



# pallas

Competence in High Performance Computing

## Professionelle Entwicklungstools auf PC's und HPC Clustern

Mario Deilmann <[mario.deilmann@pallas.com](mailto:mario.deilmann@pallas.com)>

DECUS Symposium, Bonn  
8. April 2003

Pallas GmbH  
Hermülheimer Straße 10  
D-50321 Brühl, Germany

[info@pallas.com](mailto:info@pallas.com)  
[www.pallas.com](http://www.pallas.com)



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



- 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)*



- 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 !

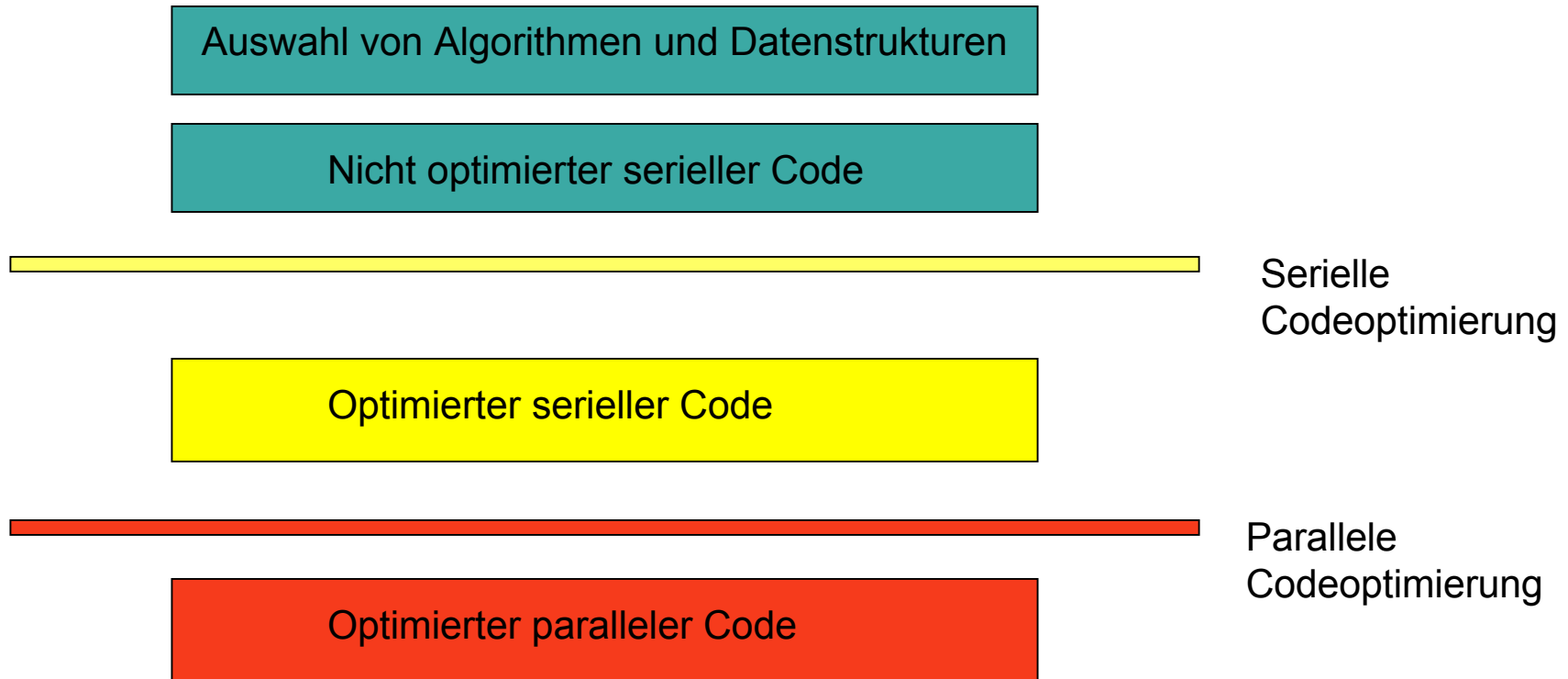


- Öl- und Gasindustrie: Seismische Analyse
- Forschung: Gewöhnliche, partielle Differentialgleichungen
- IT: *Data Mining*
- Pharmazeutischen Industrie: Wirkstoffdesign
- Meteorologie: Wettervorhersage
- Automobileindustrie: Crash Simulation
- Chemieindustrie: Simulation molekulardynamischer Prozesse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

