pallas

Competence in High Performance Computing

Professionelle Entwicklungstools auf PC‘s und HPC Clustern

Mario Deilmann <mario.deilmann@pallas.com>

DECUS Symposium, Bonn
8. April 2003

Pallas GmbH
Hermülheimer Straße 10
D-50321 Brühl, Germany

info@pallas.com
www.pallas.com

Informationen zur Person

- Name: Mario Deilmann
- Firma: Pallas
- Aufgabe: Technischen Support & Produktplanung
- Ausbildung: Diplom-Ingenieur (Maschinenbau)
  Dissertation (Direkte Numerische Simulation)

Agenda

- Sinn und Zweck von professionellen Entwicklungstools
- Kurze Übersicht über die verschiedenen Ansätze zur Optimierung
- Serielle Optimierung
  - Serielle Performance Analyse
- Parallele Programmentwicklung und Optimierung
  - Paralleler Debugger
  - MPI Performance Analyse
  - OpenMP Performance Analyse
- Hybride Tools

Think of the tremendous human tragedy of time and energy a lot of scientists have lost trying to use (massive) parallel machines.

Henry Burkhardt
(deposed CEO of Kendall Square Research **)**

Welchen Nutzen haben prof. Entwicklungstools

- Entwicklungszyklus wird effizienter
- Systemressourcen werden optimal genutzt
- Programmlaufzeit wird kürzer d.h. Performance wird größer
- Es werden mehr Probleme in der gleichen Zeit gelöst bei geringeren Ressourcen
- Markteinführung wird verkürzt
- Welche Verbesserung sind notwendig und wie groß der tatsächliche Gewinn der Änderungen an der Applikation (*speed up*)

Man spart Zeit und Geld !

Wer profitiert von Performance Gewinnen

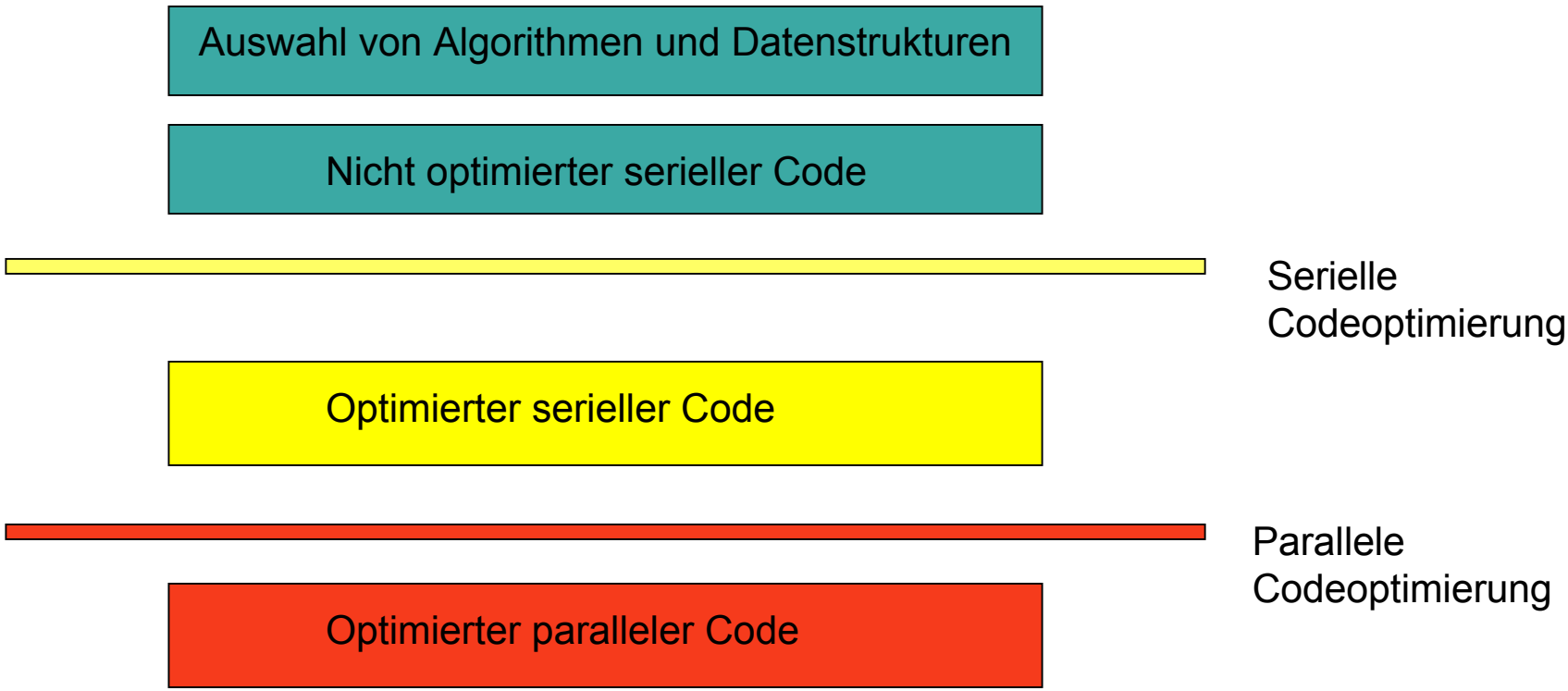
- Öl- und Gasindustrie: Seismische Analyse
- Forschung: Gewöhnliche, partielle Differentialgleichungen
- IT: *Data Mining*
- Pharmazeutischen Industrie: Wirkstoffdesign
- Meteorologie: Wettervorhersage
- Automobileindustrie: Crash Simulation
- Chemieindustrie: Simulation molekulardynamischer Prozesse
- Luftfahrtindustrie: Simulation im Windkanal
- Finanzbereich: Risikoanalysen

Lange Programm-Laufzeit und große Datenmengen

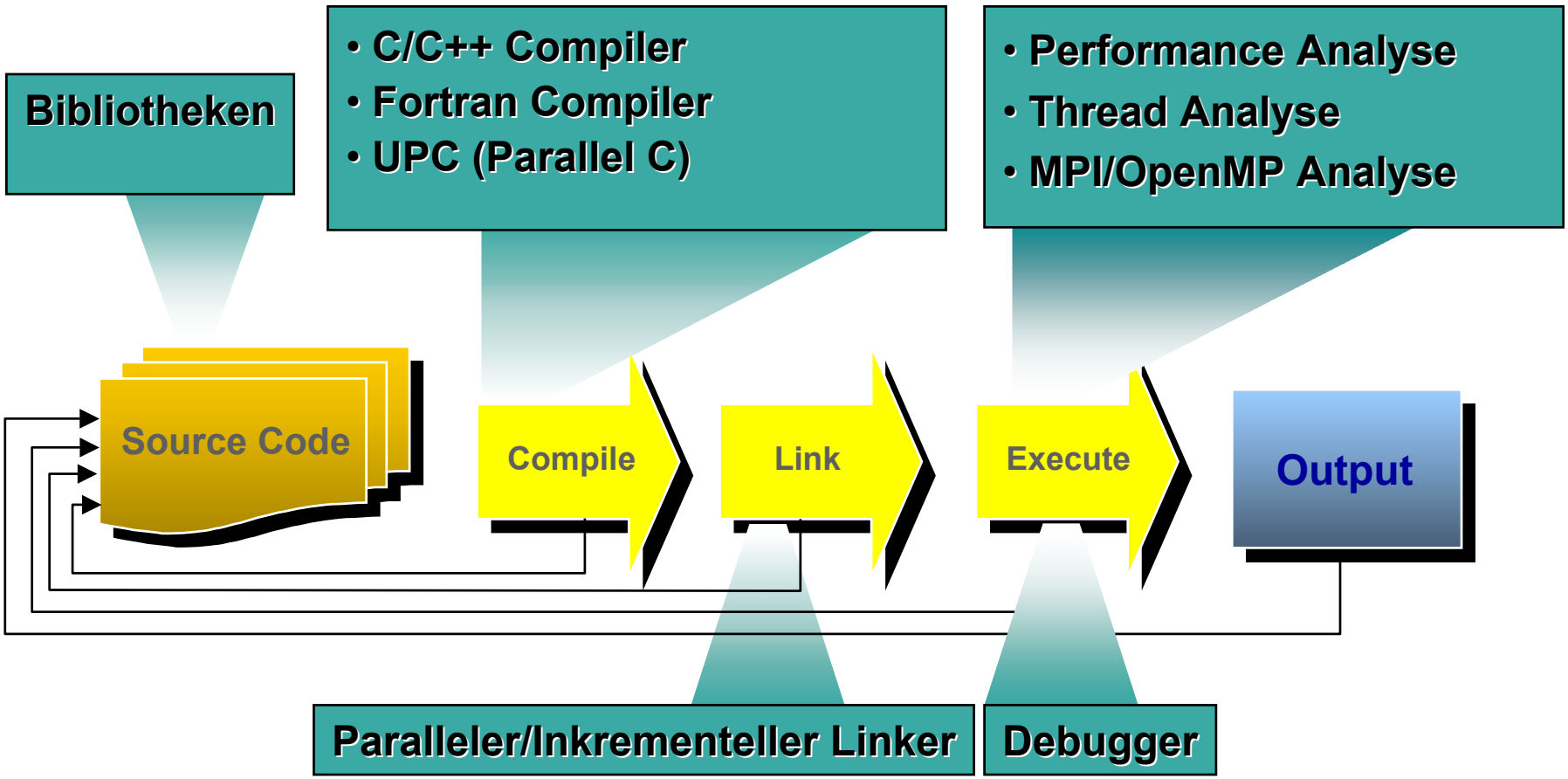
Welche Tools gibt es

- Optimierende Compiler beschleunigen die Ausführungszeit
- (Parallele) Linker verkürzen den *Build*-Zyklus
- Debugger verkürzen entscheidend die Fehlersuche
- Profiler können ermitteln welche Zeit in welchem Programmteil verbracht wurde
- Spezielle Bibliotheken ermögliche den Zugriff auf hoch optimierte plattformunabhängige Routinen (BLAS, LAPACK, FFT, JPEG, MPEG, ...)
- Performance Analysetools liefern Informationen über Lastverteilung und Performance Engpässe

Applikation Entwicklungsphase

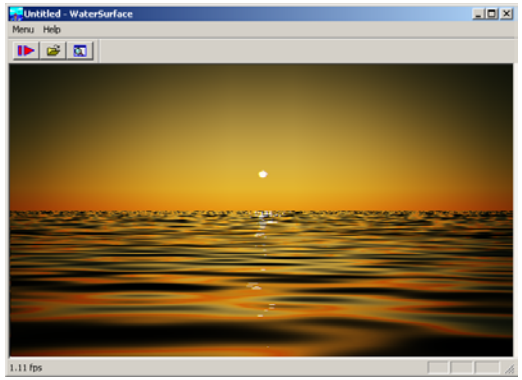


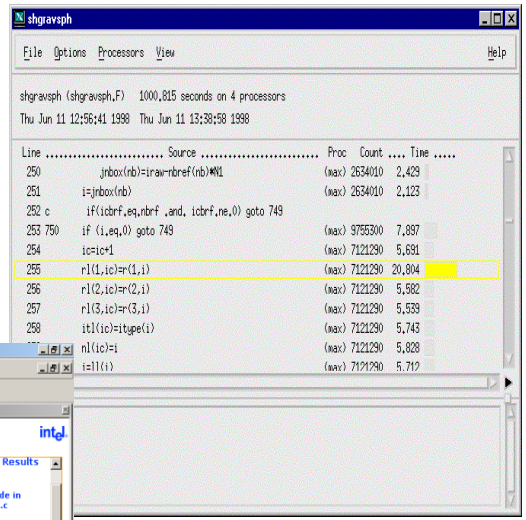
Programm Zyklus / Einsatz von Werkzeugen

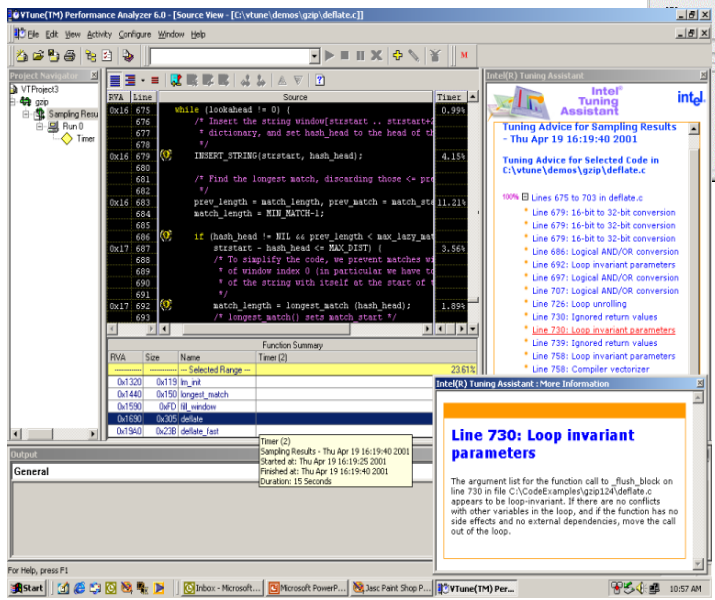


pallas

Serielle Code-Optimierung

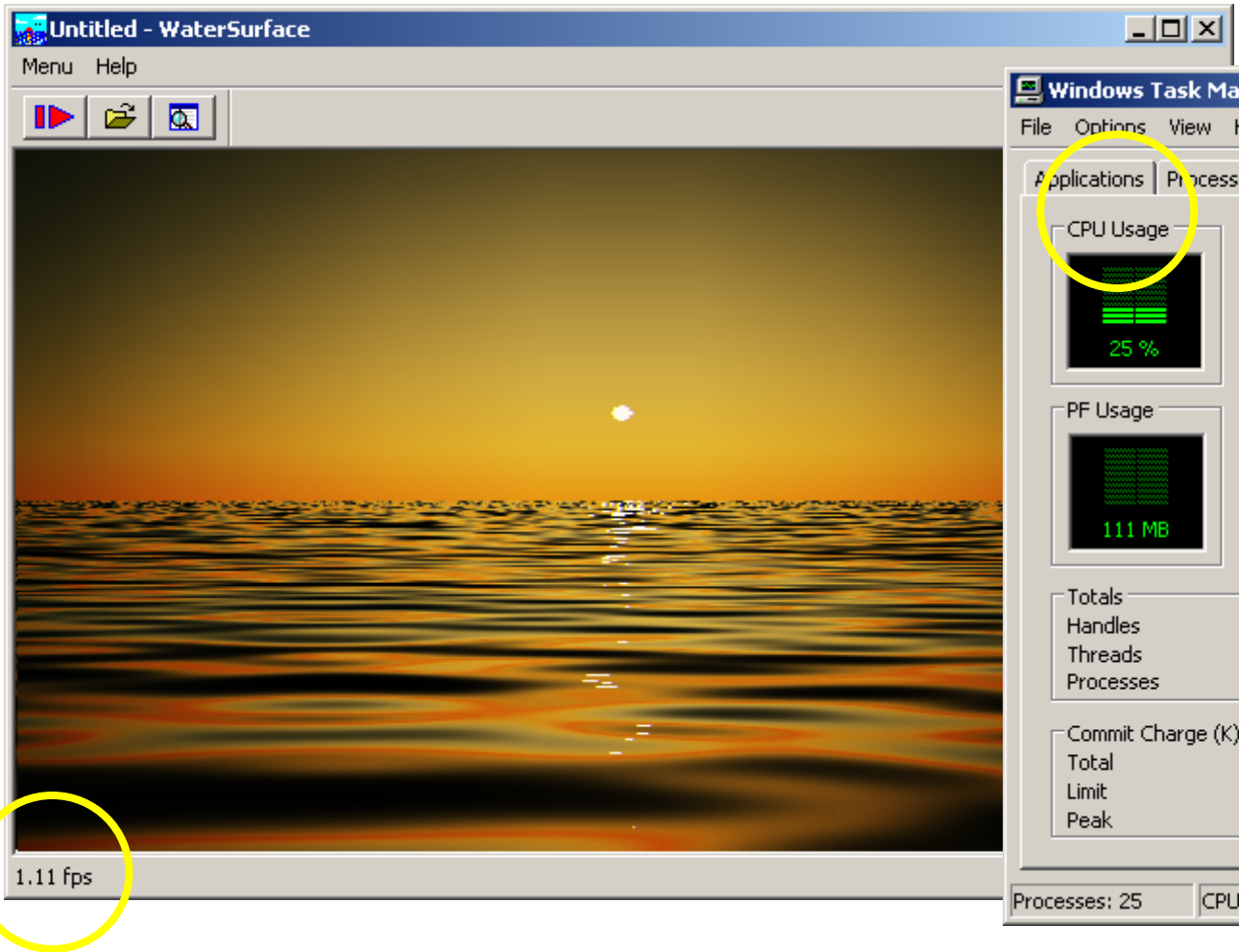


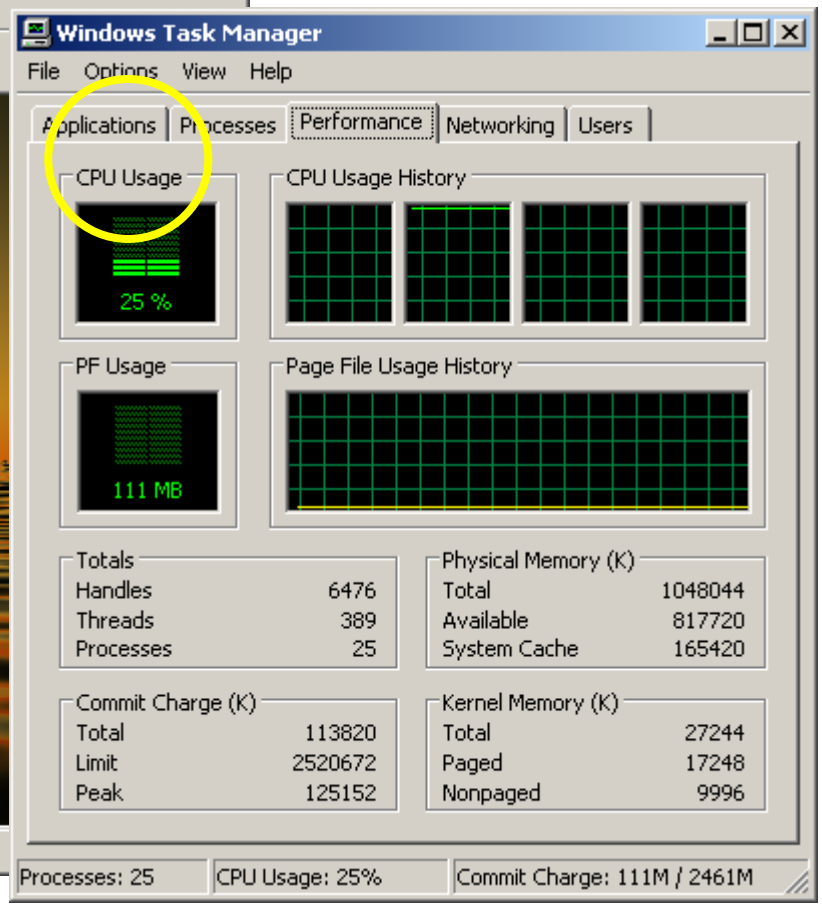




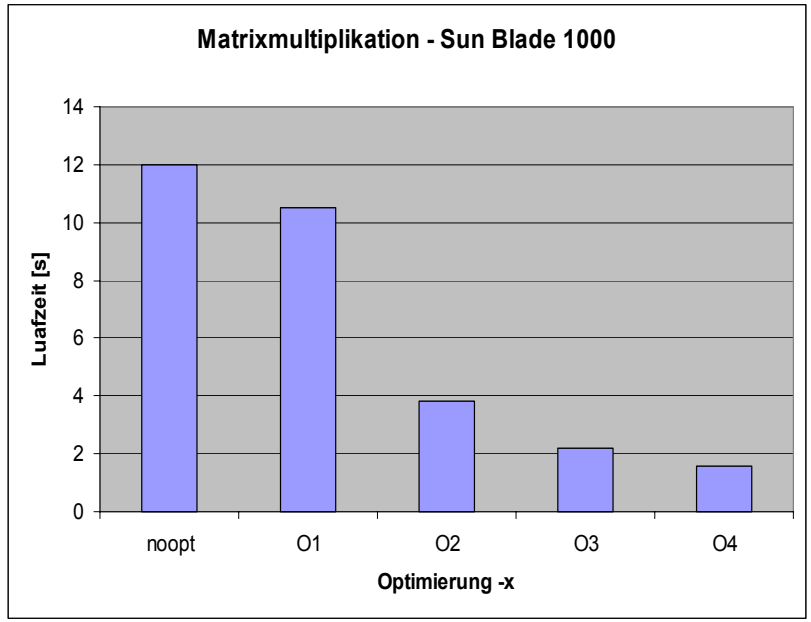
Basis Performance- nicht optimierter Code

