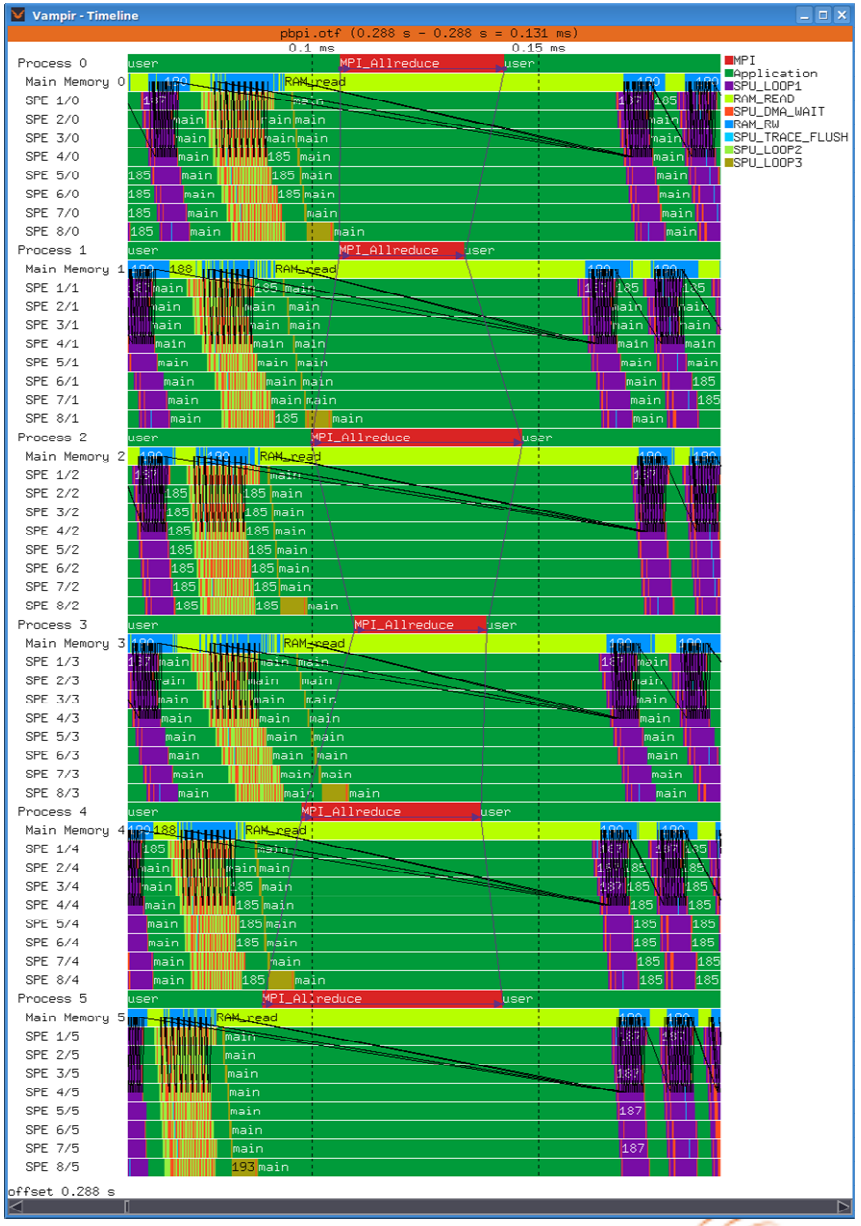
Komplexe Cell-Anwendungen: RAxML (4)



RAxML mit 16 SPEs, Lastverteilungsproblem

Hybride Cell/MPI-Anwendungen: PBPI





PBPI (Parallel Bayesian Phylogenetic Inference)
auf drei QS21-Blades (sechs Cell-Prozessoren)

CellTrace-Overhead

- Overhead-Ursachen
  - Erzeugen der Ereignisse
  - Übertragen der Ereignisdaten aus dem lokalen Speicher der SPEs in den Hauptspeicher
  - Belegung von Speicherplatz im lokalen Speicher (Trace-Puffer und Trace-Bibliothek), unter 12 KByte
- Praktische Overhead-Messungen (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 %

(*) Erhöhter Overhead durch intensive Nutzung von DMA-Listen
Trace-Overhead ohne DMAs: 1,4 %

- Einleitung
- Software-Tracing auf Cell-Prozessoren
- Implementierung und Funktionalität
- Ergebnisse und Overhead-Betrachtung
- **Zusammenfassung und Ausblick**

Zusammenfassung

- Konzept für Software-Tracing auf der Cell-Architektur vorgestellt
- Prototypische Implementierung: CellTrace
  - Ermöglicht Aufzeichnung von Cell-spezifischen Ereignissen
  - Trace-Overhead typischerweise unter fünf Prozent
- Visualisierung der Ereignisdaten mit Vampir möglich
  - Wertvolle Einblicke in Laufzeitverhalten von Cell-Anwendungen
  - Intuitive Performance-Analyse und –Optimierung
  - Begleitende Nutzung bei Software-Entwicklung möglich
- Unterstützung für große, hybride Cell/MPI-Anwendung

Ausblick

- Mögliche Verbesserungen beim Tracing:
  - Vollständige Integration in VampirTrace
  - Effizientere Erzeugung der SPE-Traces möglich, erfordert Unterstützung von einseitiger Kommunikation durch OTF und Vampir
  - Nutzung zusätzlicher Analysemöglichkeiten wie Alignment-Checks

- Mögliche Verbesserungen bei der Visualisierung:
  - Anzeige der Kommunikator-Namen
  - Einfärbung von DMA-Nachrichten nach DMA-Tag, Größe oder Bandbreite
  - Darstellung der Intensität von Hauptspeicher-Zugriffen auf Basis der Bandbreiten

Literatur (Auszug)

- BRUNST, Holger; NAGEL, Wolfgang E.: Scalable Performance Analysis of Parallel Systems: Concepts and Experiences. *ParCo* 2003
- BLAGOJEVIC et. al.: Scheduling Asymmetric Parallelism on a PlayStation3 Cluster. To appear in ISCCG 2008
- BLAGOJEVIC et. al.: Dynamic multigrain parallelization on the Cell Broadband Engine. *ACM SIGPLAN 2007*
- Hermanns, M.-A.; Mohr, B.; Wolf, F.: Event-Based Measurement and Analysis of One-Sided Communication. Euro-Par 2005
- KURZAK, Jakub; BUTTARI, Alfredo; DONGARRA, Jack: Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization. *IEEE Transactions on Parallel and Distributed Systems* 2007
- JURENZ, Matthias: VampirTrace Software and Documentation. ZIH, TU-Dresden. 2007
- KNÜPFER et. al.: Introducing the Open Trace Format (OTF). ICCS 2006
- IBM, SCEI, Toshiba: *Cell Broadband Engine Architecture*. 2005
- IBM, Sony, Toshiba: *Cell Broadband Engine Programming Handbook*. 2007
- EICHENBERGER et. al.: Using advanced compiler technology to exploit the performance of the Cell Broadband Engine architecture. *IBM Systems Journal* 45, 2006
- CHOW, Alex C.; FOSSUM, Gordon C.; BROKENSHIRE, Daniel A.: A Programming Example: Large FFT on the Cell Broadband Engine. 2005
- HACKENBERG, Daniel: Einsatz und Leistungsanalyse der Cell Broadband Engine. ZIH, TU-Dresden. 2007