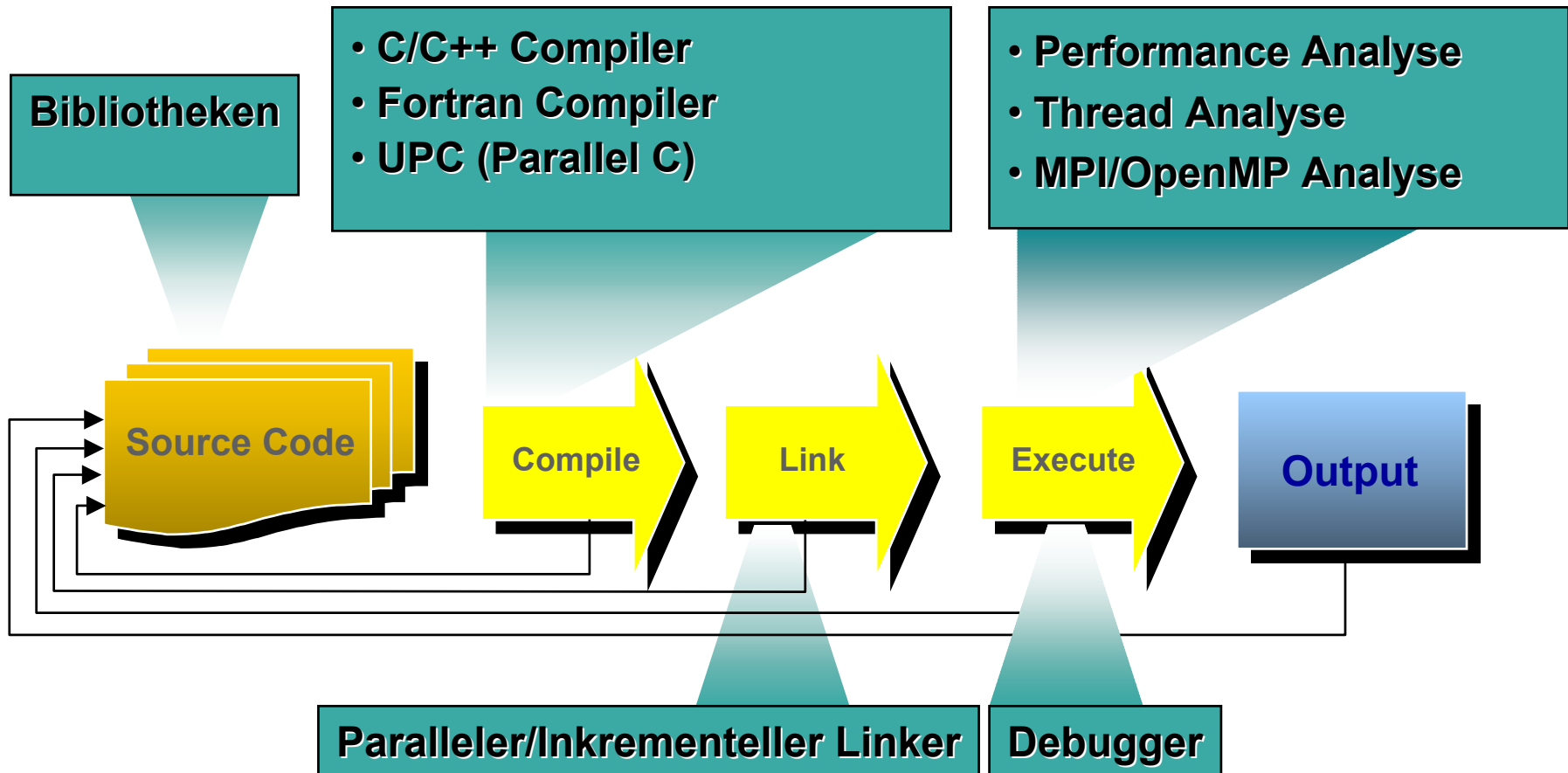
Lange Programm-Laufzeit und große Datenmengen



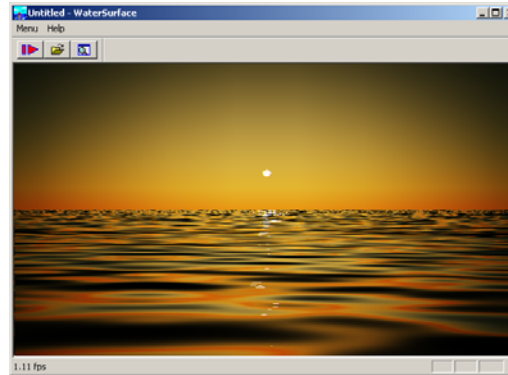
- Optimierende **Compiler** beschleunigen die Ausführungszeit
- (Parallele) **Linker** verkürzen den *Build*-Zyklus
- **Debugger** verkürzen entscheidend die Fehlersuche
- **Profiler** können ermitteln welche Zeit in welchem Programmteil verbracht wurde
- Spezielle **Bibliotheken** ermögliche den Zugriff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







# Serielle Code-Optimierung



shgravsph

shgravsph (shgravsph.F) 1000,815 seconds on 4 processors  
Thu Jun 11 12:56:41 1998 Thu Jun 11 13:30:58 1998

Line	Source	Proc	Count	Time
250	jrbox(nb)=iraw+nbref(nb)*M1	(max)	2634010	2,429
251	isjrbox(nb)	(max)	2634010	2,123
262 c	if(icbrf_eq,brf_and, icbrf_ne,0) goto 749			
263 750	if (i.eq,0) goto 749	(max)	9755300	7,697
264	ic=ic+1	(max)	7121290	5,631
265	r1(1,ic)=(1,i)	(max)	7121290	20,804
266	r1(2,ic)=(2,i)	(max)	7121290	5,582
267	r1(3,ic)=(3,i)	(max)	7121290	5,539
268	t1(ic)=itype(i)	(max)	7121290	5,743
	n1(ic)=i	(max)	7121290	5,828
	is1(i)	(max)	7121290	5,712

VTune(TM) Performance Analyzer 6.0 [Source View - [C:\vtune\demos\gzip\deflate.c]]

Project Navigator: VTProject3, gzip, Run 0, Timer

RVA	Size	Name	Function Summary	Timer (2)
0x1320	0x119	in_int		
0x1440	0x150	longest_match		
0x1500	0x2D	ll_window		
0x1620	0x20	deflate		
0x1940	0x2B	deflate_fast		

Output: Sampling Results - Thu Apr 19 16:19:40 2001, Started at: Thu Apr 19 16:19:25 2001, Finished at: Thu Apr 19 16:19:40 2001, Duration: 15 Seconds

Intel(R) Tuning Assistant

Tuning Advice for Sampling Results - Thu Apr 19 16:19:40 2001

Tuning Advice for Selected Code in C:\vtune\demos\gzip\deflate.c

- 100% Lines 675 to 703 in deflate.c
- Line 679: 16-bit to 32-bit conversion
- Line 679: 16-bit to 32-bit conversion
- Line 679: 16-bit to 32-bit conversion
- Line 688: Logical AND/OR conversion
- Line 692: Loop invariant parameters
- Line 697: Logical AND/OR conversion
- Line 707: Logical AND/OR conversion
- Line 726: Loop unrolling
- Line 730: Ignored return values
- Line 730: Loop invariant parameters
- Line 739: Ignored return values
- Line 758: Loop invariant parameters
- Line 758: Compiler vectorizer

Line 730: Loop invariant parameters

The argument list for the function call to flush\_block on line 730 in file C:\CodeExamples\gzip134\deflate.c appears to be loop-invariant. If there are no conflicts with other variables in the loop, and if the function has no side effects and no external dependencies, move the call out of the loop.