- Simulation einer bewegten Wasseroberfläche



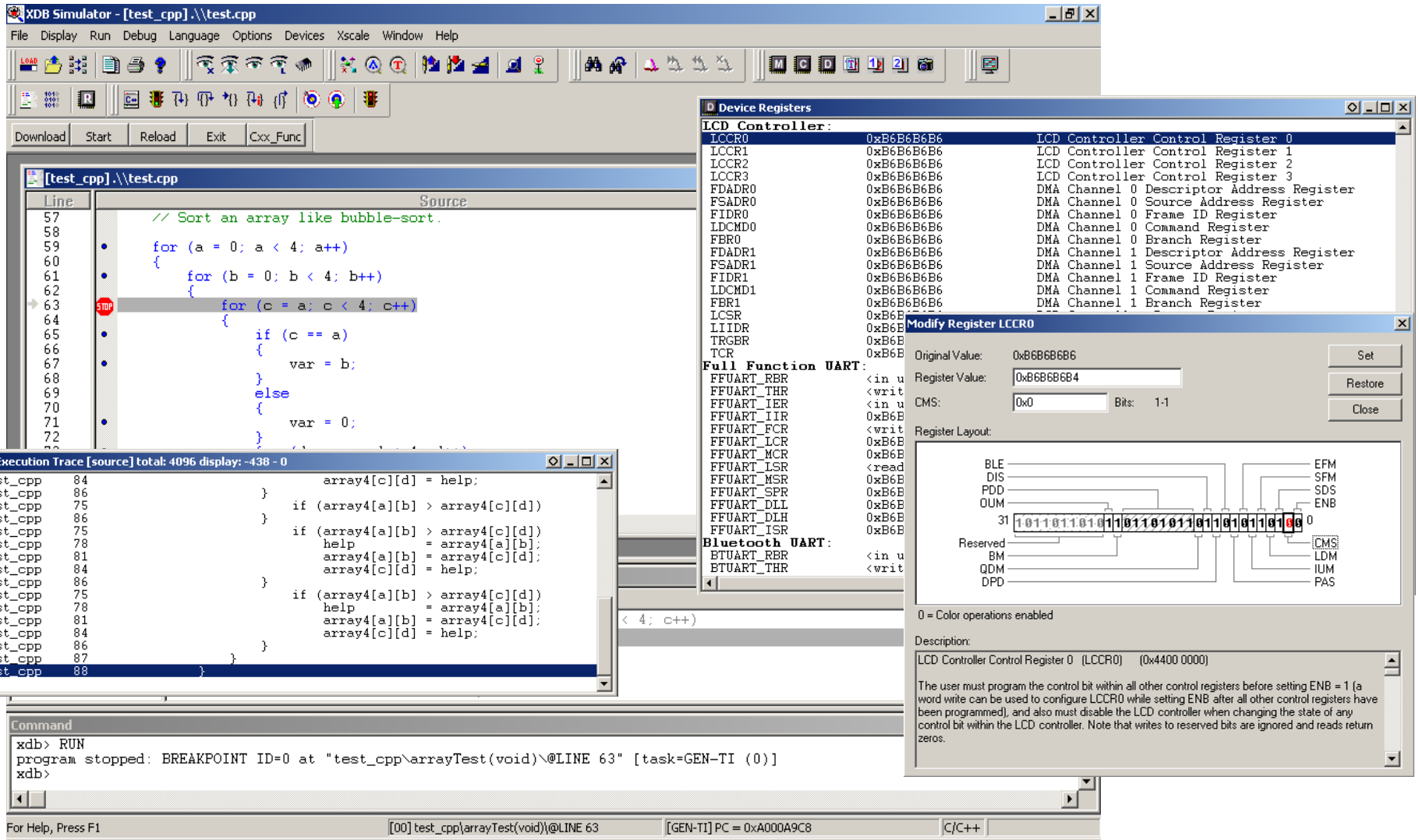


Compiler Optimierungen

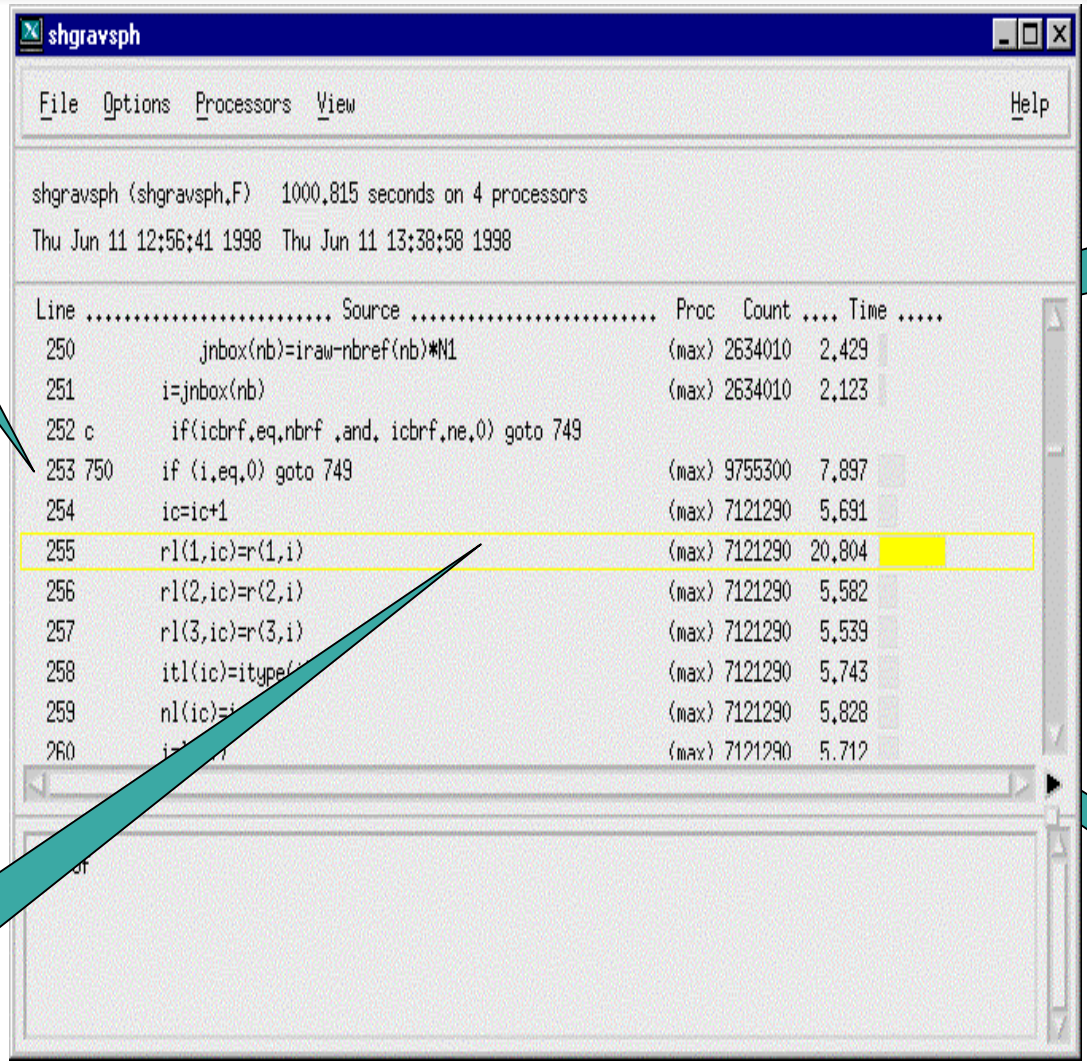


- Vektorisierung
- Code-Transformation (Schleifen)
- Spekulations-Analyse
  - Daten / Programm Code in den Cache laden
- Explizite Parallelisierung
- Inter Prozedur Analyse
  - Umstrukturierung des Programmcodes auf Basis verschiedenerer Programm-Module
- Profile Feedback Analyse
  - Umstrukturierung des Programmcodes auf Basis repräsentativer Programmläufe

Serieller Debugger



Profiler



Routine

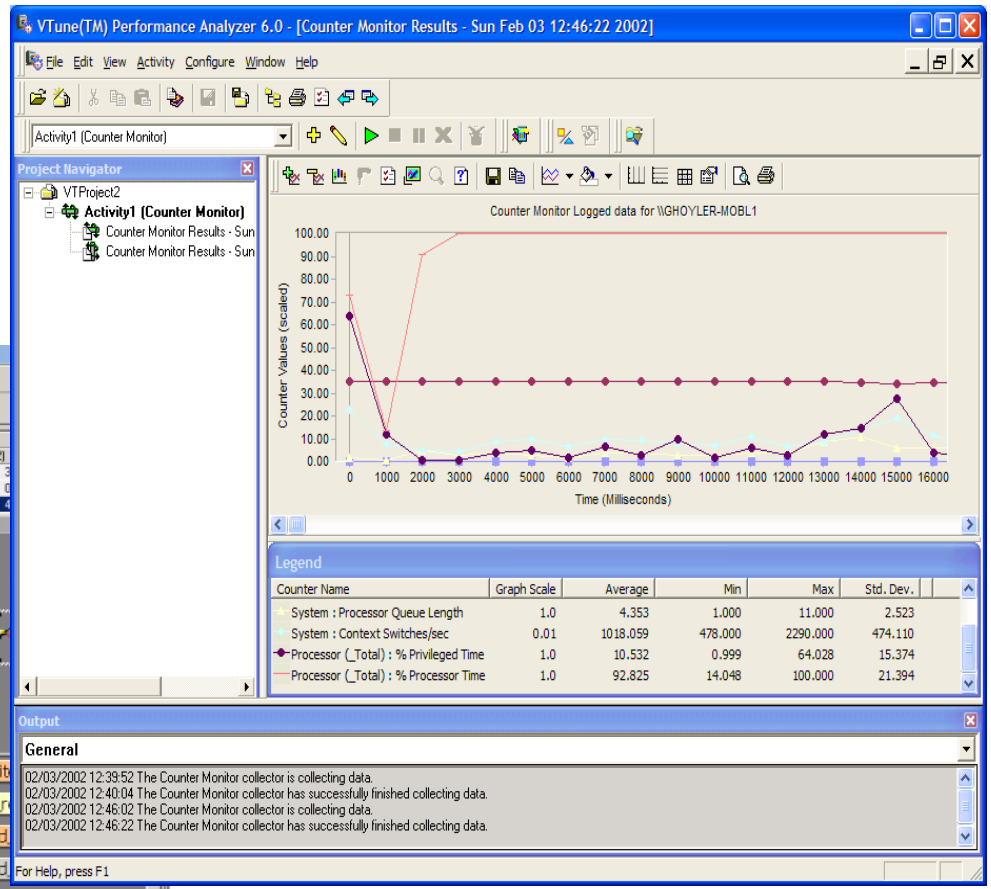
Anzahl der Aufrufe

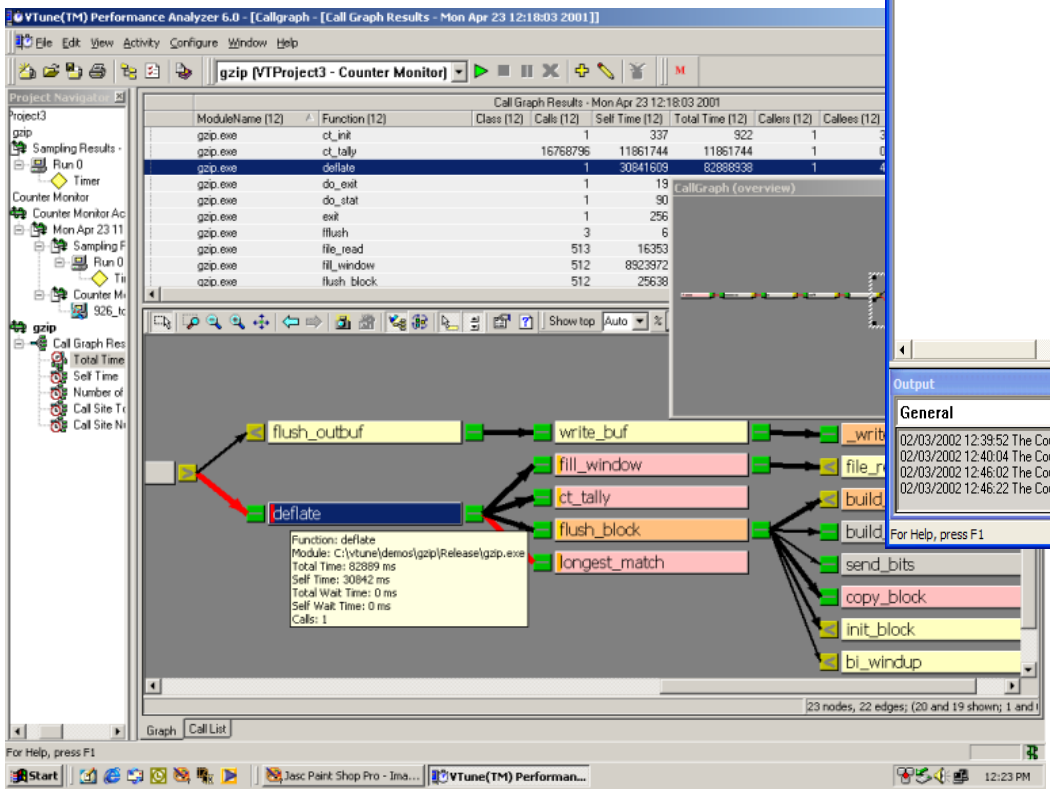
Quellcode-informationen

Laufzeit

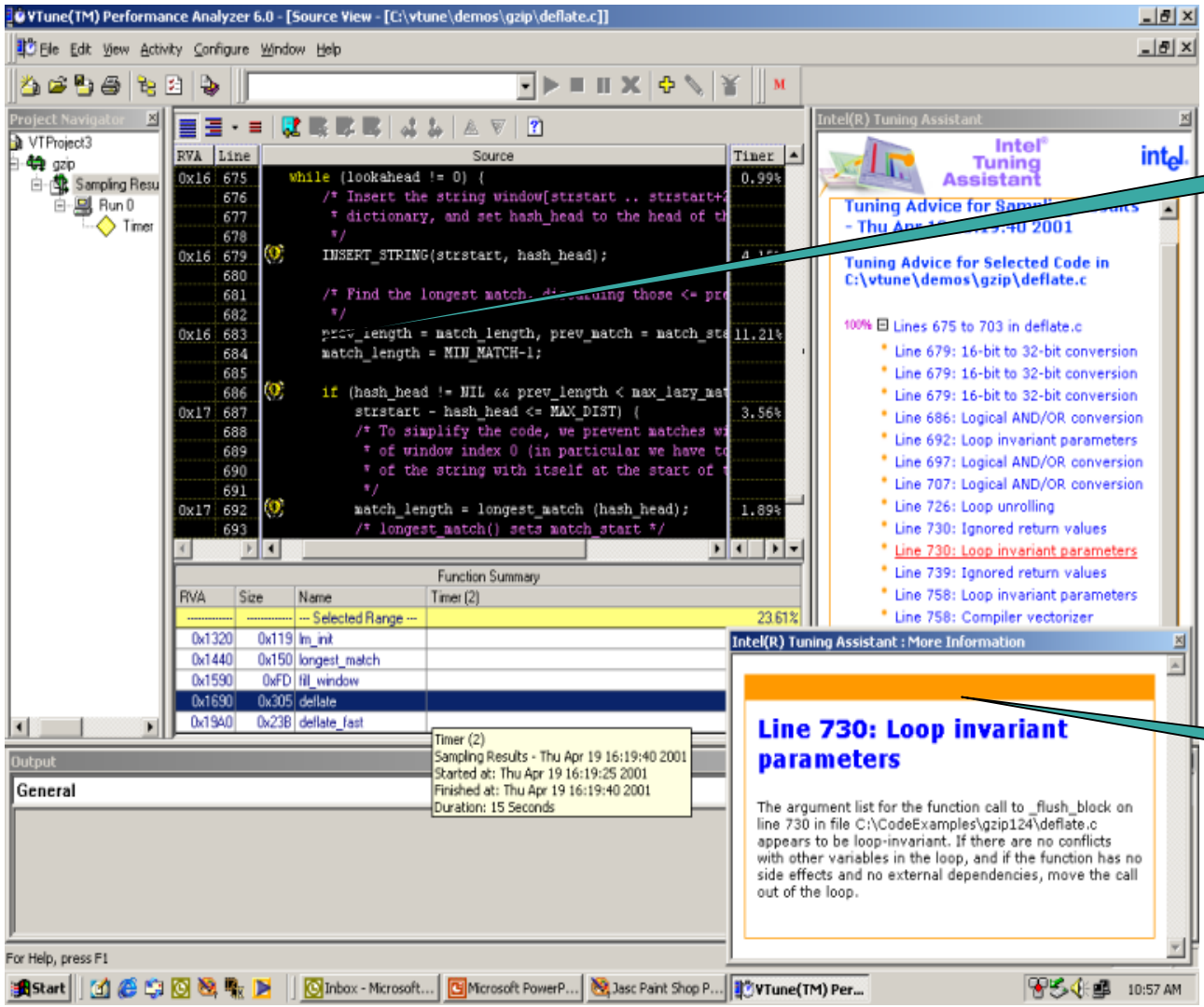
Profiler

- Statistisches Sampling
  - Prozessor Counter
  - DLL‘s, Threads





Profiler - Tuning Assistent

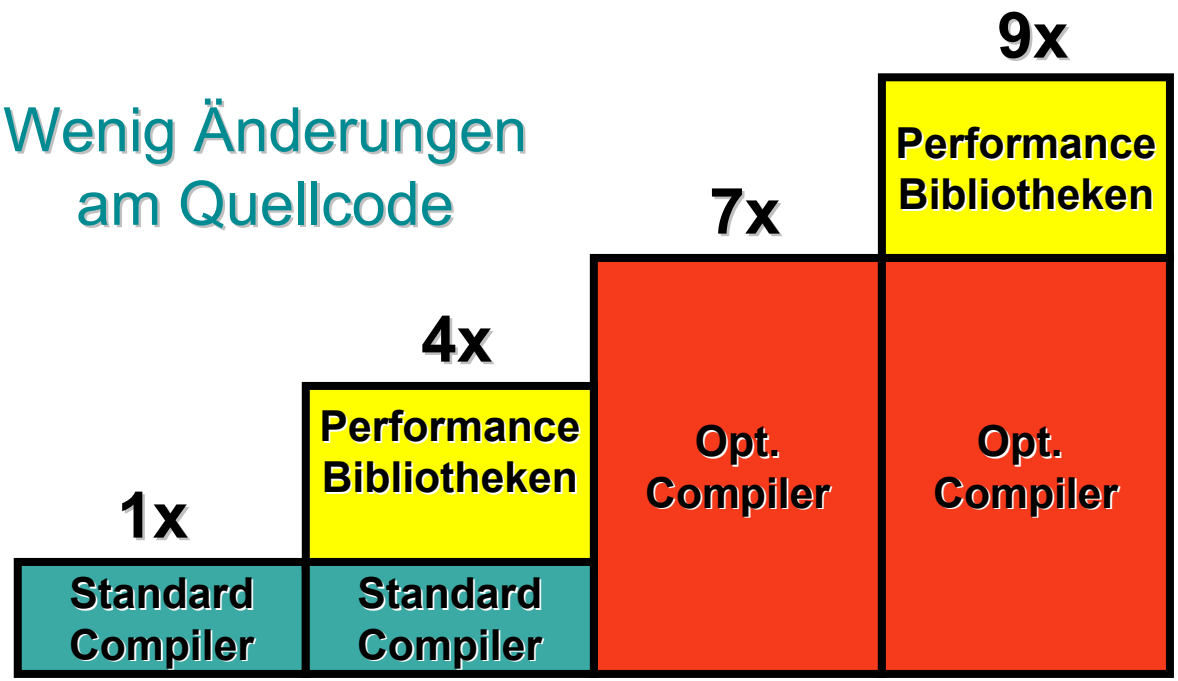


Quellcode

Hinweis

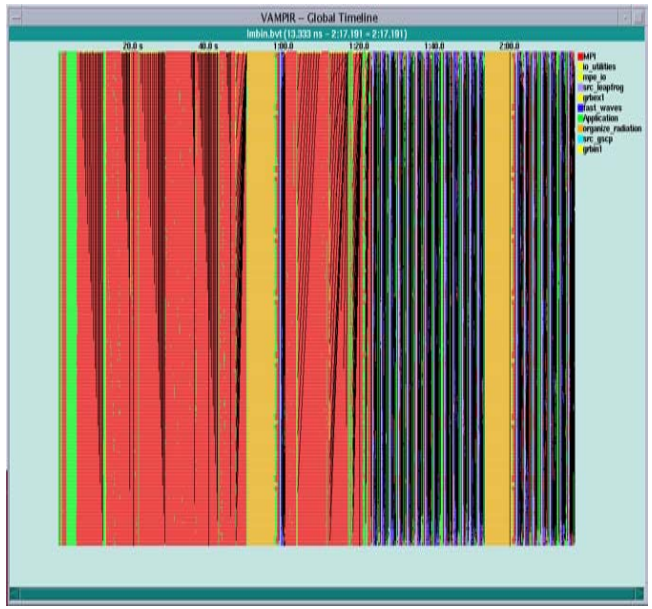
Optimierungszyklus - optimierter serieller Code