- Simulation einer bewegten Wasseroberfläche

The screenshot shows a Windows desktop environment. On the left, a window titled "Untitled - WaterSurface" displays a simulation of a moving water surface with a sunset background. A yellow circle highlights the text "1.11 fps" in the bottom-left corner of this window. On the right, the "Windows Task Manager" is open, showing the "Performance" tab. A yellow circle highlights the "CPU Usage" section, which indicates 25% usage. Below this, the "Commit Charge" is shown as 111 MB out of 2461 MB. The Task Manager also displays various system statistics such as Handles, Threads, Processes, Physical Memory, and Kernel Memory.

Totals		Physical Memory (K)	
Handles	6476	Total	1048044
Threads	389	Available	817720
Processes	25	System Cache	165420

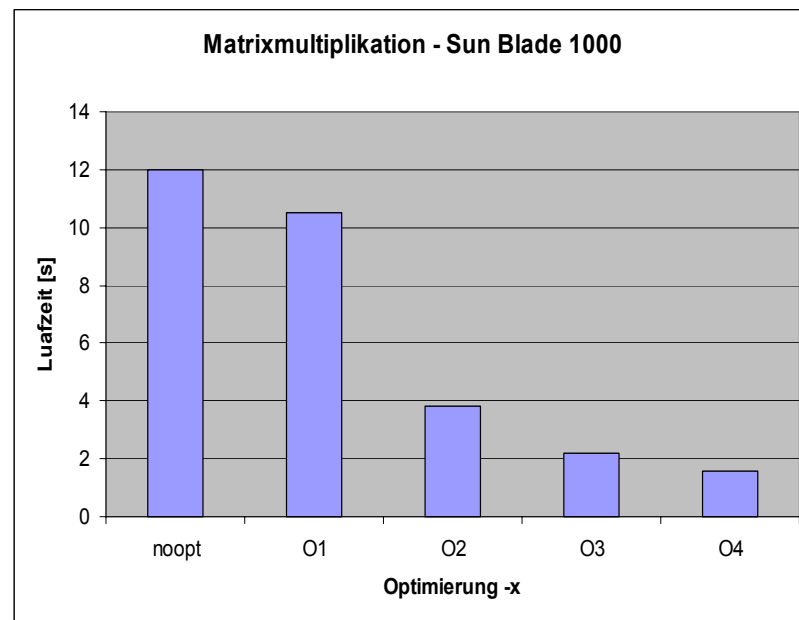
  

Commit Charge (K)		Kernel Memory (K)	
Total	113820	Total	27244
Limit	2520672	Paged	17248
Peak	125152	Nonpaged	9996

Processes: 25    CPU Usage: 25%    Commit Charge: 111M / 2461M



- 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





**XDB Simulator - [test\_cpp] - \\test.cpp**

File Display Run Debug Language Options Devices Xscale Window Help

Download Start Reload Exit Cxx\_Func

**[test\_cpp] \\test.cpp**

```
Line Source
57 // Sort an array like bubble-sort.
58
59 • for (a = 0; a < 4; a++)
60 {
61 • for (b = 0; b < 4; b++)
62 {
63 • for (c = a; c < 4; c++)
64 {
65 • if (c == a)
66 {
67 • var = b;
68 • }
69 • else
70 {
71 • var = 0;
72 • }
73 • }
74 • }
75 • }
76 • }
```

**Execution Trace [source] total: 4096 display: -438 - 0**

```
test_cpp 84 array4[c][d] = help;
test_cpp 86 }
test_cpp 75 if (array4[a][b] > array4[c][d])
test_cpp 86 }
test_cpp 75 if (array4[a][b] > array4[c][d])
test_cpp 78 help = array4[a][b];
test_cpp 81 array4[a][b] = array4[c][d];
test_cpp 84 array4[c][d] = help;
test_cpp 86 }
test_cpp 75 if (array4[a][b] > array4[c][d])
test_cpp 78 help = array4[a][b];
test_cpp 81 array4[a][b] = array4[c][d];
test_cpp 86 array4[c][d] = help;
test_cpp 87 }
test_cpp 88 }
```

**Device Registers**

**LCD Controller:**

LCCR0	0xB6B6B6B6	LCD Controller Control Register 0
LCCR1	0xB6B6B6B6	LCD Controller Control Register 1
LCCR2	0xB6B6B6B6	LCD Controller Control Register 2
LCCR3	0xB6B6B6B6	LCD Controller Control Register 3
FDADRO	0xB6B6B6B6	DMA Channel 0 Descriptor Address Register
FSADRO	0xB6B6B6B6	DMA Channel 0 Source Address Register
FIDRO	0xB6B6B6B6	DMA Channel 0 Frame ID Register
LDCMDO	0xB6B6B6B6	DMA Channel 0 Command Register
FBRO	0xB6B6B6B6	DMA Channel 0 Branch Register
FDADR1	0xB6B6B6B6	DMA Channel 1 Descriptor Address Register
FSADR1	0xB6B6B6B6	DMA Channel 1 Source Address Register
FIDR1	0xB6B6B6B6	DMA Channel 1 Frame ID Register
LDCMD1	0xB6B6B6B6	DMA Channel 1 Command Register
FBR1	0xB6B6B6B6	DMA Channel 1 Branch Register
LCSR	0xB6B6	
LIIDR	0xB6B6	
TRGBR	0xB6B6	
TCR	0xB6B6	

**Full Function UART:**

FFUART_RBR	<in u
FFUART_THR	<writ
FFUART_IER	<in u
FFUART_IIR	0xB6B6
FFUART_FCR	<writ
FFUART_LCR	0xB6B6
FFUART_MCR	0xB6B6
FFUART_LSR	<read
FFUART_MSR	0xB6B6
FFUART_SFR	0xB6B6
FFUART_DLL	0xB6B6
FFUART_DLH	0xB6B6
FFUART_ISR	0xB6B6

**Bluetooth UART:**

BTUART_RBR	<in u
BTUART_THR	<writ

**Modify Register LCCR0**

Original Value: 0xB6B6B6B6

Register Value: 0xB6B6B6B4

CMS: 0x0 Bits: 1-1

Register Layout:

0 = Color operations enabled

Description:  
LCD Controller Control Register 0 (LCCR0) (0x4400 0000)

The user must program the control bit within all other control registers before setting ENB = 1 (a word write can be used to configure LCCR0 while setting ENB after all other control registers have been programmed), and also must disable the LCD controller when changing the state of any control bit within the LCD controller. Note that writes to reserved bits are ignored and reads return zeros.

**Command**

```
xdb> RUN
program stopped: BREAKPOINT ID=0 at "test_cpp|arrayTest(void)@LINE 63" [task=GEN-TI (0)]
xdb>
```

For Help, Press F1 [00] test\_cpp|arrayTest(void)@LINE 63 [GEN-TI] PC = 0xA000A9C8 C/C++





Routine

Anzahl  
der  
Aufrufe

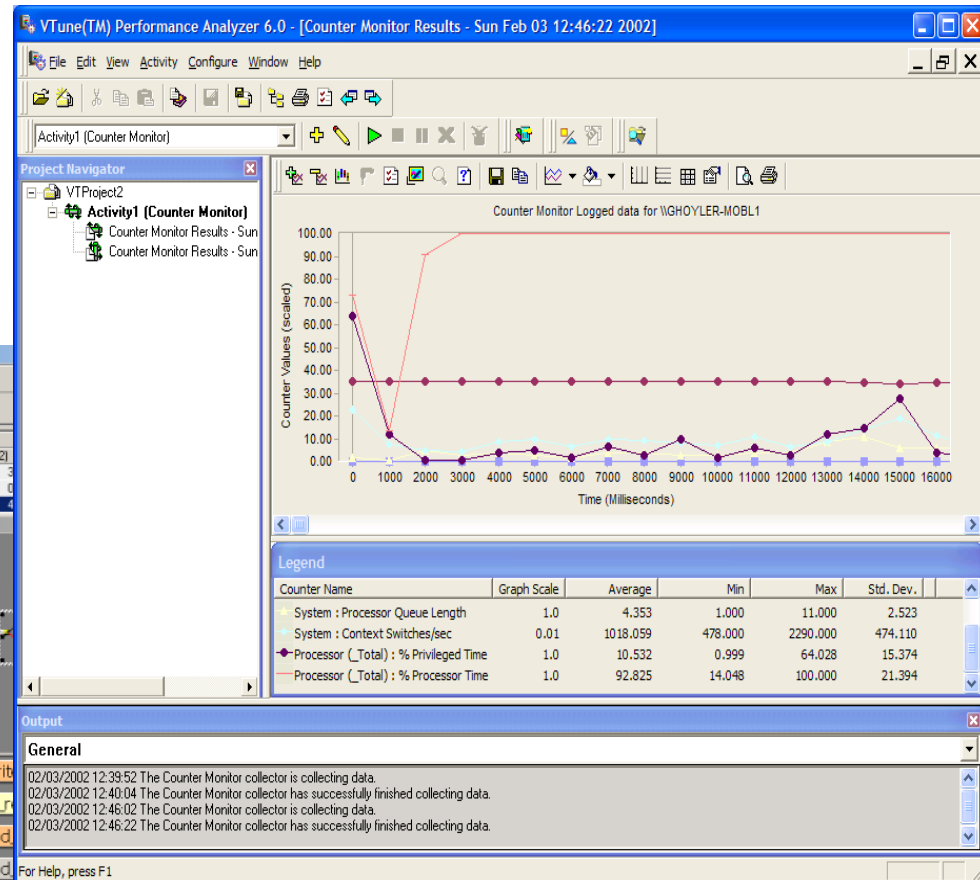
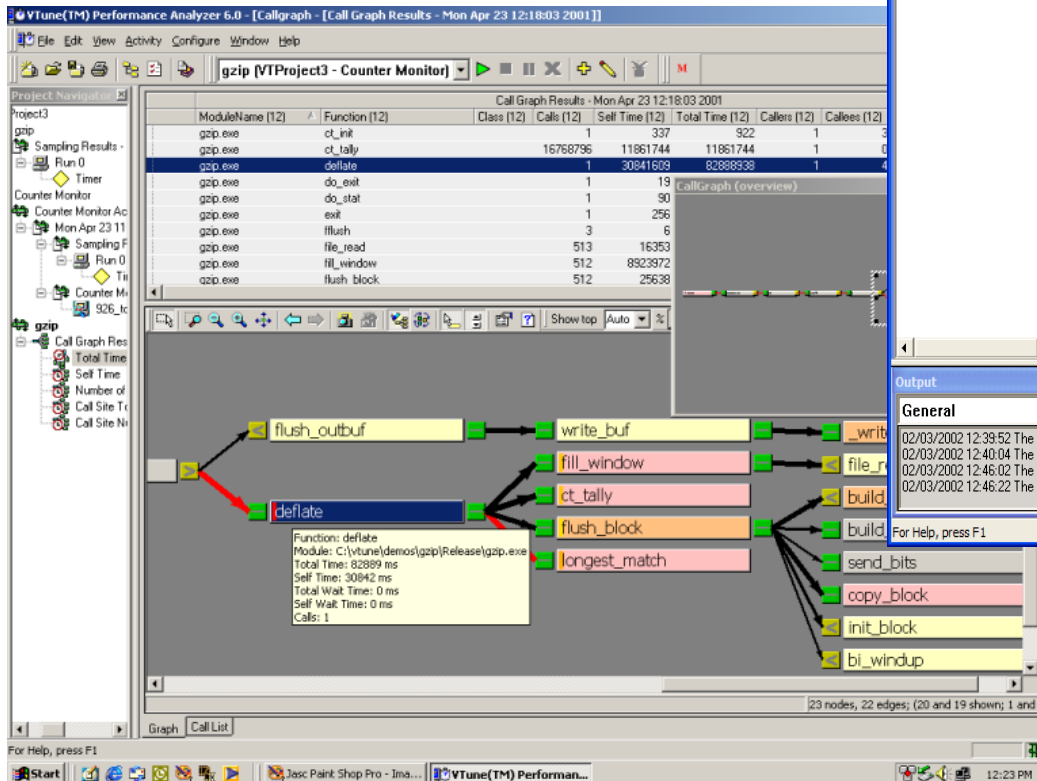
Line	Source	Proc	Count	Time
250	jnbox(nb)=iraw-nbref(nb)*N1	(max) 2634010	2,429	
251	i=jnbox(nb)	(max) 2634010	2,123	
252 c	if(icbrf,eq,nbrf ,and, icbrf,ne,0) goto 749			
253 750	if (i,eq,0) goto 749	(max) 9755300	7,897	
254	ic=ic+1	(max) 7121290	5,691	
255	r1(1,ic)=r(1,i)	(max) 7121290	20,804	
256	r1(2,ic)=r(2,i)	(max) 7121290	5,582	
257	r1(3,ic)=r(3,i)	(max) 7121290	5,539	
258	itl(ic)=itype/	(max) 7121290	5,743	
259	nl(ic)=i	(max) 7121290	5,828	
260	i=i+1	(max) 7121290	5,712	