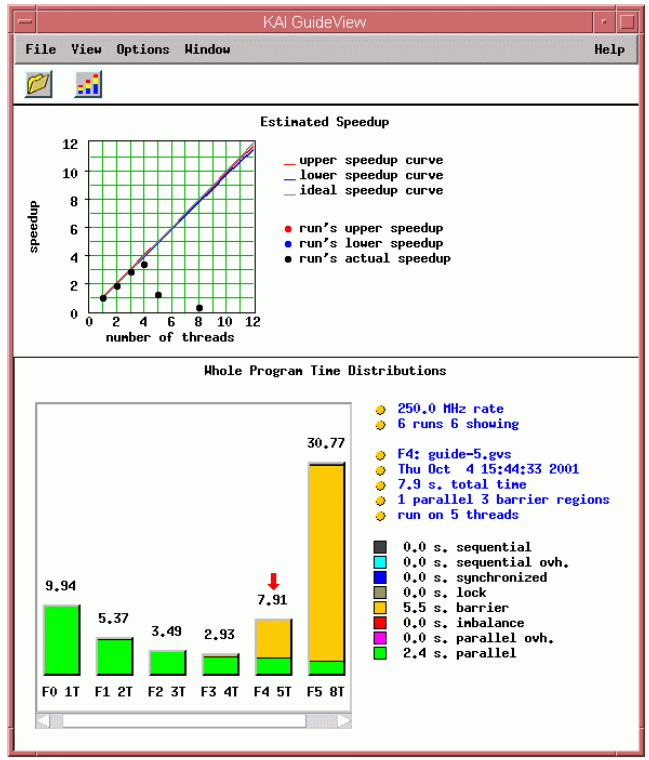
9x schneller



pallas

Parallele Code-Optimierung



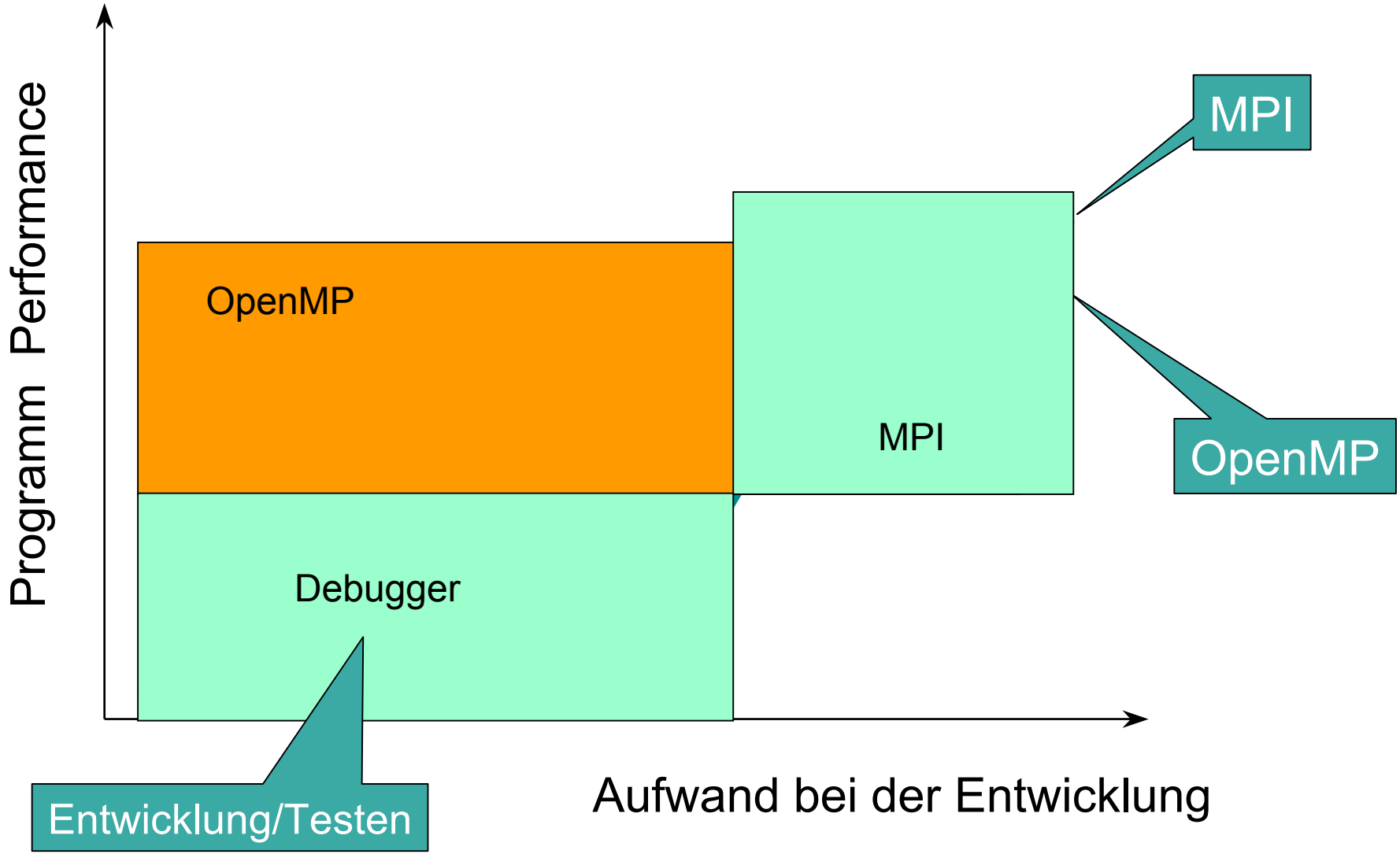


Parallele Programmiermodelle

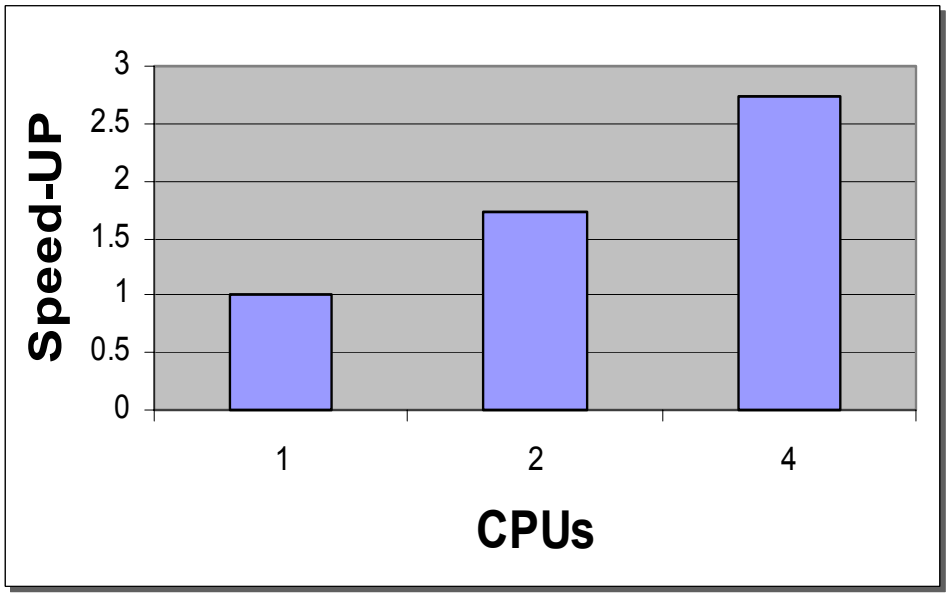
- Multithread
- O
- M
- Hy

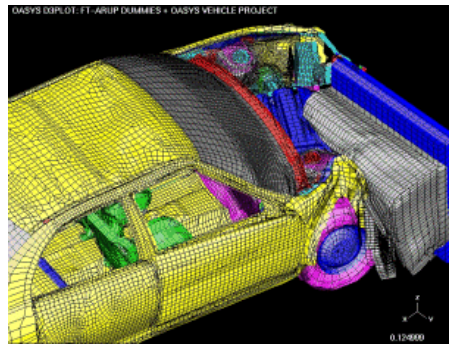
```
!$omp parallel
      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
      print *, "Process ", myid, " of ", numprocs, " is alive"
c       send b to each other process
        do 25 i = 1,bcols
        call MPI_BCAST(b(1,i), brows, MPI_DOUBLE_PRECISION, master,
     &           MPI_COMM_WORLD, ierr)
 25     continue

c       send a row of a to each other process; tag with row number
        do 40 i = 1,numprocs-1
          do 30 j = 1,acols
            buffer(j) = a(i,j)
 30         continue
          call MPI_SEND(buffer, acols, MPI_DOUBLE_PRECISION, i,
     &           i, MPI_COMM_WORLD, ierr)
          numsent = numsent+1
```

Gegenüberstellung: Performance - Aufwand



Fall Studie - LS-DYNA (OpenMP)





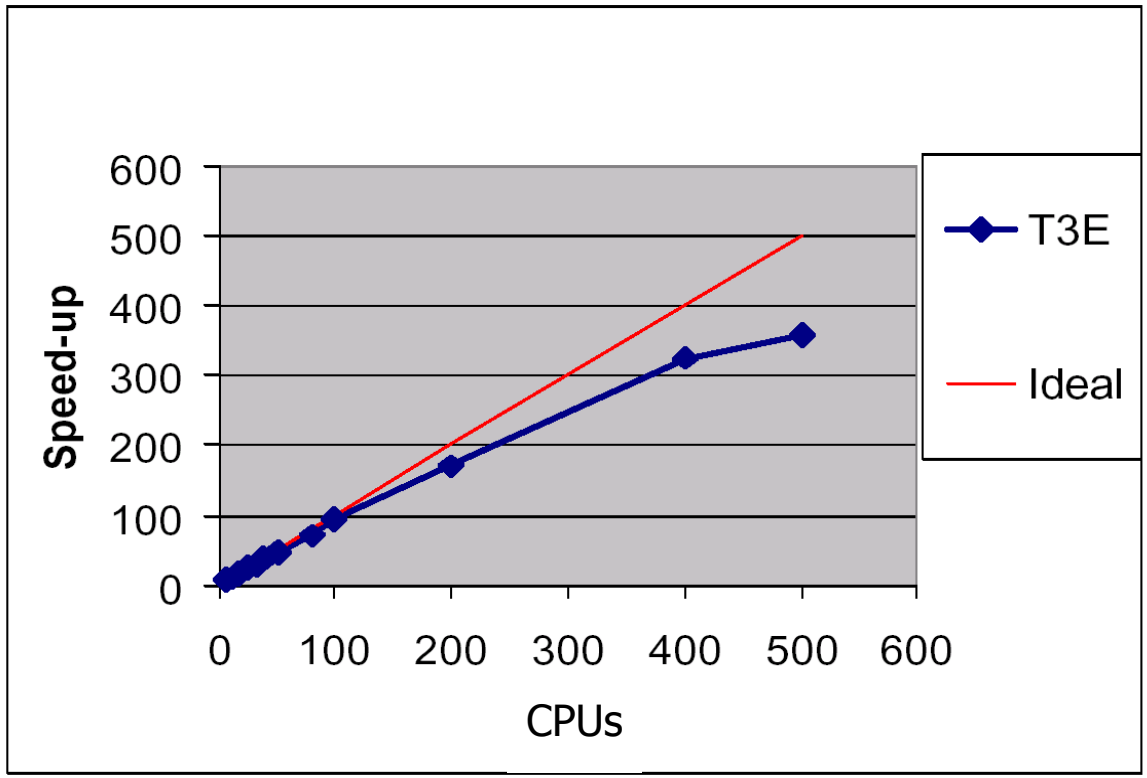
3D: 111.029 Elemente
1 CPU Zeit = 119955 Sekunden
2 CPU Zeit = 69113 Sekunden
4 CPU Zeit = 43864 Sekunden
Quad Xeon

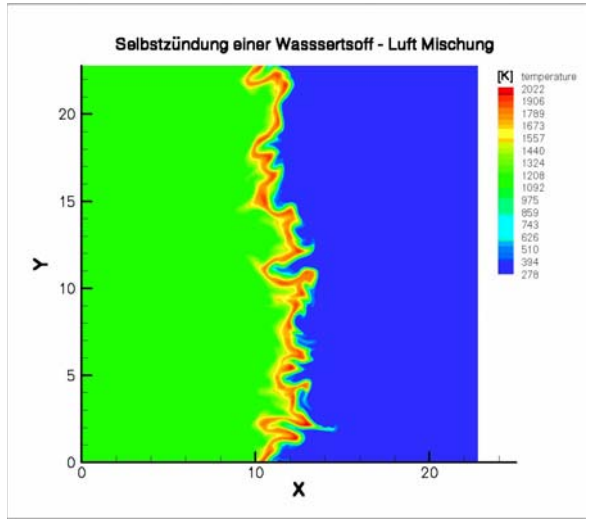
LS-DYNA ist ein Finite Element Programm für Crash-Simulationen

Ca. 750,000 Zeilen Fortran und C Quellcode

1.74x *speed up* auf 2 CPUs, 2.74x *speed up* auf 4 CPUs

Fall Studie – Verbrennungssimulation (MPI)





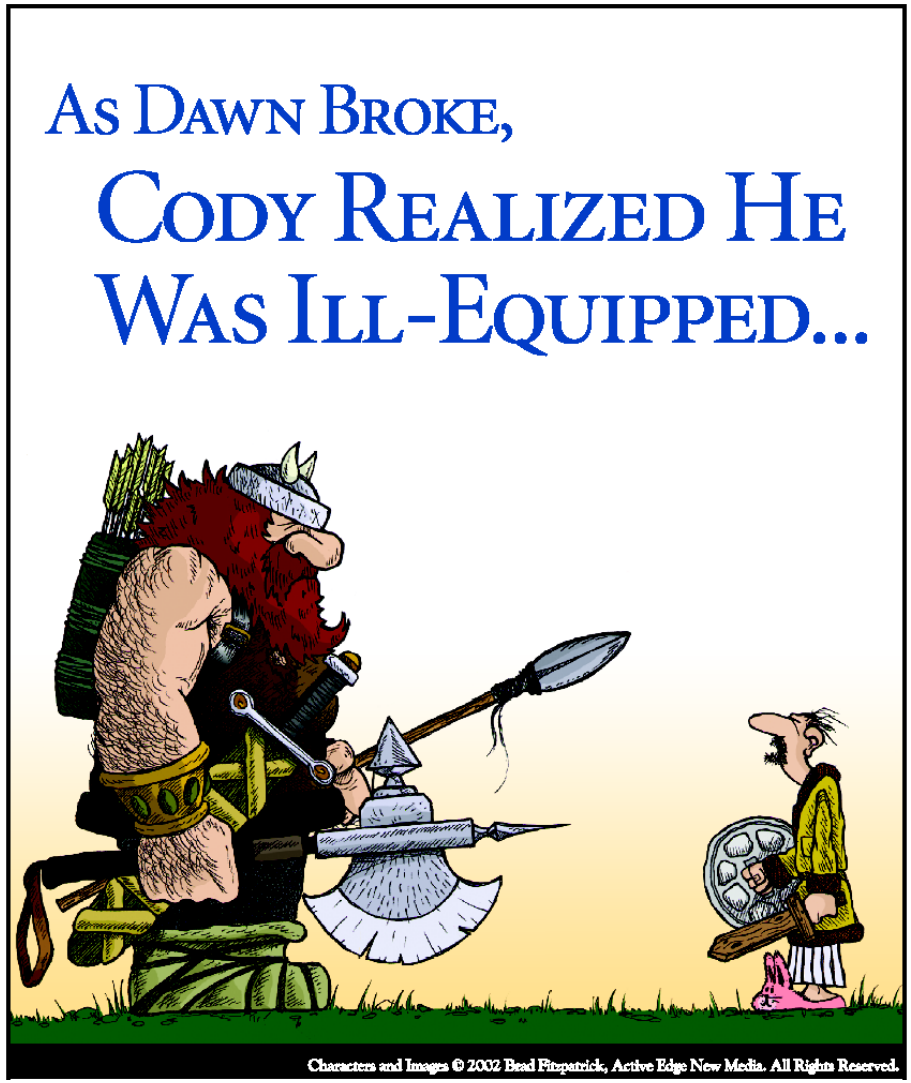
1 Mio. Elemente
2D: 1000 x 1000
Cray T3E
1CPU – 100 Stunden
100CPUs – 1,5 Stunden