Quellcode-  
informationen

Laufzeit



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



# Profiler - Tuning Assistant



The screenshot displays the VTune Performance Analyzer 6.0 interface. The main window shows the source code for a deflate.c file. The code includes a while loop and an if statement. The Intel(R) Tuning Assistant window is open, providing advice for the selected range of lines (675 to 703). The advice includes several suggestions, such as 16-bit to 32-bit conversion and logical AND/OR conversion. A specific hint for Line 730 is highlighted, suggesting the use of loop-invariant parameters for a function call.

RVA	Size	Name	Timer (2)
0x1320	0x119	in_init	
0x1440	0x150	longest_match	
0x1590	0xFD	fill_window	
0x1690	0x305	deflate	
0x1940	0x23B	deflate_fast	

Function Summary

RVA	Size	Name	Timer (2)
--- Selected Range ---			
0x1320	0x119	in_init	
0x1440	0x150	longest_match	
0x1590	0xFD	fill_window	
0x1690	0x305	deflate	
0x1940	0x23B	deflate_fast	

Timer (2)

Sampling Results - Thu Apr 19 16:19:40 2001  
Started at: Thu Apr 19 16:19:25 2001  
Finished at: Thu Apr 19 16:19:40 2001  
Duration: 15 Seconds

Intel(R) Tuning Assistant

Tuning Advice for Selected Code in C:\vtune\demos\gzip\deflate.c

- 100% Lines 675 to 703 in deflate.c
  - Line 679: 16-bit to 32-bit conversion
  - Line 679: 16-bit to 32-bit conversion
  - Line 679: 16-bit to 32-bit conversion
  - Line 686: Logical AND/OR conversion
  - Line 692: Loop invariant parameters
  - Line 697: Logical AND/OR conversion
  - Line 707: Logical AND/OR conversion
  - Line 726: Loop unrolling
  - Line 730: Ignored return values
  - Line 730: Loop invariant parameters
  - Line 739: Ignored return values
  - Line 758: Loop invariant parameters
  - Line 758: Compiler vectorizer

Intel(R) Tuning Assistant : More Information

**Line 730: Loop invariant parameters**

The argument list for the function call to `_flush_block` on line 730 in file `C:\CodeExamples\gzip124\deflate.c` appears to be loop-invariant. If there are no conflicts with other variables in the loop, and if the function has no side effects and no external dependencies, move the call out of the loop.

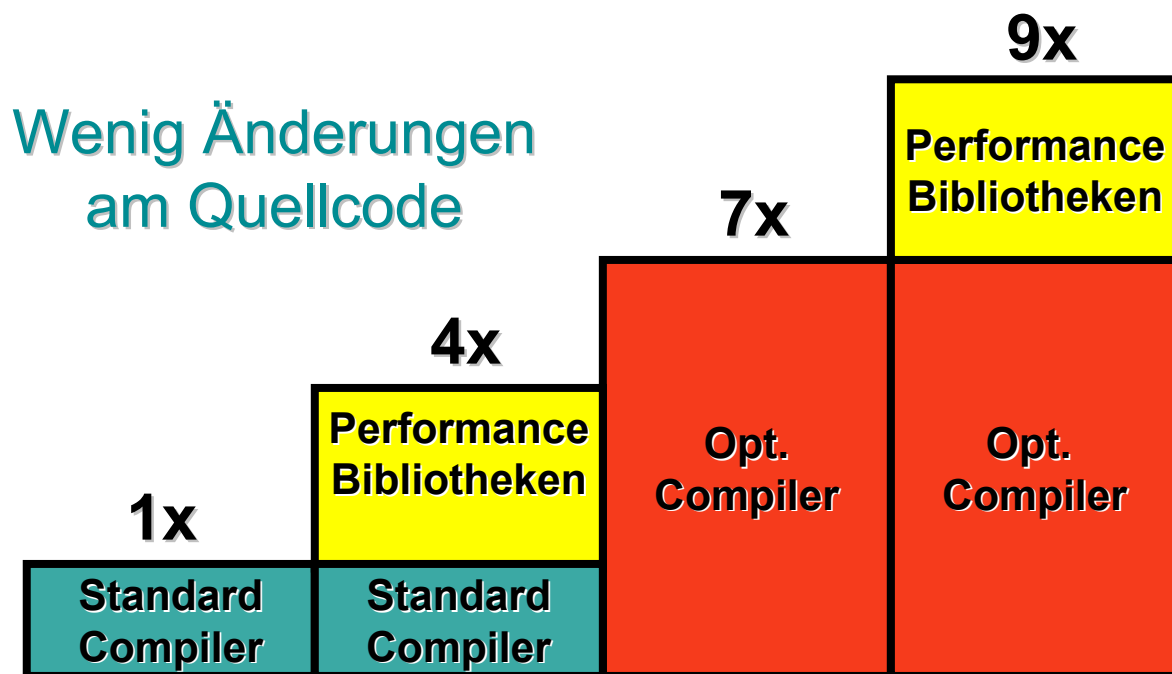
Quellcode

Hinweis

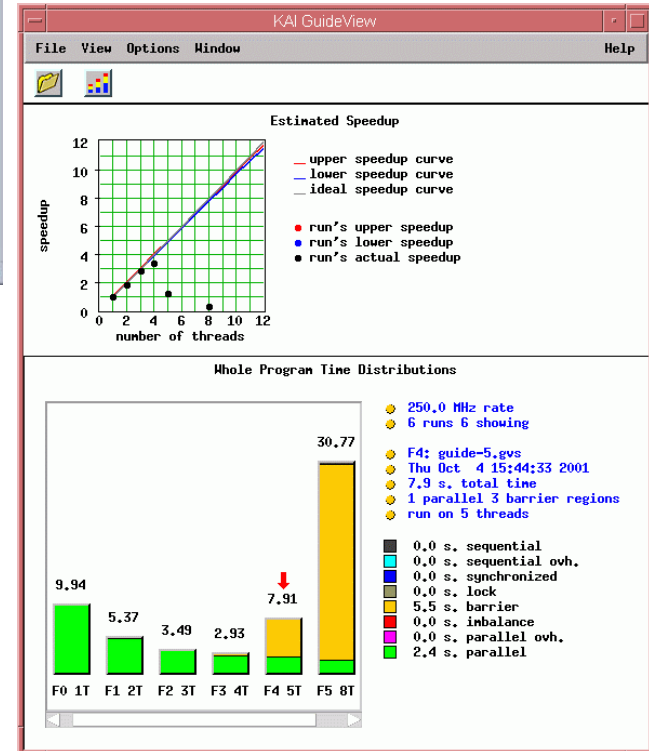
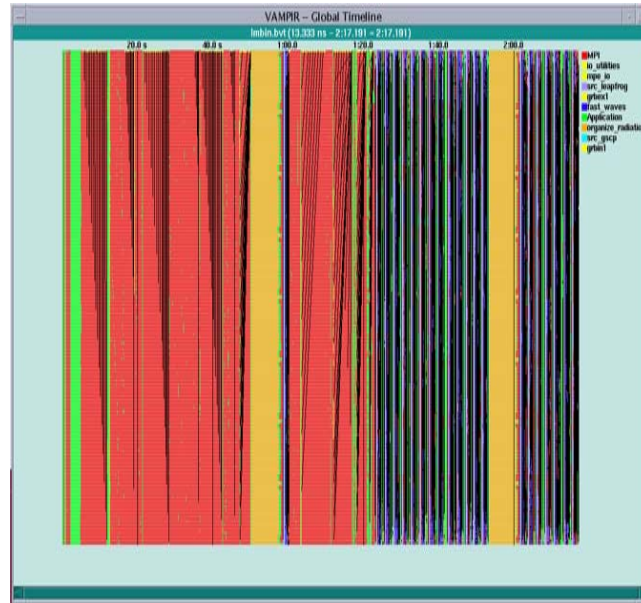