NSCORE – Programm zur Direkten numerischen Simulation von Verbrennungsvorgängen

350x *speed up* auf 512 CPUs

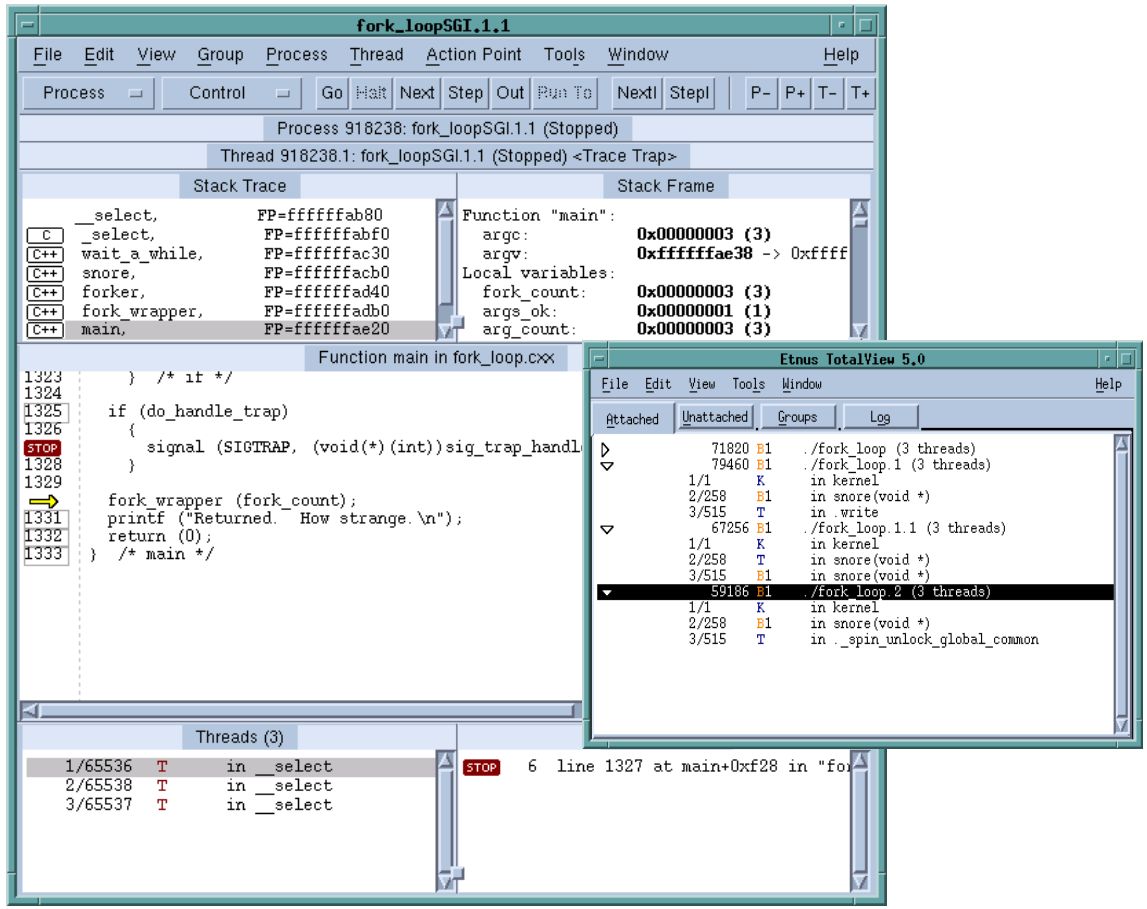
pallas

Parallele Programmentwicklung



pallas

Paralleler Debugger



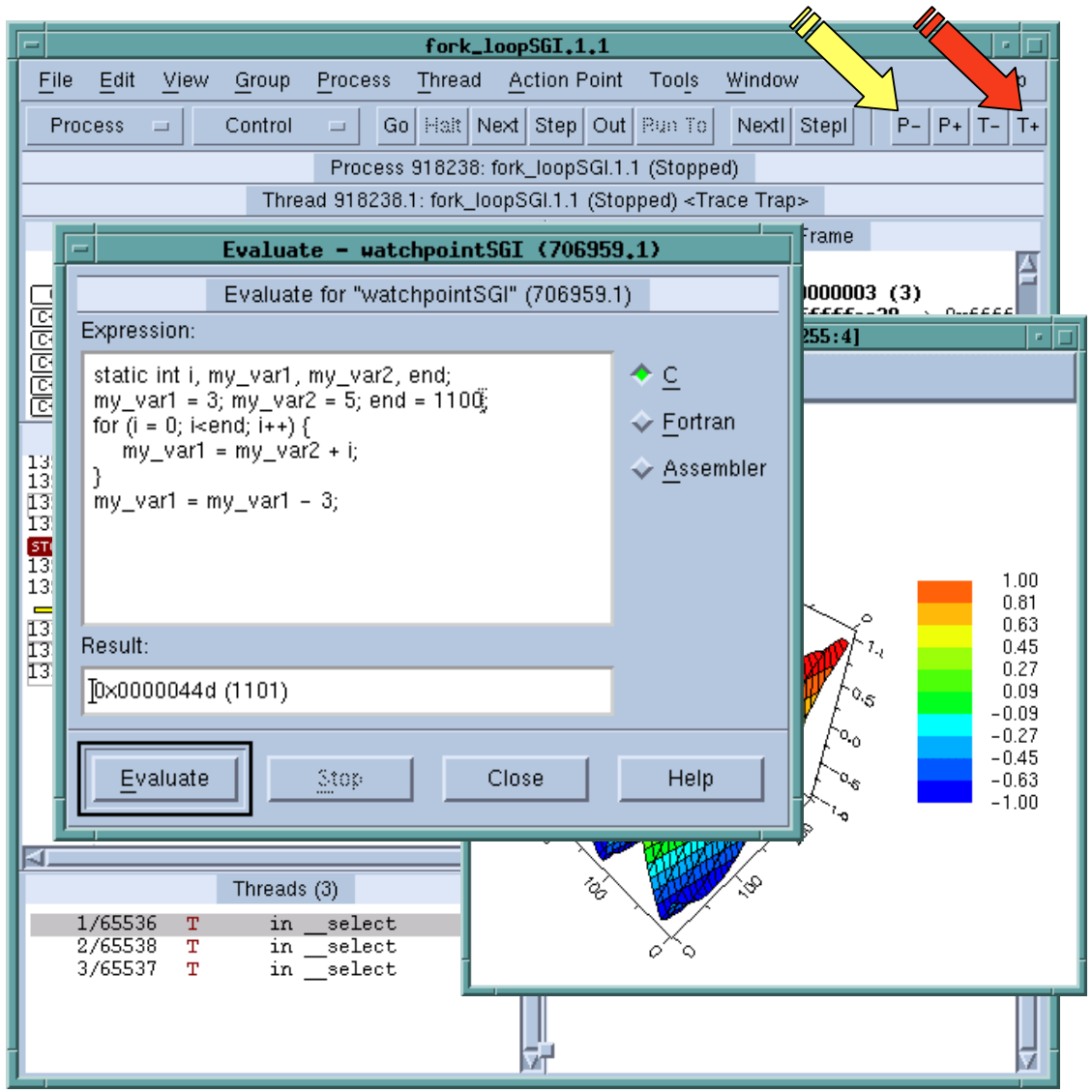
Anforderungen an einen parallelen Debugger

Eigenschaften eines seriellen *command line* Debuggers plus

- Graphische User Interface
- Dynamischer *call tree* (grafisch)
- Darstellung der Nachrichten-Kommunikation (grafisch)
- Übersichtliche Darstellung paralleler Prozesse (> 100 threads)
- Unterstützung verschiedener paralleler Programmierparadigmen
- Kontrollstrukturen zur Synchronisation paralleler Prozesse
- Automatische Start aller beteiligten Prozesse

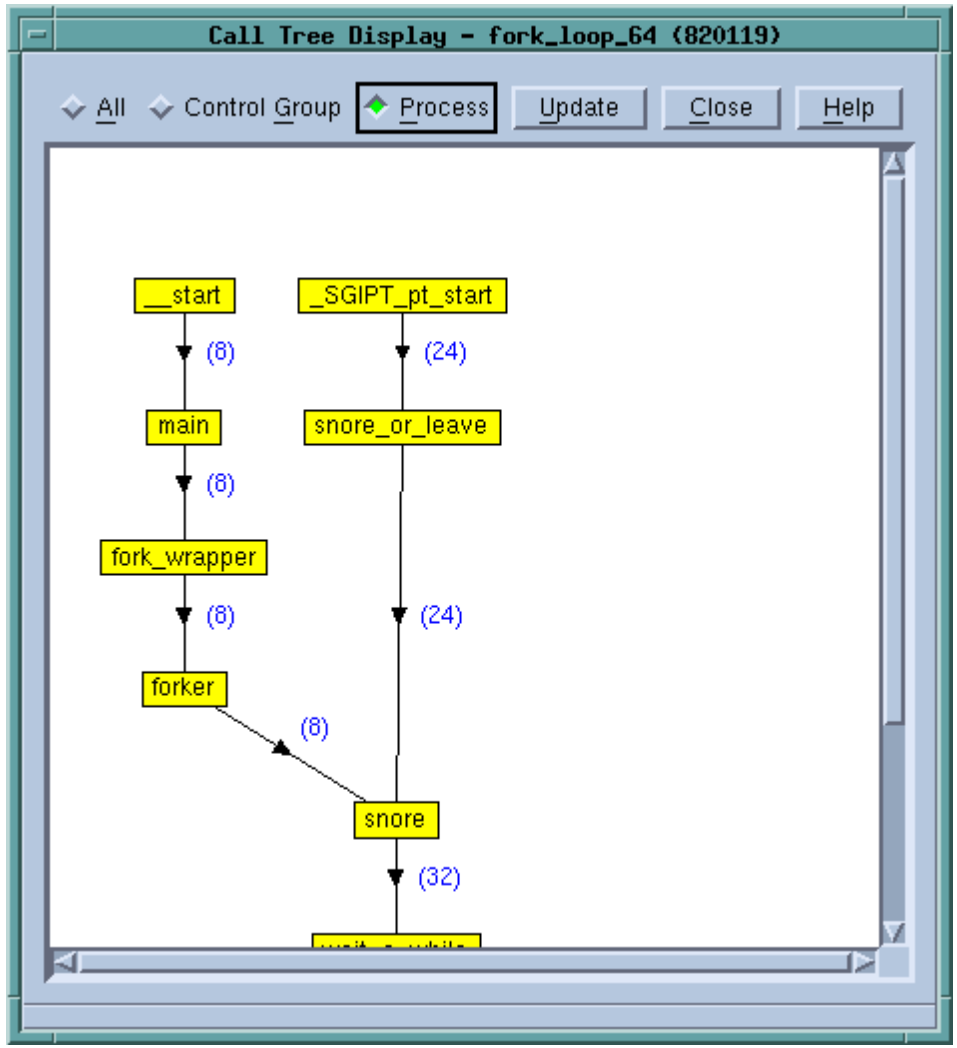
Debugger Features

- Debug-Aktionen
  - Breakpoint
  - Evaluationpoint
  - Watchpoint
- Daten Management
  - Darstellung
  - Statistiken
  - Visualisierung
- Änderungen während des Programmlaufs



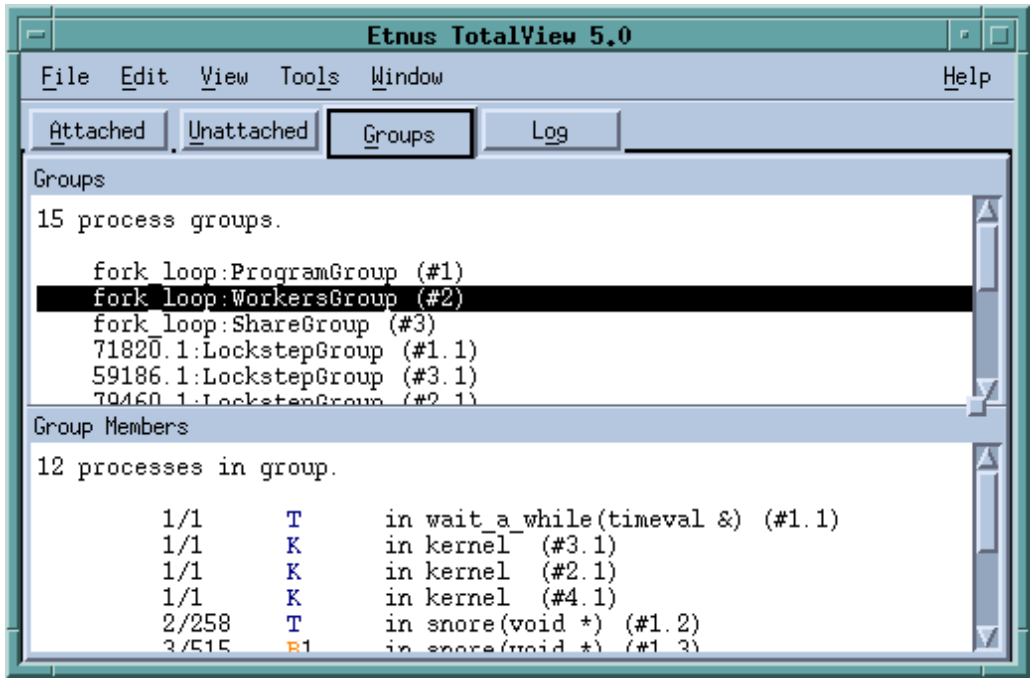
Grafische Darstellung des *call tree's*

- Dynamisch Darstellung des Programmablaufs
- Potentielle Konflikte können visualisiert werden
- Kontrolle und Abhängigkeiten des Programmablaufs



Dedizierte Kontrolle der einzelnen Prozesse

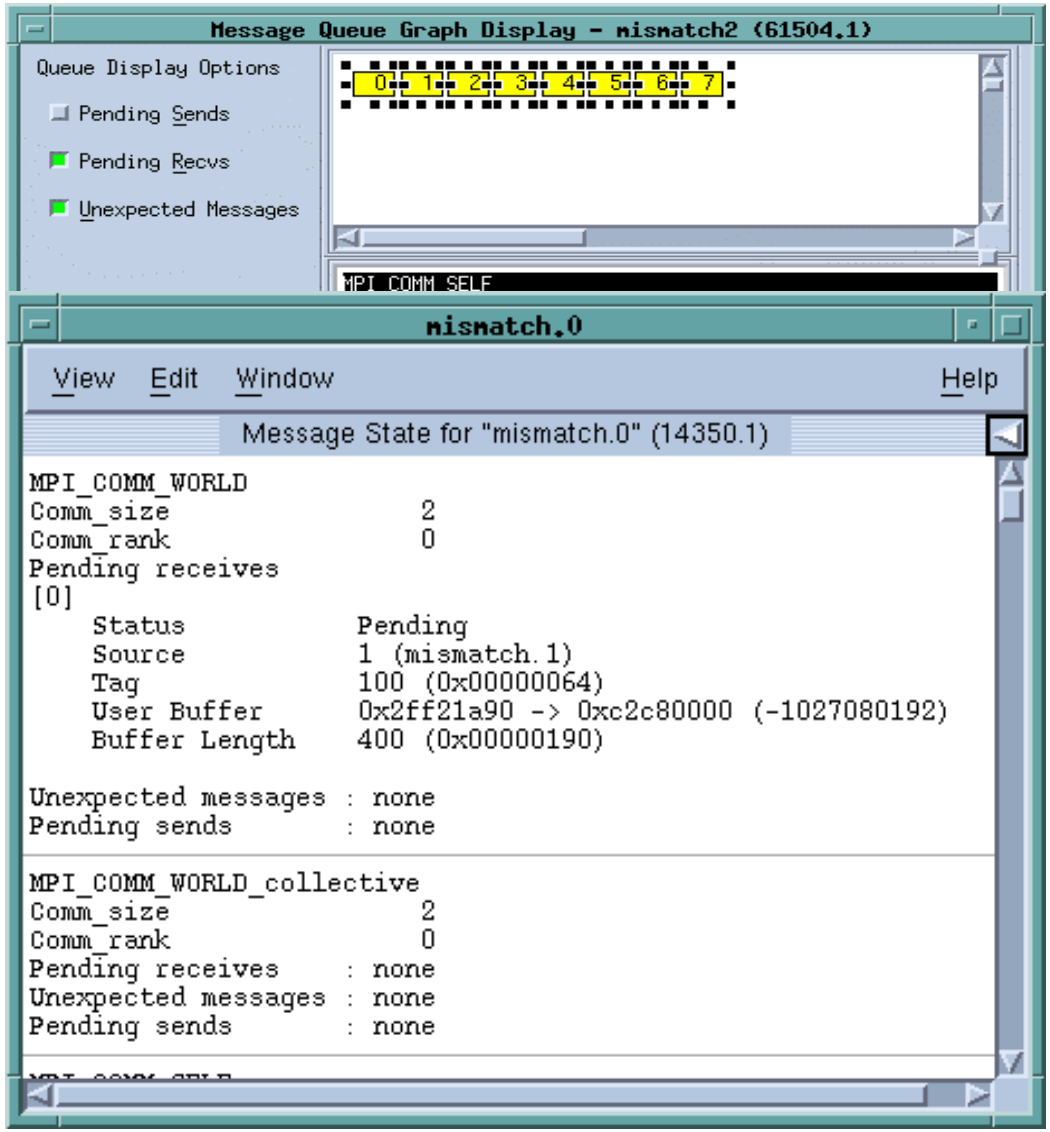
- Kontrolle einzelner Threads und Ablaufkontrolle auf Threadlevel
- Beliebige Gruppierung von Threads und Prozessen
- Dynamische Gruppierung von Threads und Prozessen
- Definition von Ablaufkriterien für Prozessgruppen
  - breakpoints
  - barrier
  - …



Visualisierung der Kommunikation

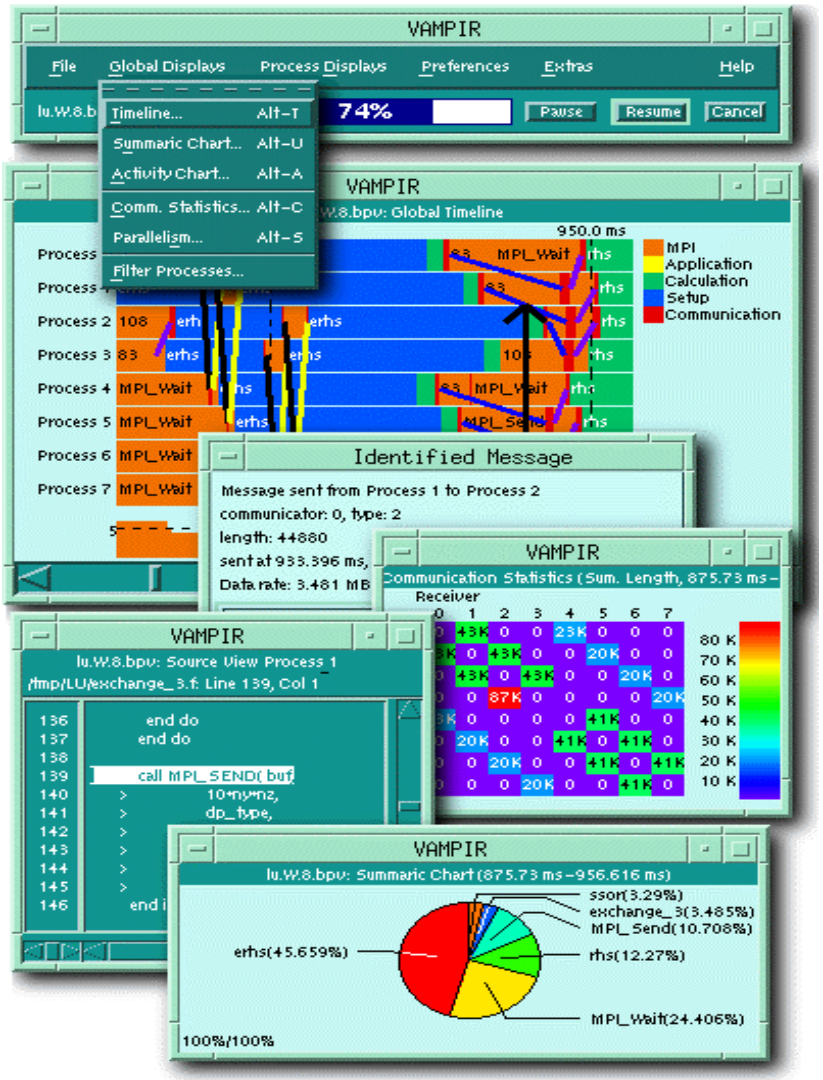
- Darstellung der Nachrichtenwege zu einem definierten Zeitpunkt
- Inter-Prozess Kommunikation
- Visualisierung von Kommunikations-Problemen

  (hängende, unerwartet Nachrichten)



pallas

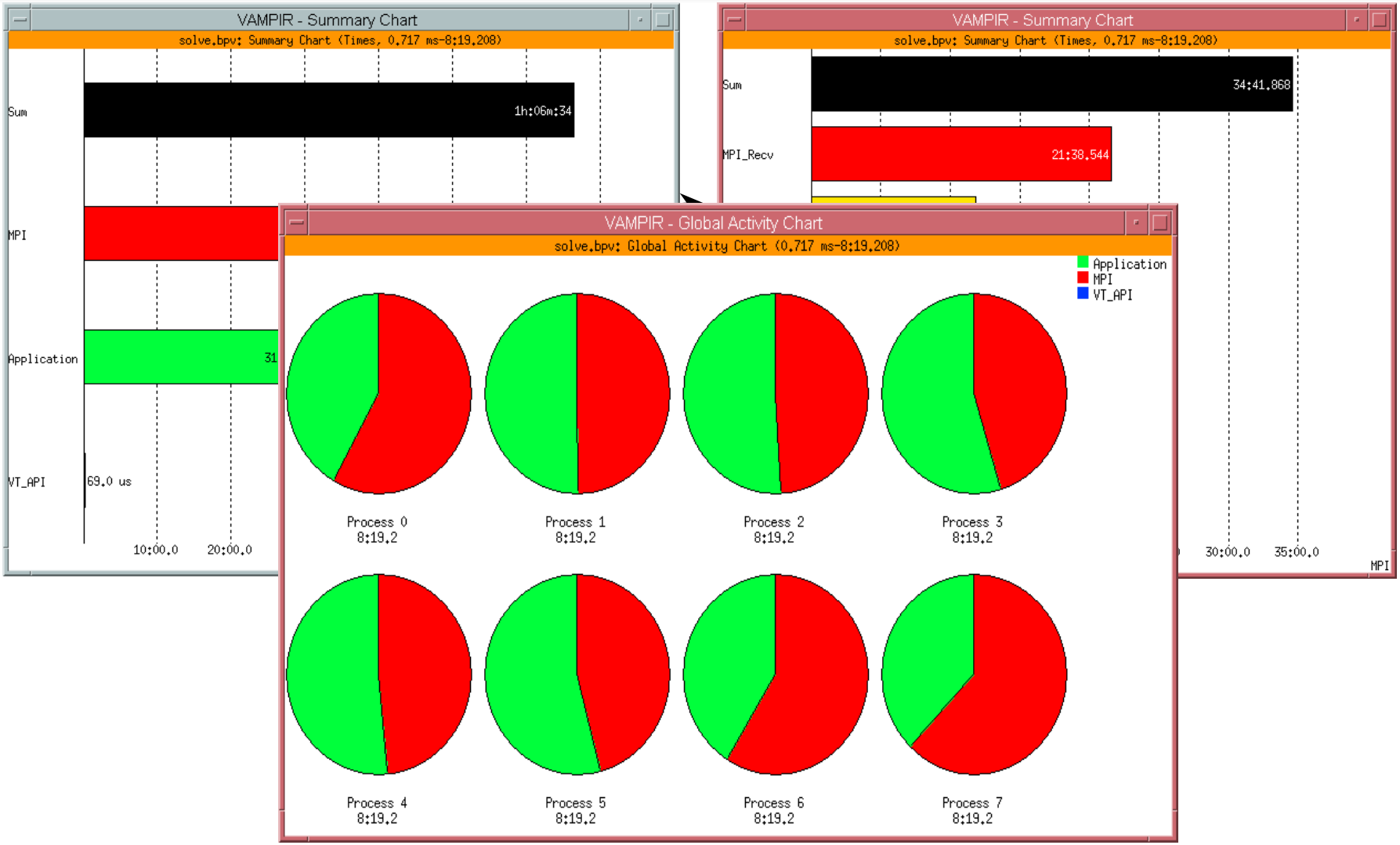
MPI Performance Analyse



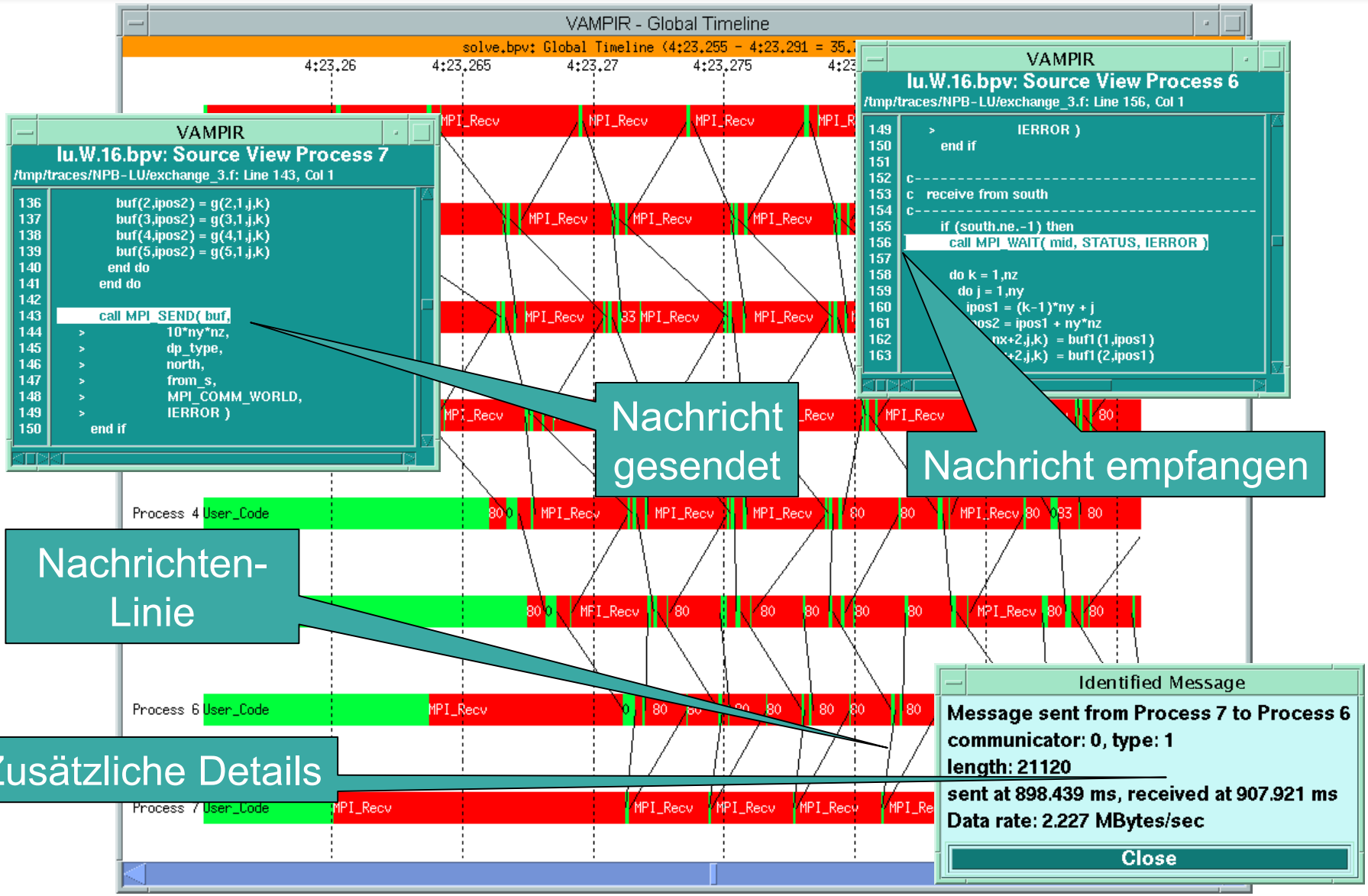
Anforderungen an ein Performance Analysetool

- Analyse von MPI und User Routinen des optimierten Programms
- Statistische Analyse von Programm und Kommunikation
- Darstellung der Lastverteilung
- Einfache und schnelle Handhabung
- Filtermöglichkeiten (Prozesse, Nachrichten, kollektive and MPI–I/O Operationen)
- Geringen Einfluss auf die Applikations-Performance

Statistik des Programmablaufs

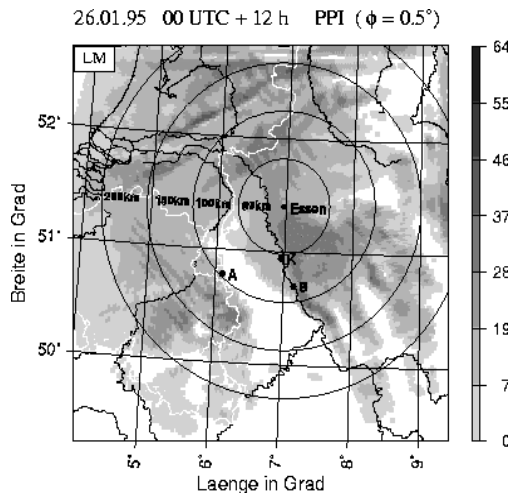


Detail Information



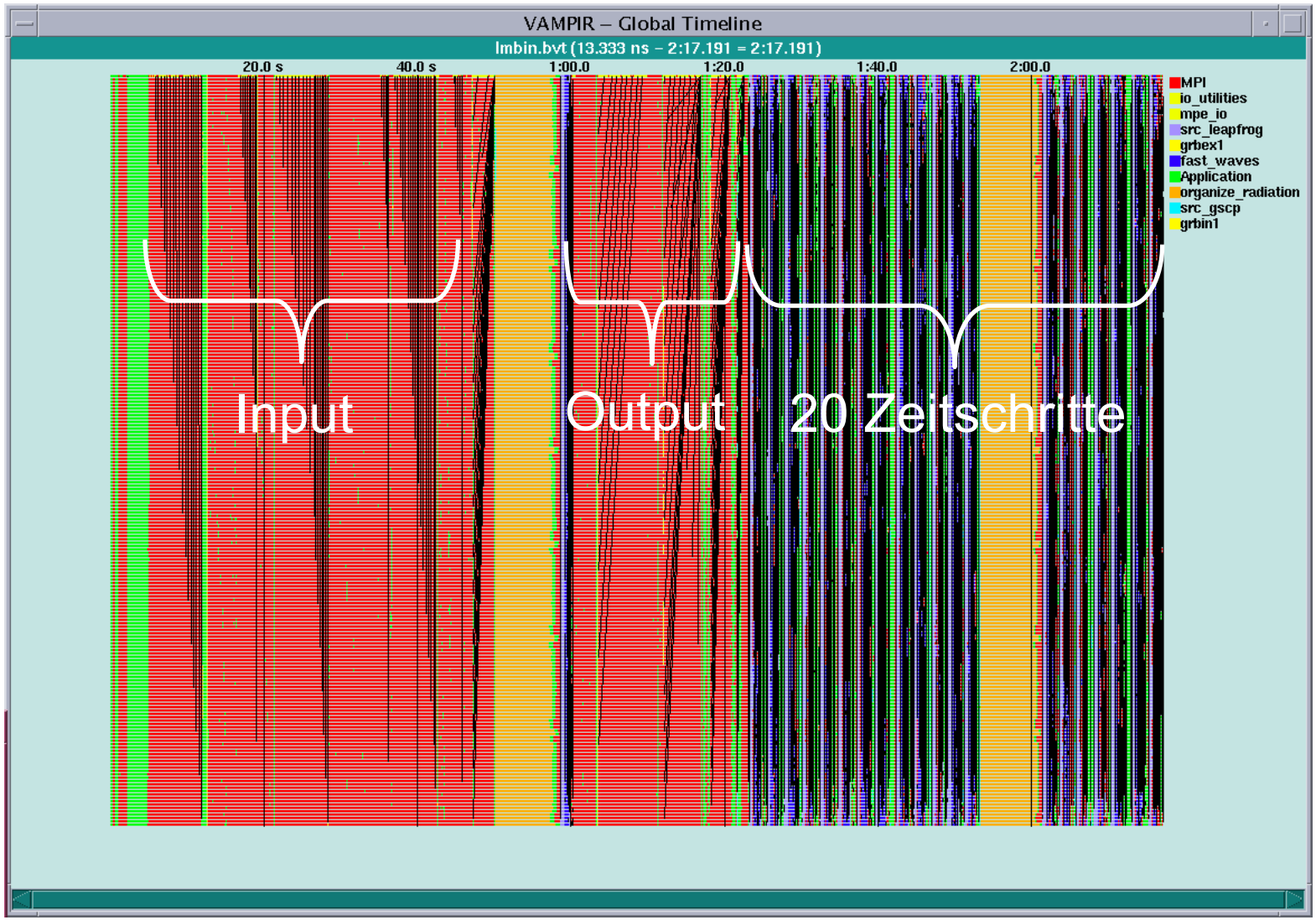
Fall Studie - LM

LM (Lokal Modell) ist das Programm des Deutschen Wetterdienstes (DWD) für die Wettervorhersage im Skalenbereich von 1-10 km.



- Untersucht wurden :
  - Einfluss von Prozessorgitter zu Knotengitter Aufteilung
  - Überlappung von I/O und Berechnung

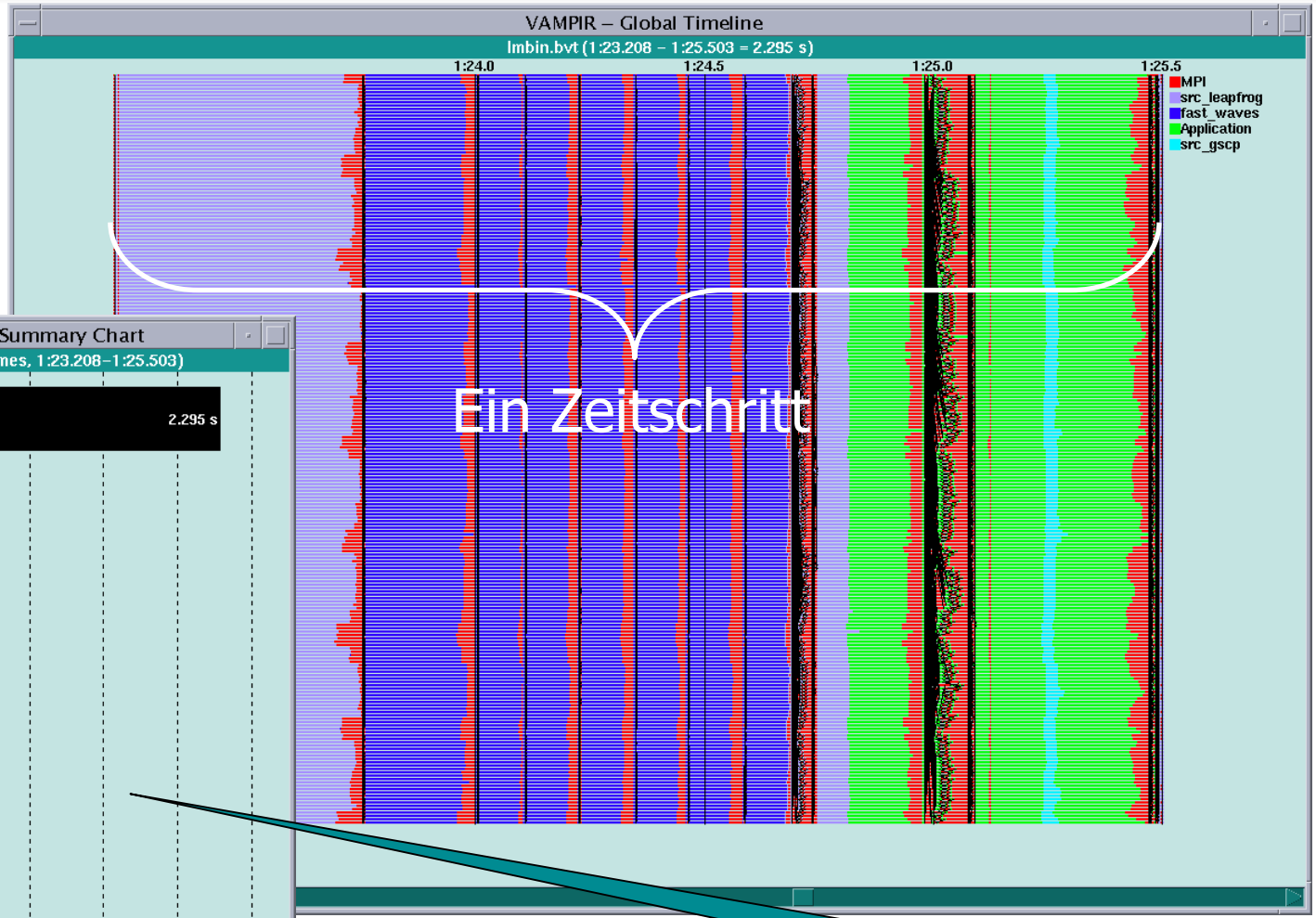
Trace LM - Test Fall auf 224 Prozessoren



LM - Laufzeit Phase

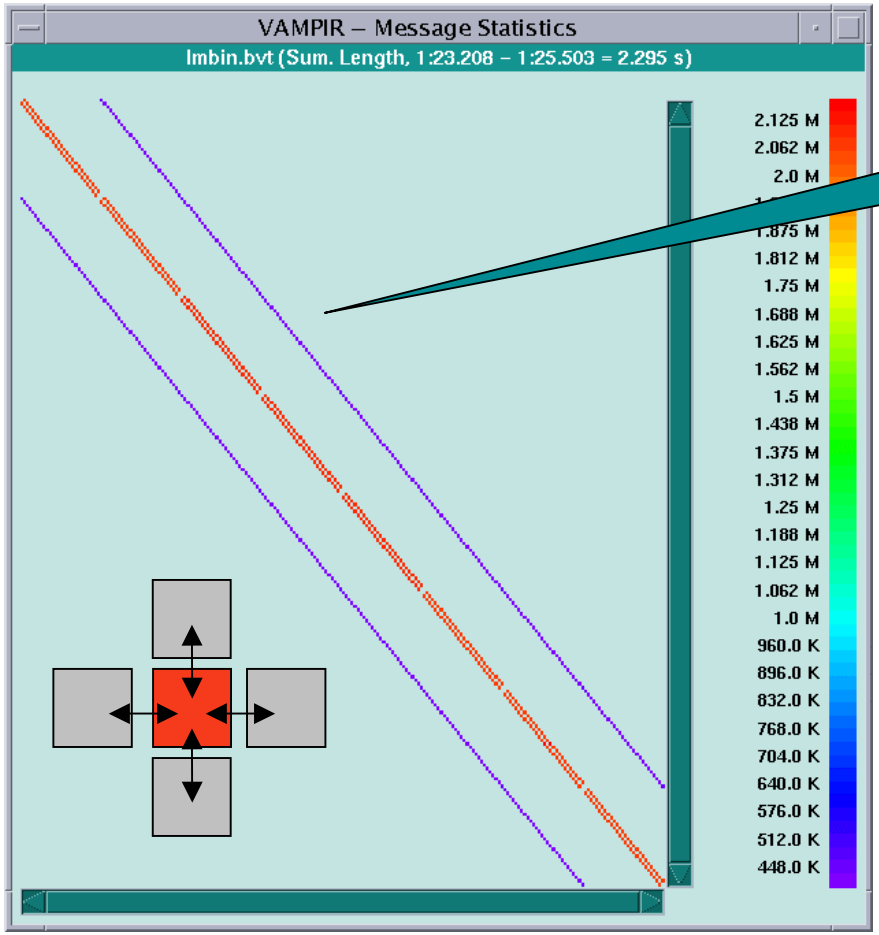
- Input
  - Dauer beim Start von LM : 45 s
- Zeitschritte
  - Dauer ca. 2.3 s
  - Auswertung der Kommunikation Performance
  - Datenverteilung: Prozess→Knoten Aufteilung
- Output
  - Dauer ca. 20 s beim Start
  - Ca. 20 s nach jedem Zeitschritt

LM – Zeitschritt Analyse

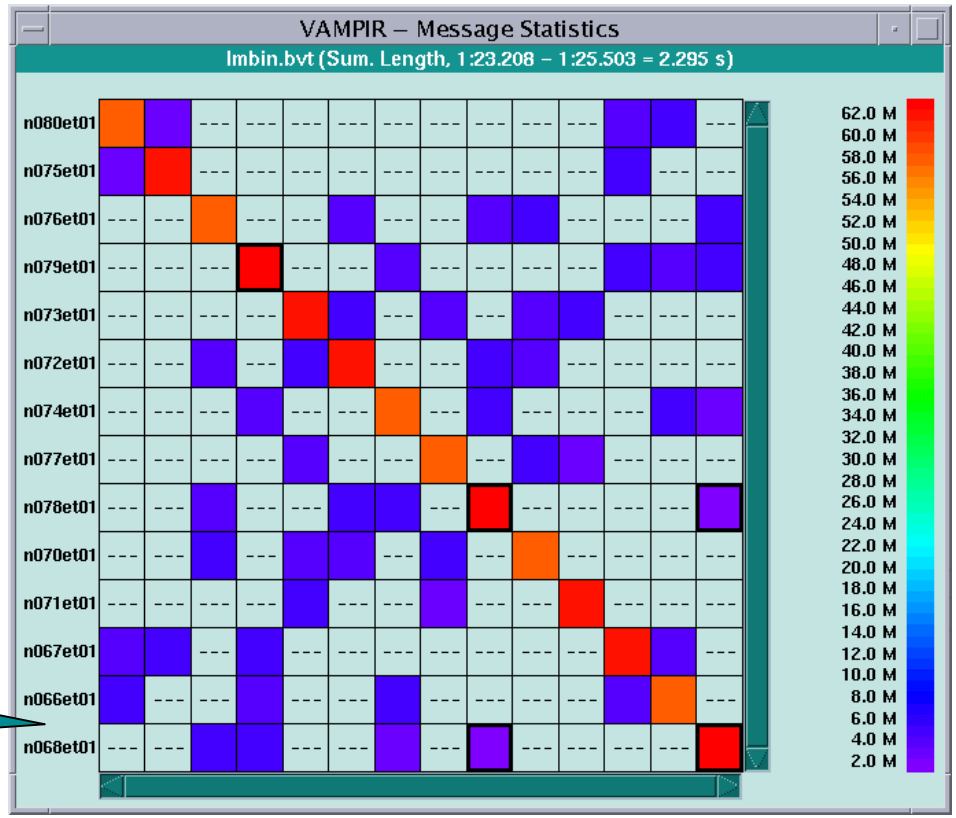


Statistik über alle Prozesse

LM – Knotenmapping Analyse



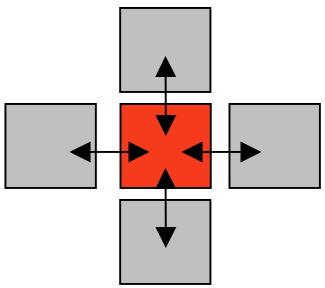
Kommunikation zwischen den Prozessen

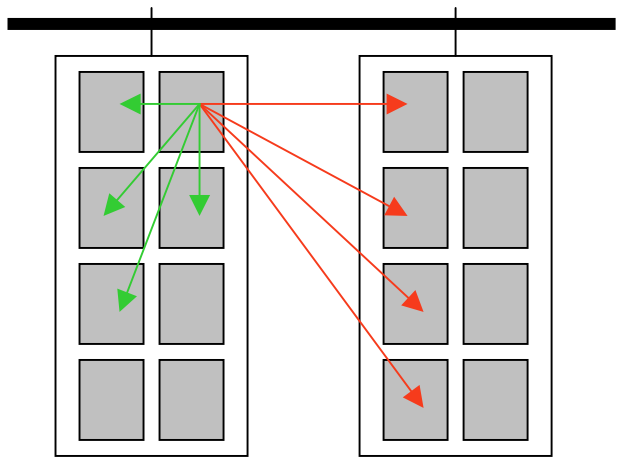


Kommunikation zwischen den SMP Knoten

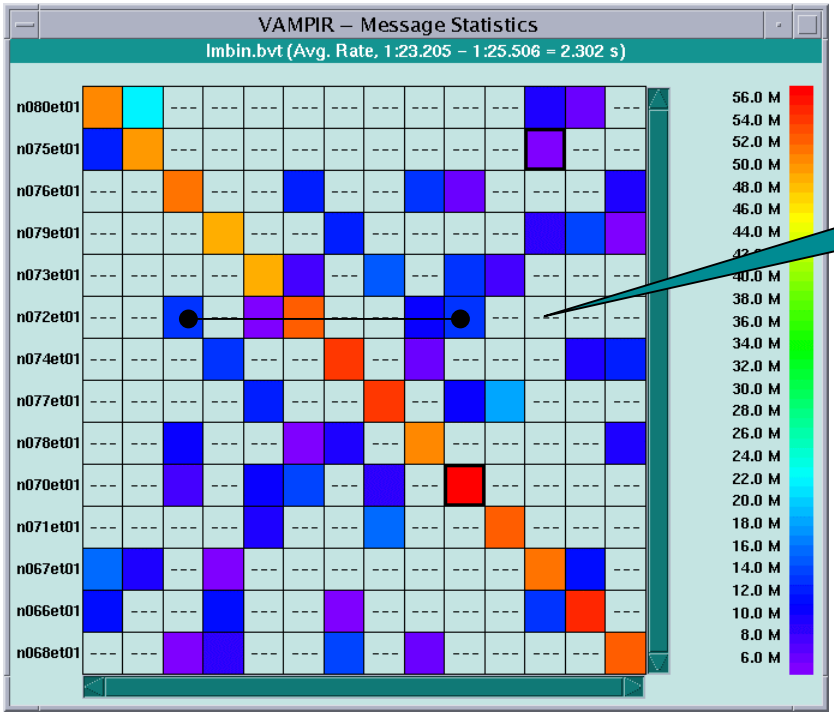
LM – Zeitschritt Analyse

- MPI Kommunikation beträgt ca. 16% der Laufzeit
- Lastverteilung ist gut (geringer Kommunikationsoverhead)
- Kommunikations-Matrix
  - Typische 2–D Nachbarkommunikation
  - Zuviel off–node Kommunikation
- Lösungsansatz: Verbessertes Mapping durch Inner-Knoten-Kommunikation (*shared memory **statt** interconnect*)