**9x schneller**



# Parallele Code-Optimierung





- Multithreaded OpenMP parallel level

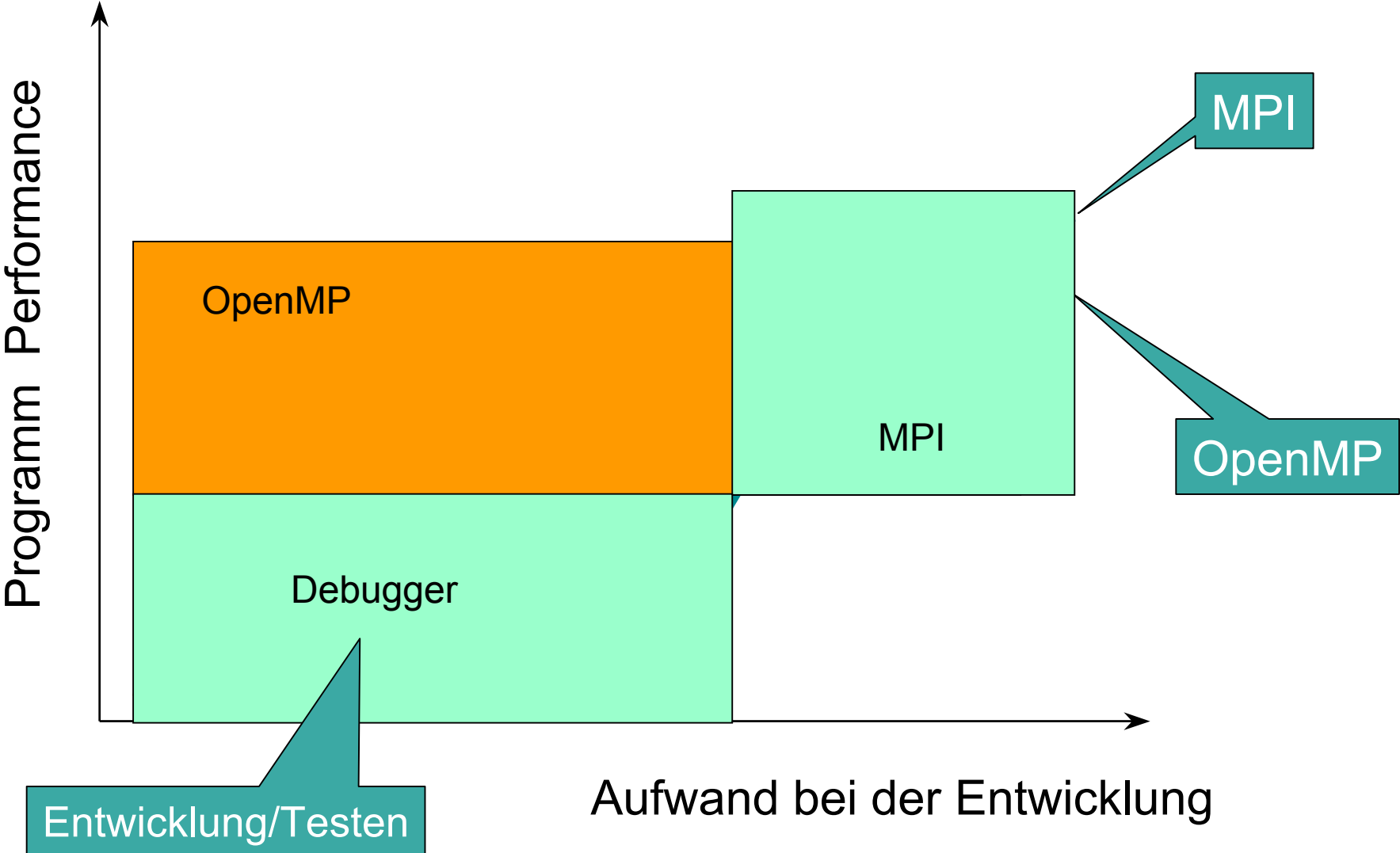
```
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
print *, "Process ", myid, " of ", numprocs, " is alive"
c    send b to each other process
do 25 i = 1,bcols
  call MPI_BCAST(b(1,i), brows, MPI_DOUBLE_PRECISION, master,
 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

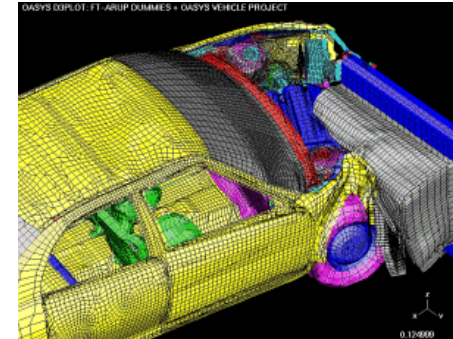
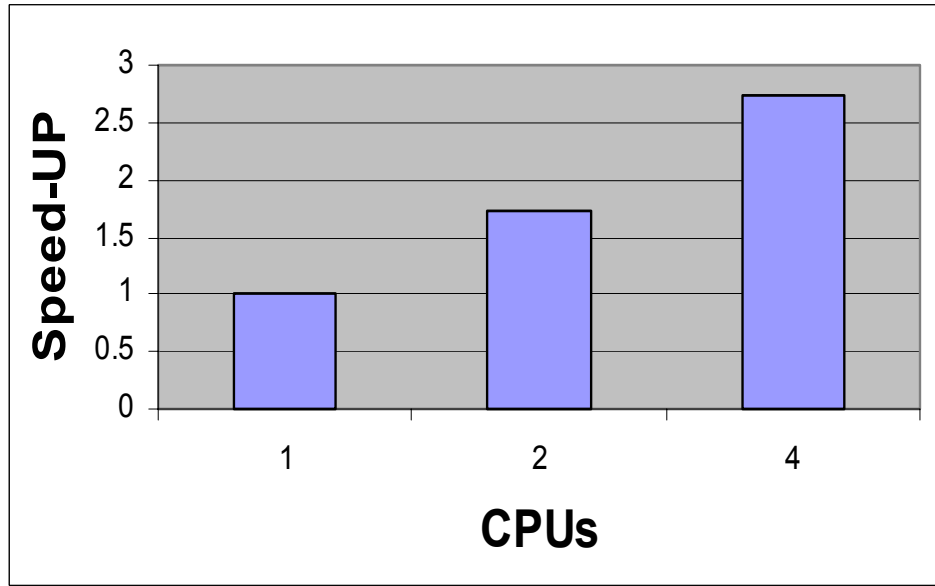
c    send a row of a to each other process; tag with row number
do 40 i = 1,numprocs-1
  do 30 j = 1,acols
    buffer(j) = a(i,j)
30  continue
  call MPI_SEND(buffer, acols, MPI_DOUBLE_PRECISION, i,
    & i, MPI_COMM_WORLD, ierr)
  numsent = numsent+1
```