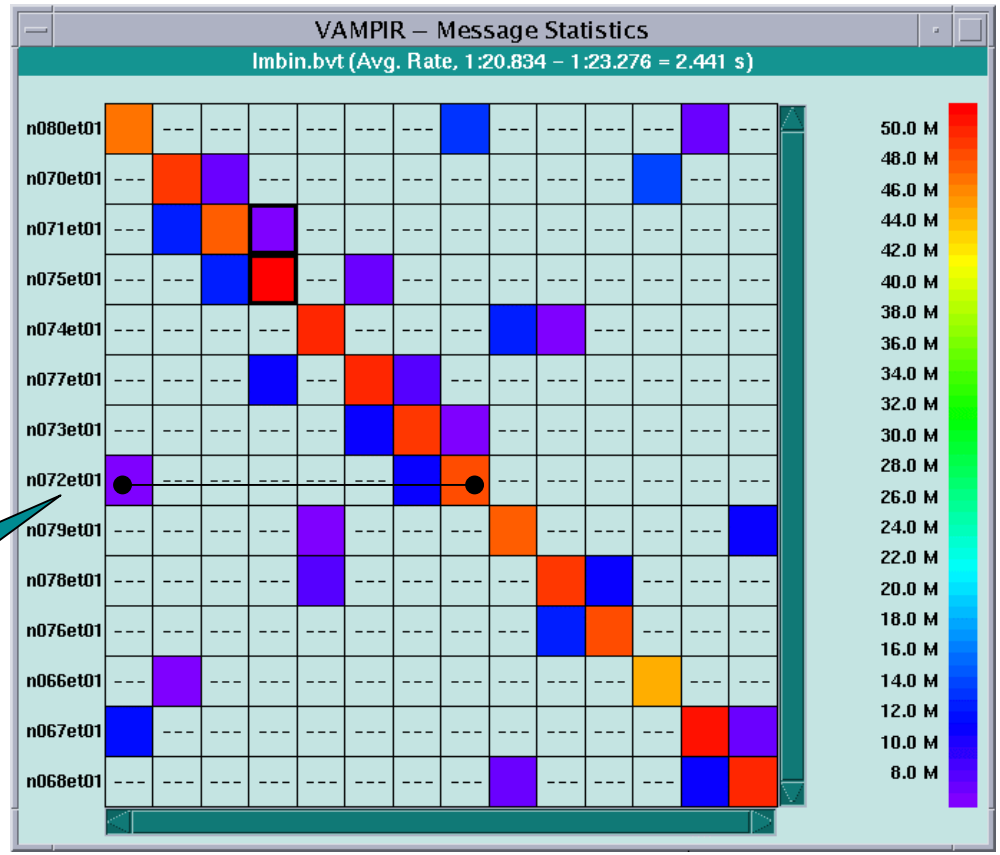




Verbesserung des Prozessor Mappings

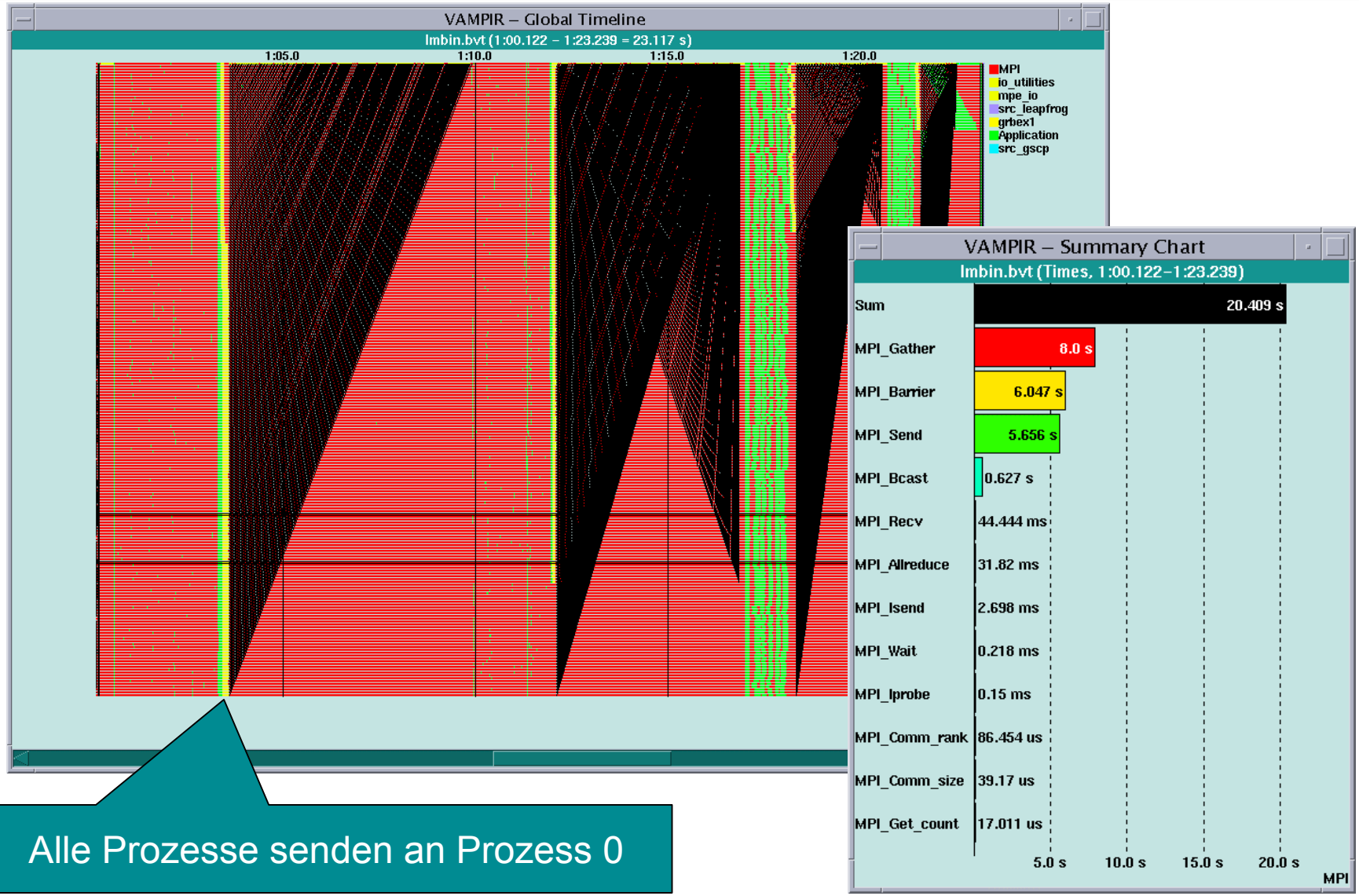


Ursprüngliches Knotenmapping



Verbessertes Knotenmapping

Analyse der Output Phase

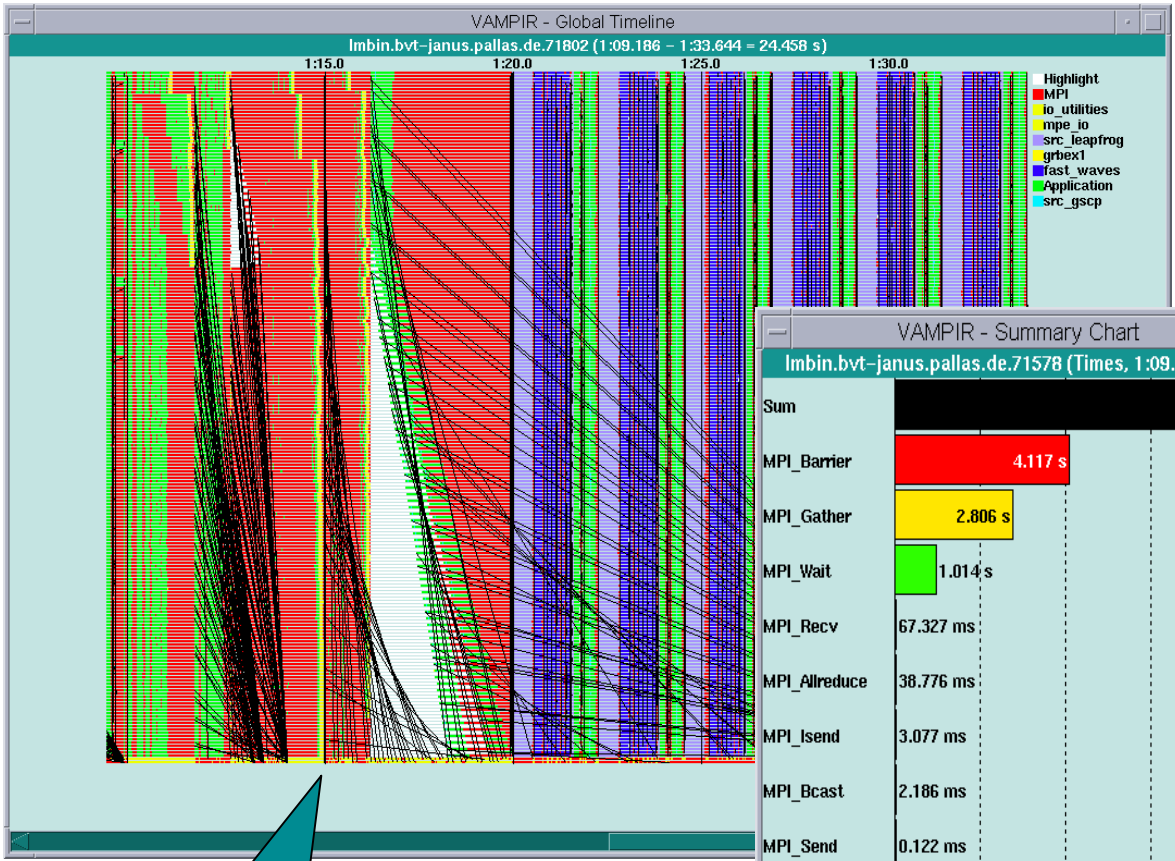


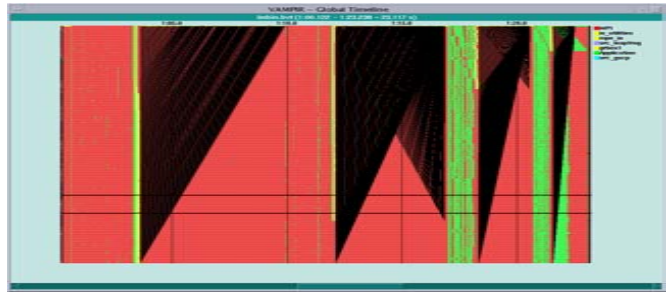
Alle Prozesse senden an Prozess 0

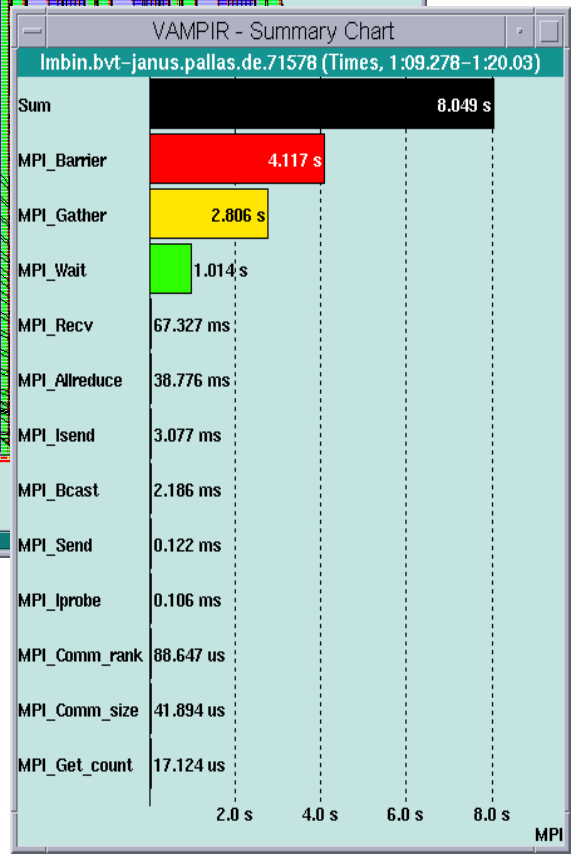
Analyse der Output Phase

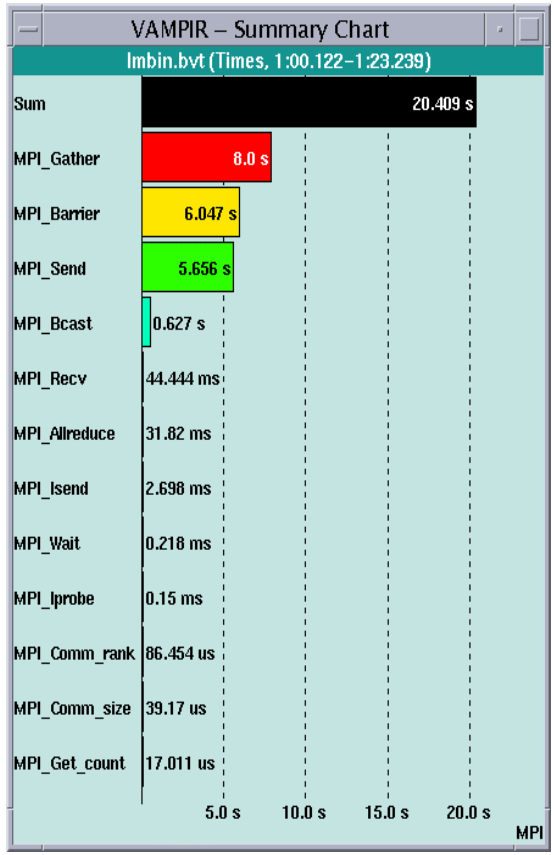
- Das Schreiben des Output blockiert alle Prozesse und benötigt soviel Zeit wie 8 Zeitschritte
- Der gesamte Output ist somit über Prozess 0 serialisiert
- Lösungsansatz: Verwendung eines dedizierten I/O Knotens
  - Jeder Prozess sendet seine Output Daten an den I/O Knoten und rechnet weiter
  - Der I/O Knoten führt die Festplattenzugriffe durch während die Berechnung weiter geht

Analyse der Output Phase – I/O Knoten





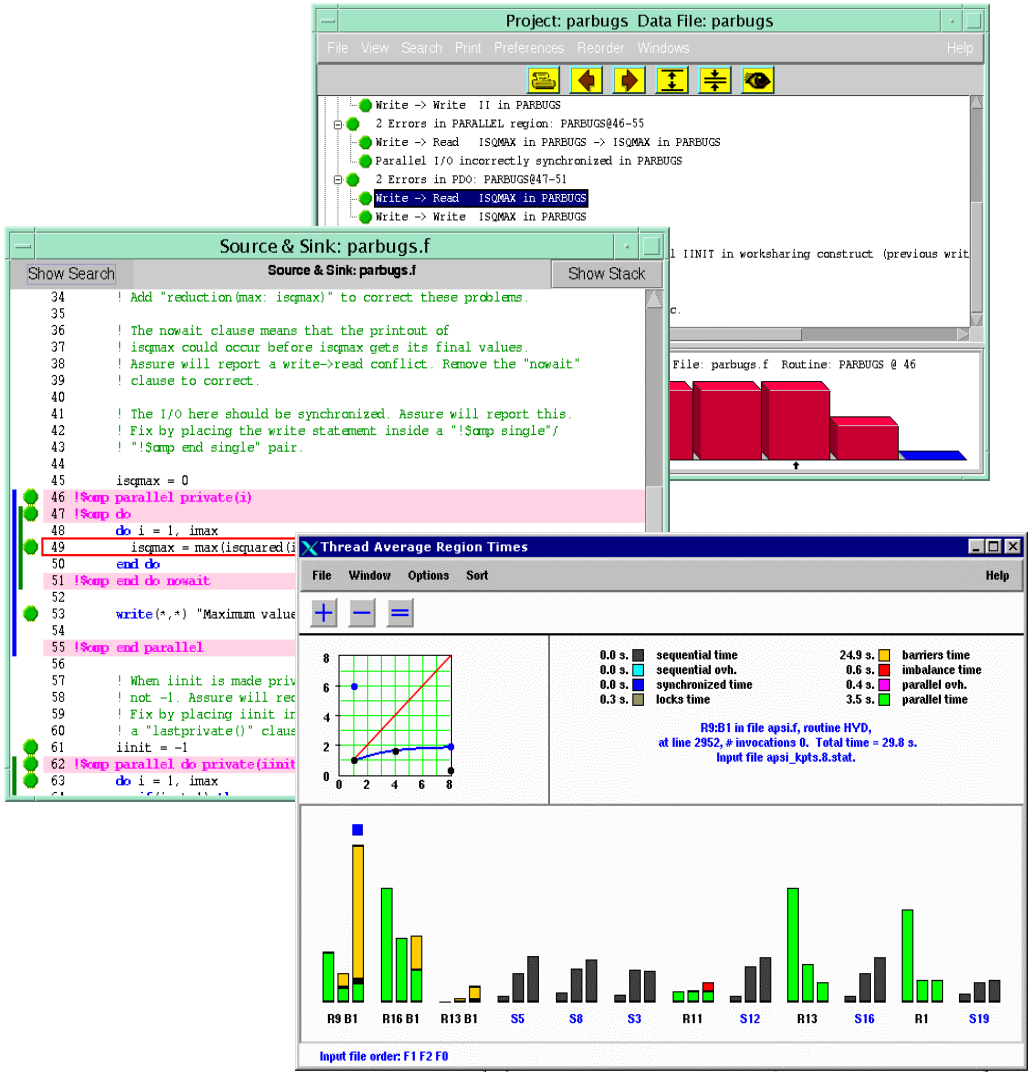




Überlagerung Berechnung- I/O

pallas

OpenMP Analyse



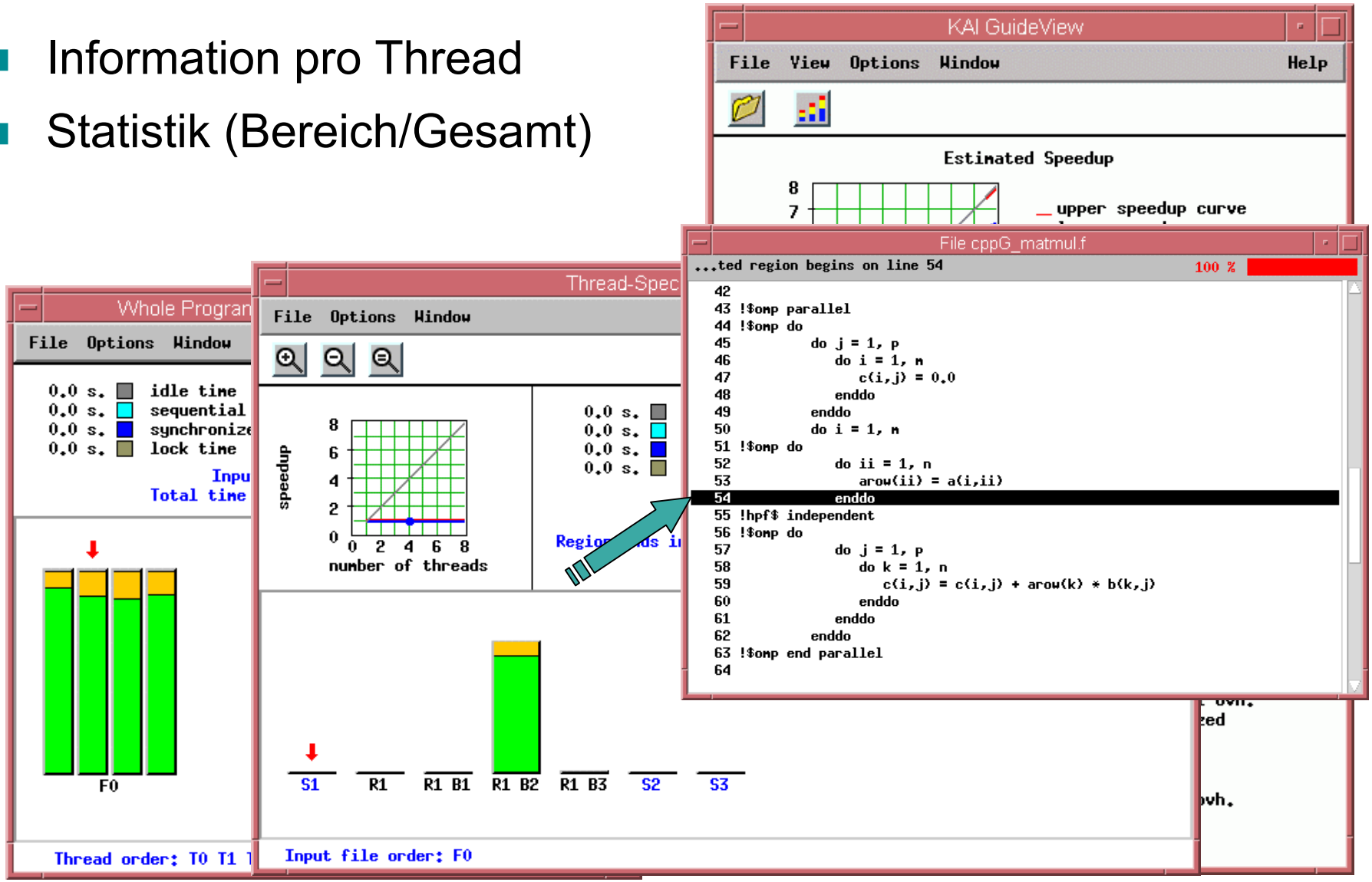
Anforderungen an ein OpenMP Analysetool

- OpenMP Performance Probleme
  - Probleme bei der Lastverteilung
  - Synchronisation Probleme
- Darstellung folgender Faktoren
  - Effektiver *speed up*
  - Ausführungszeitaufteilung (Parallel, Seriell, Overhead)
- Vergleich verschiedene Programmläufe

Anzahl der verwendeten Threads wird einfach über ein Umgebungsvariable getriggert !

OpenMP Beispiel - Matrixmultiplikation

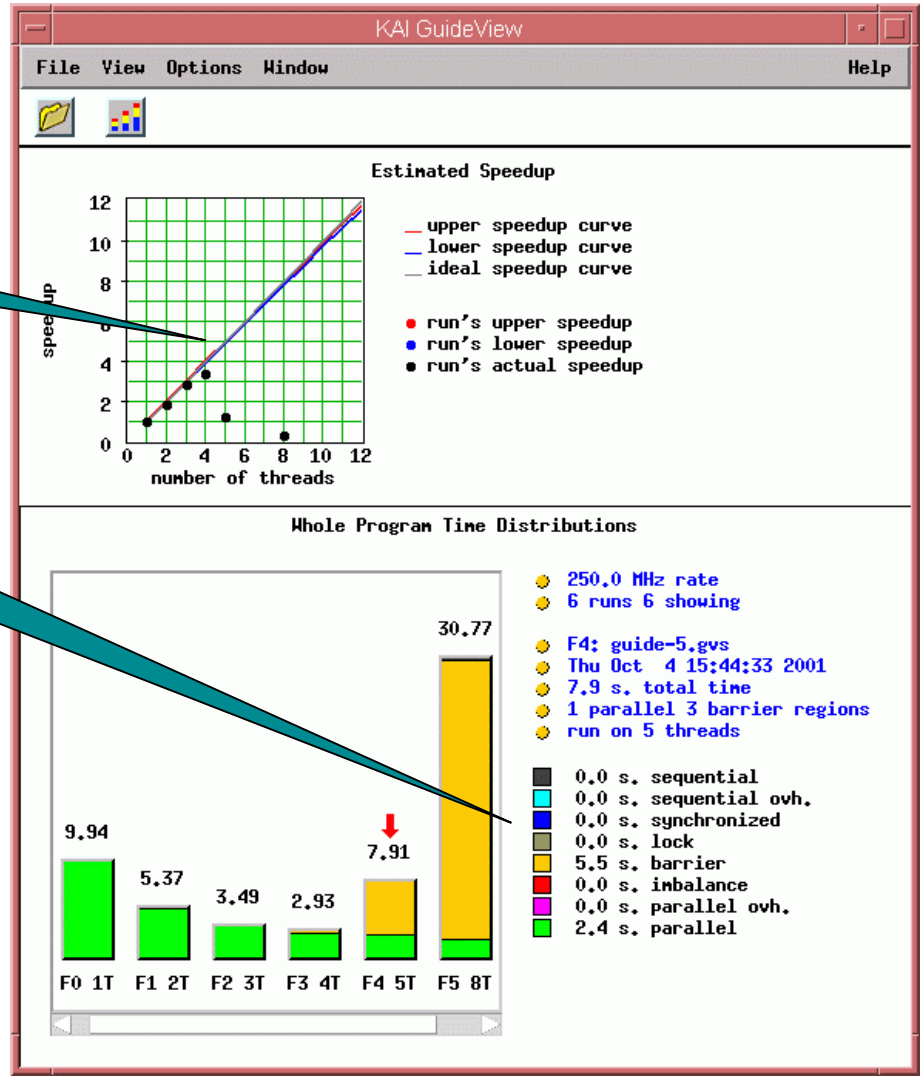
- Information pro Thread
- Statistik (Bereich/Gesamt)



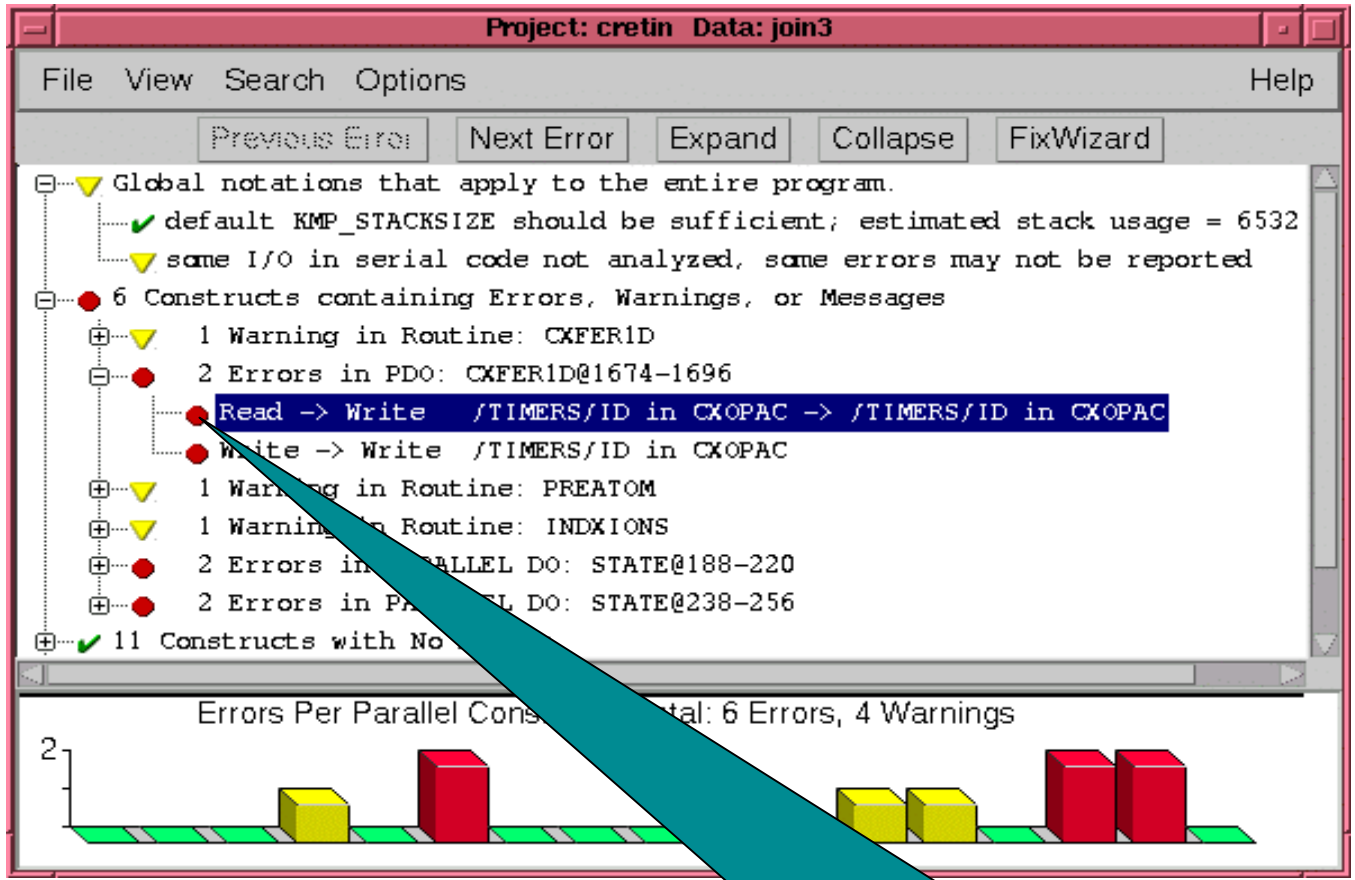
OpenMP Beispiel - Matrixmultiplikation

Vergleich idealer-tatsächlicher Performance – *speed up*

Identifizierung von bottlenecks (*barriers, locks, seq. time*)

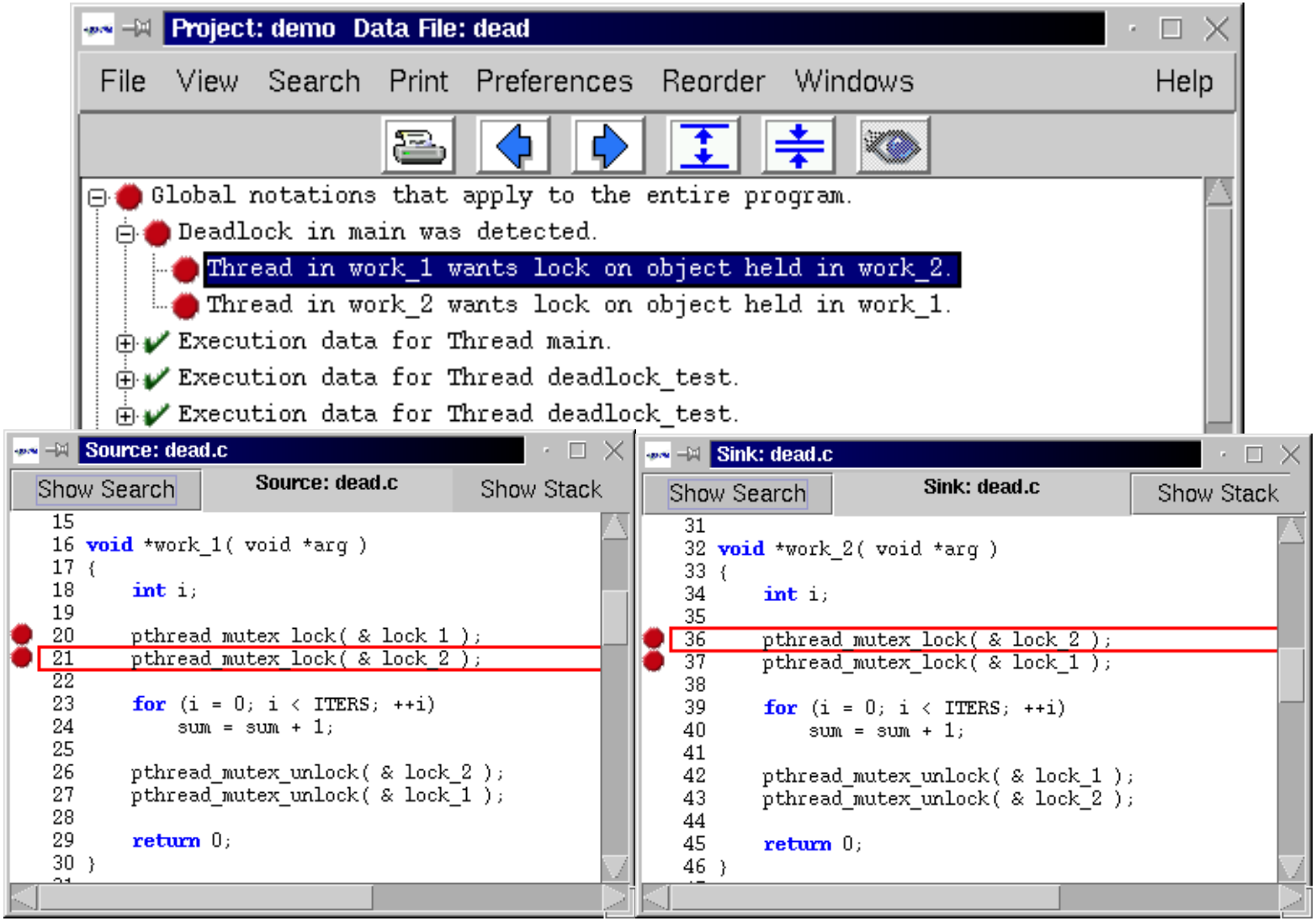


Thread Analyse – parallele Probleme

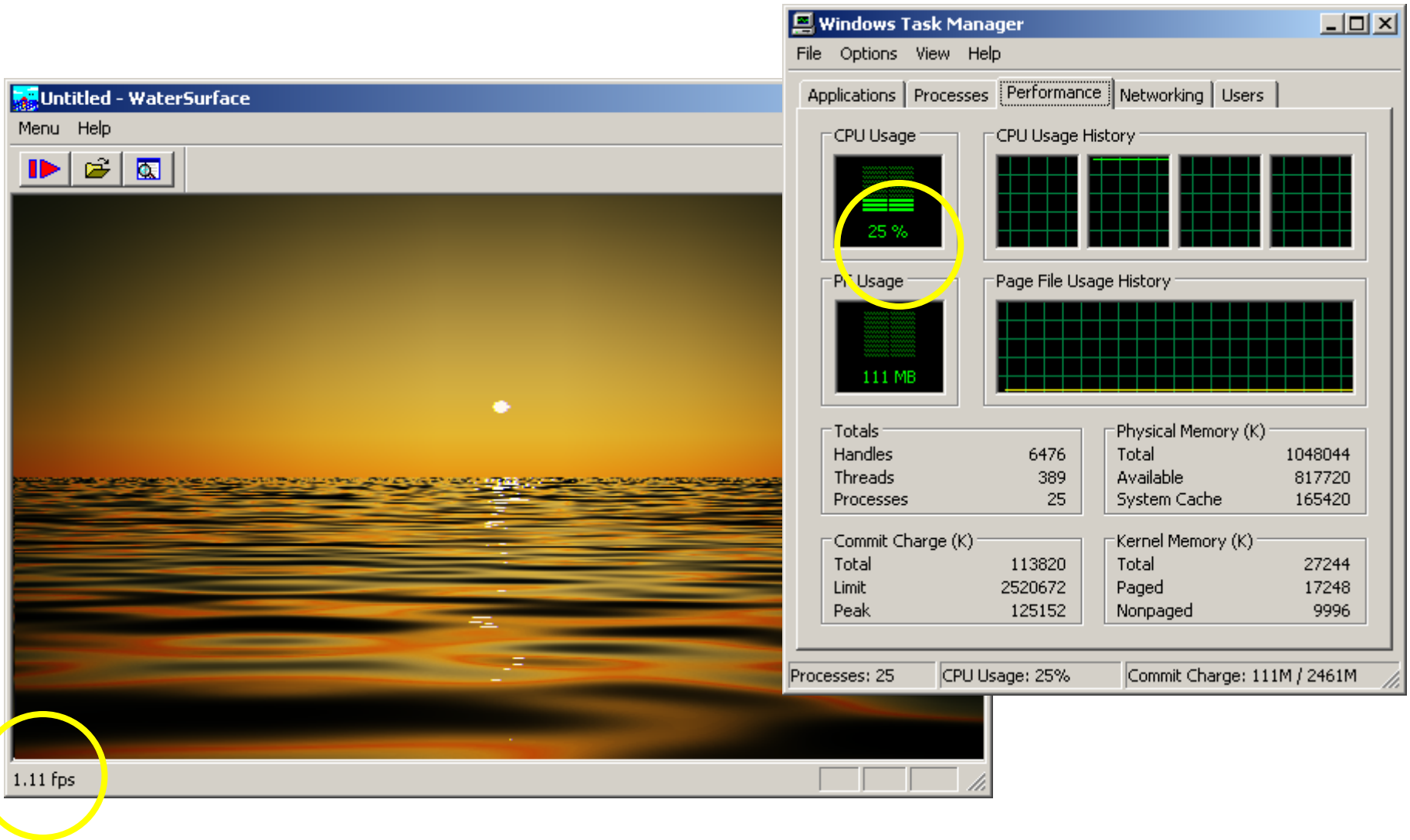


Synchronisationsprobleme (*memory leaks*)

Thread Analyse - *deadlock*

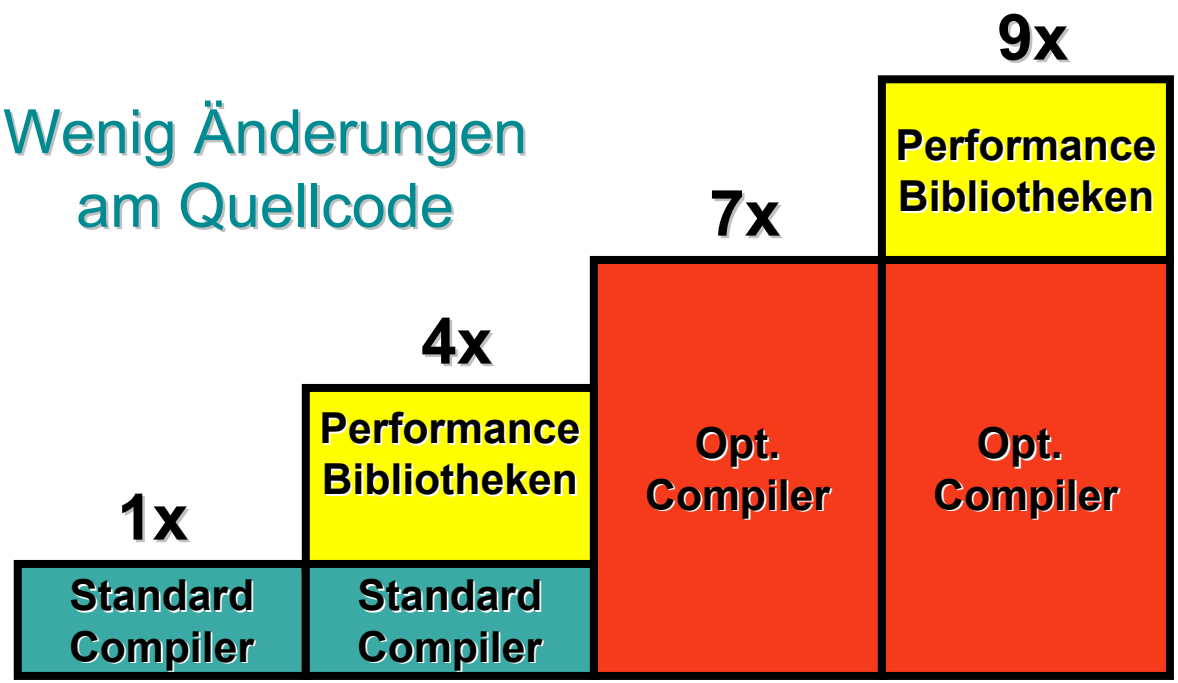


Basis Performance – nicht optimierter Code

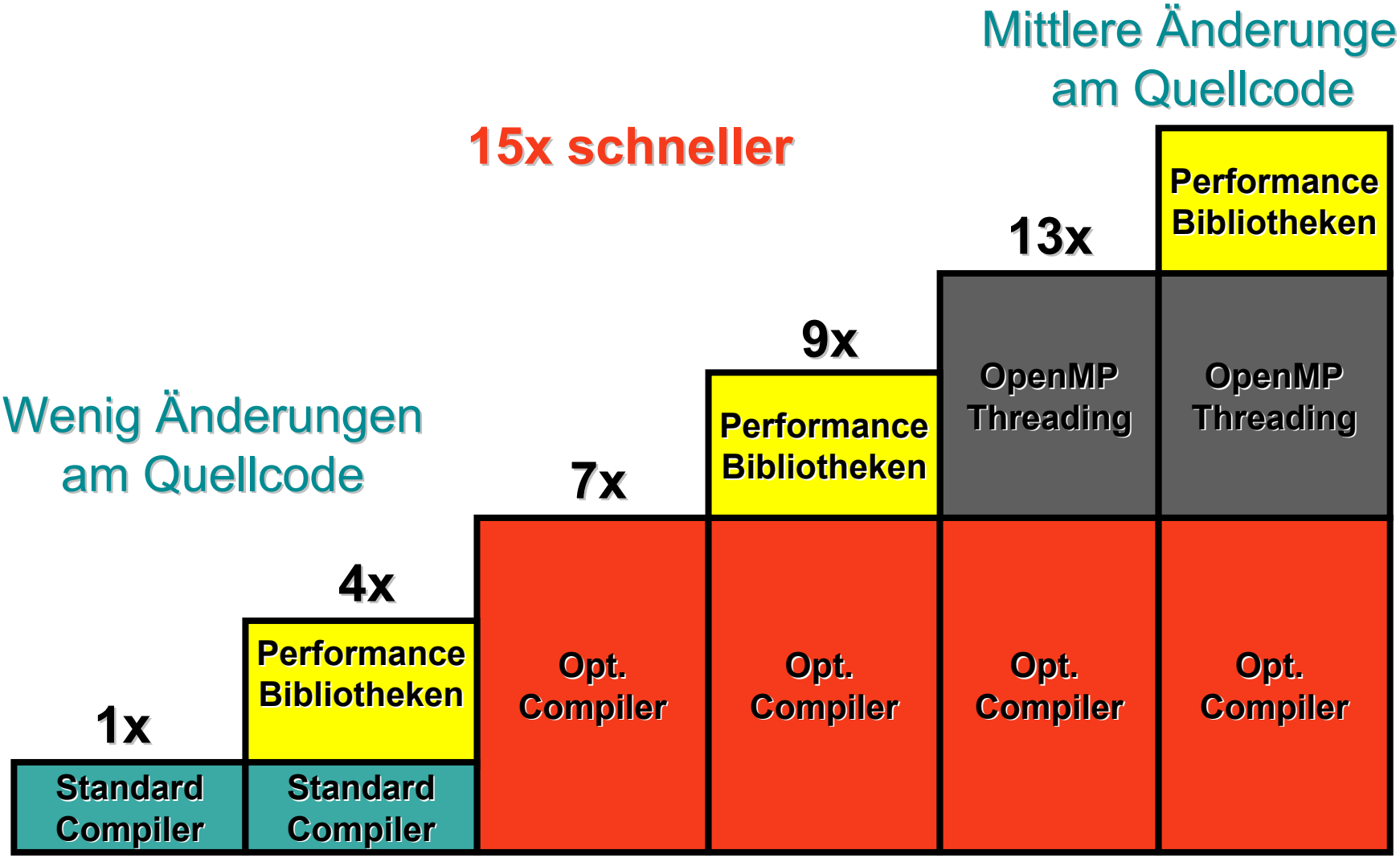


Optimierter serieller Code

9x schneller

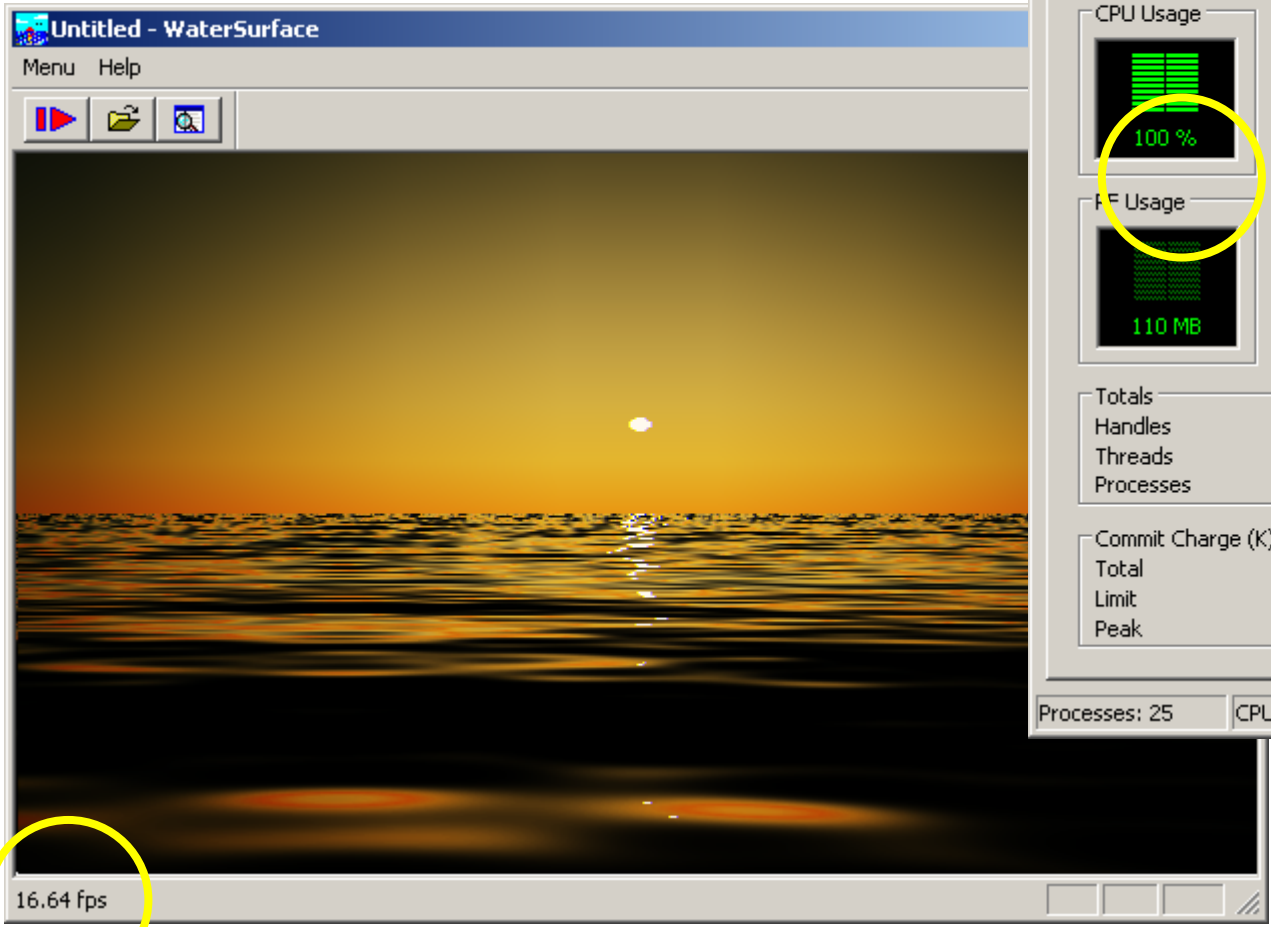


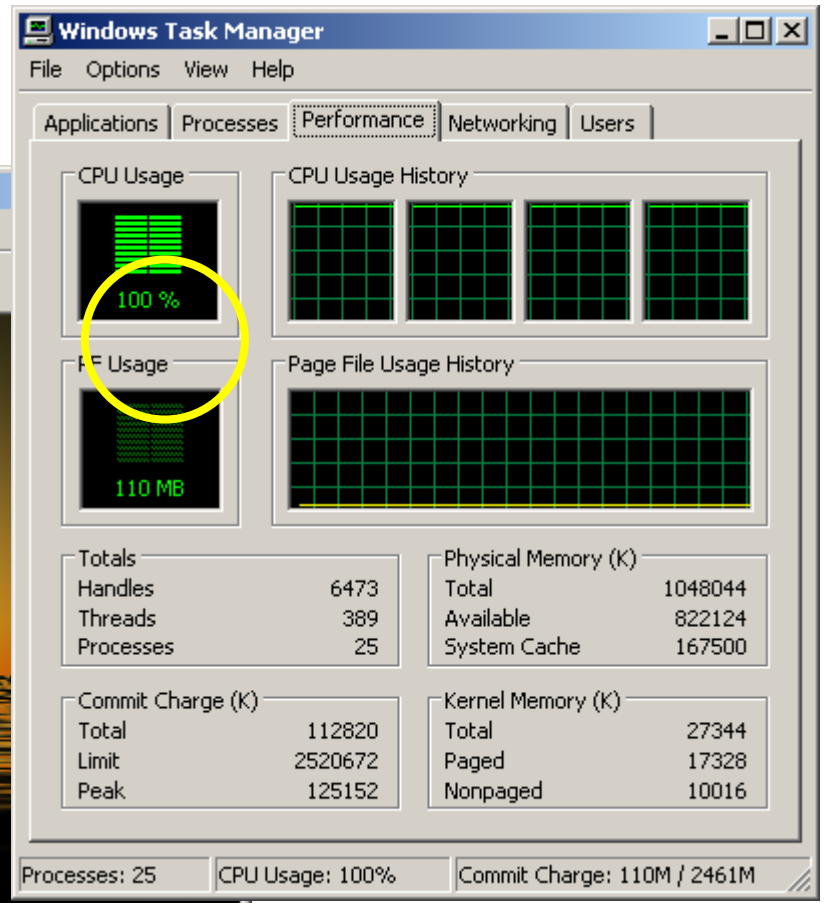
Optimierter paralleler Code - OpenMP



Optimierte parallel Performance

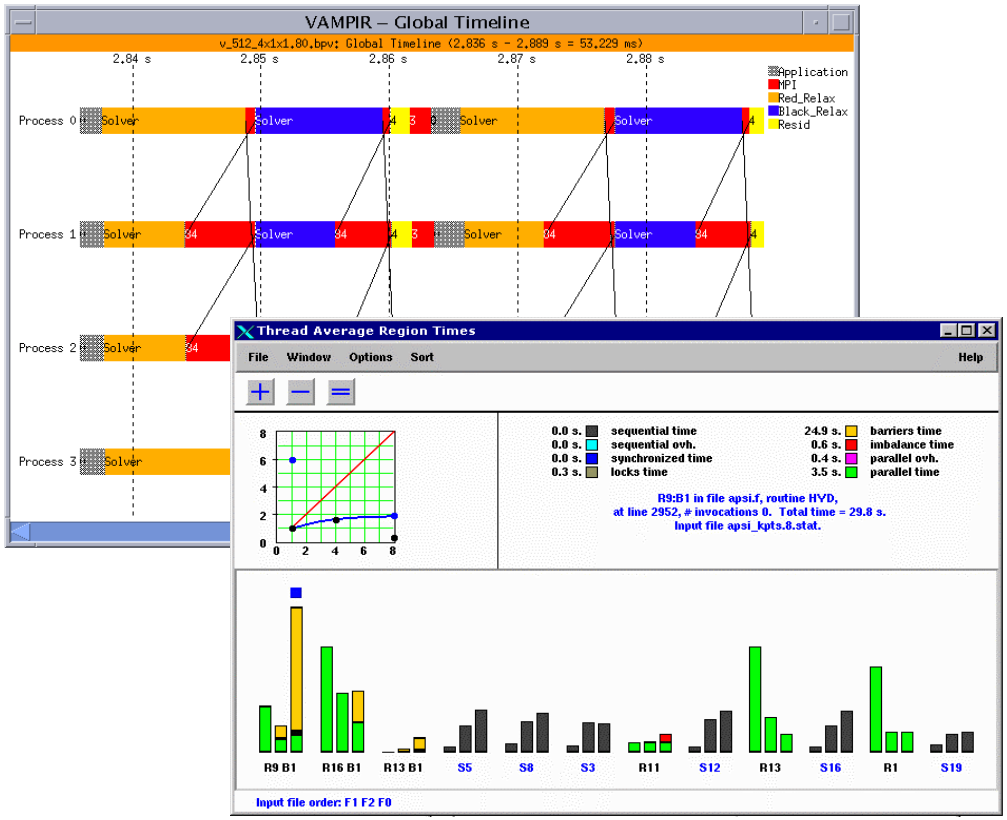
15x schneller



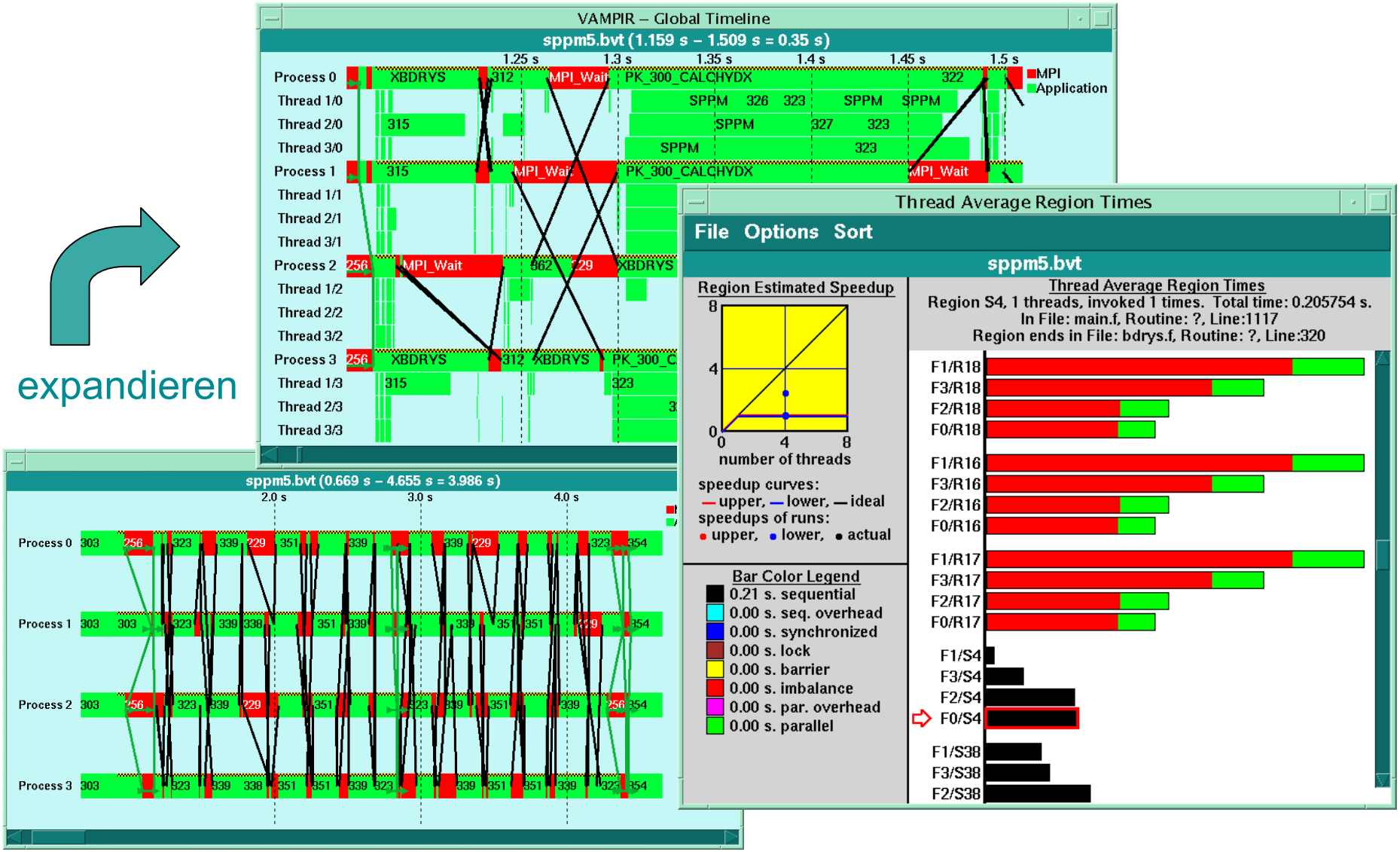


pallas

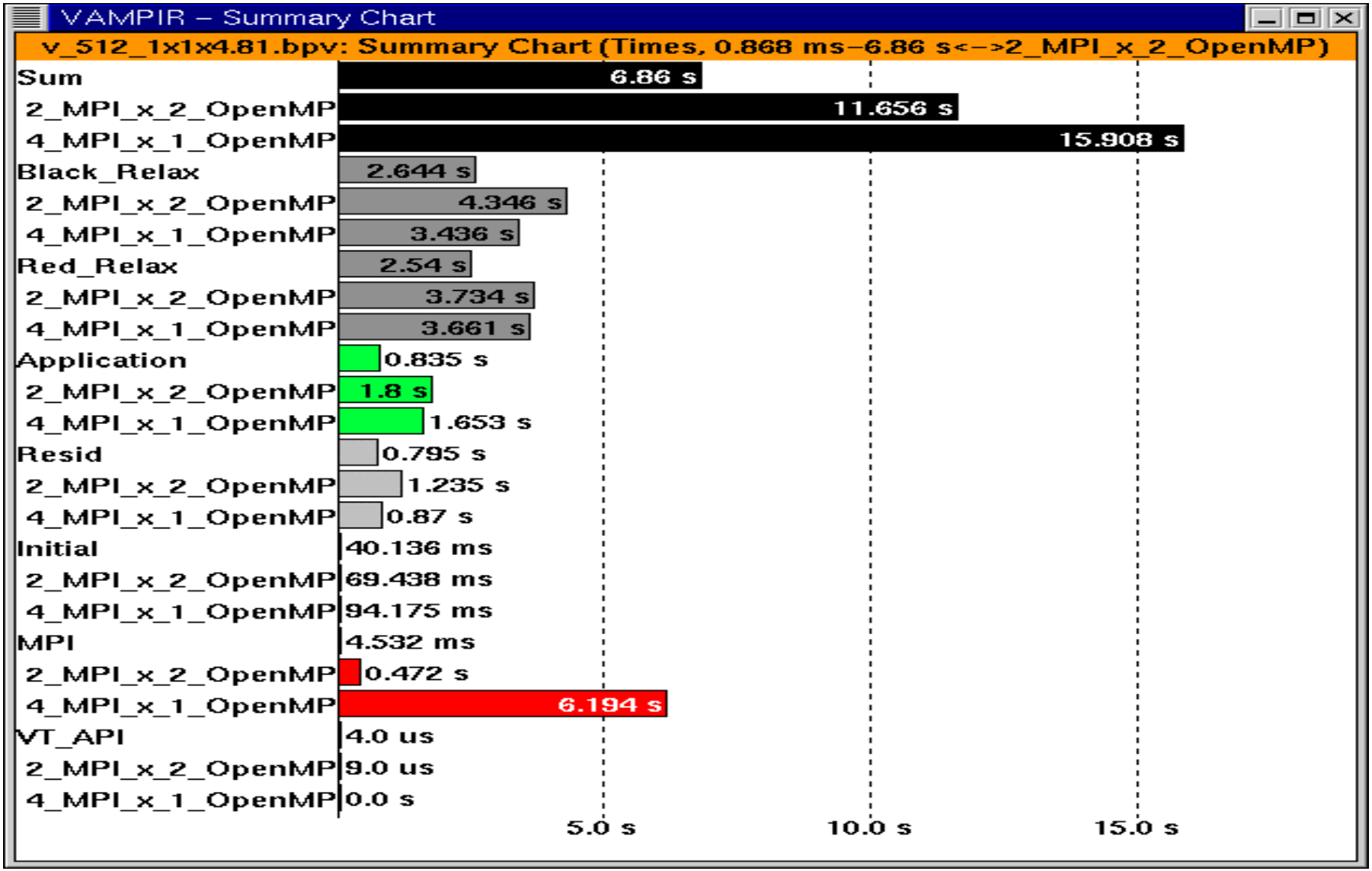
Hybride Modelle



MPI – OpenMP Analyse



MPI+OpenMP – verschiedene Programmläufe



Einsatzgebiet der Werkzeuge

SMP PC

ASCI-Q 30 Teraflop System HP

- SMP System
- 2-4 CPUs
- 4 GB Hauptspeicher
- 100 GB Festplattenspeicher

- 374 * 32-way GS320 System
- 12.000 Alpha 21264 CPUs
- 12 TB Hauptspeicher
- 600 TB Festplattenspeicher

Vielen Dank für Ihre Aufmerksamkeit ☺

Pallas GmbH
Hermülheimer Straße 10
D–50321 Brühl,
Germany

info@pallas.com
www.pallas.com