ding

ninen

# Gegenüberstellung: Performance - Aufwand



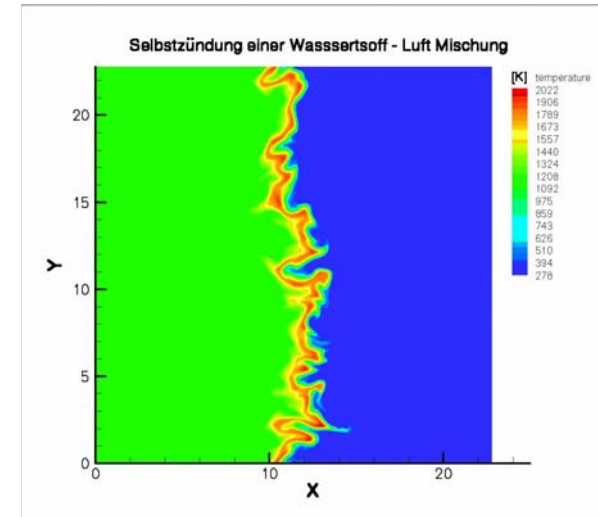
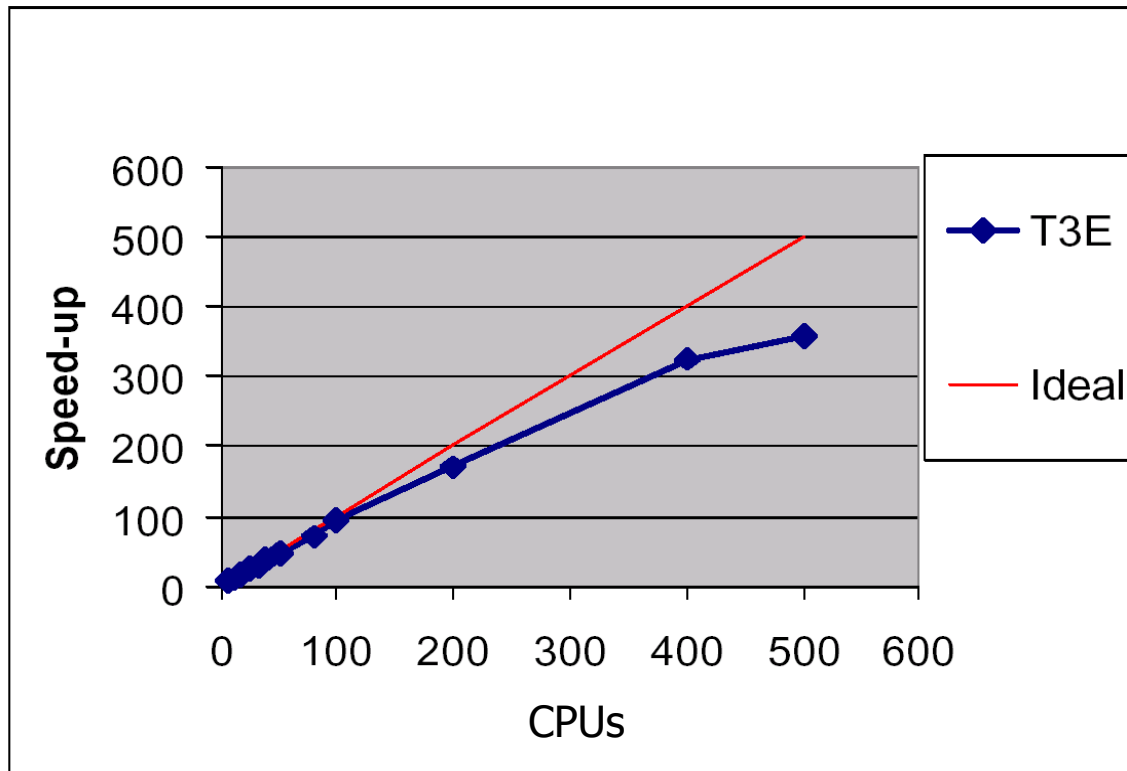


3D: 111.029 Elemente  
1 CPU Zeit = 119955 Sekunden  
2 CPU Zeit = 69113 Sekunden  
4 CPU Zeit = 43864 Sekunden  
Quad Xeon

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



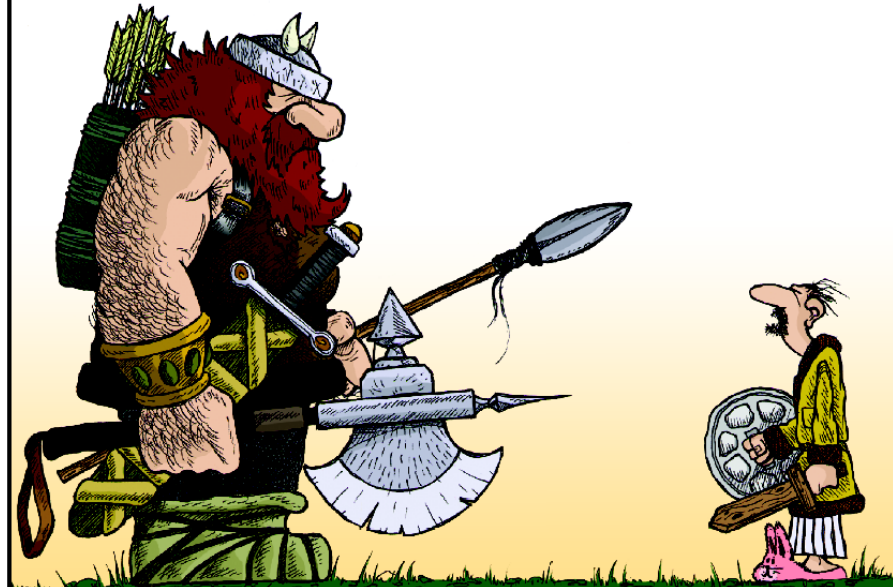
**1 Mio. Elemente**  
**2D: 1000 x 1000**  
**Cray T3E**  
**1CPU – 100 Stunden**  
**100CPUs – 1,5 Stunden**

NSCORE – Programm zur Direkten numerischen Simulation von Verbrennungsvorgängen

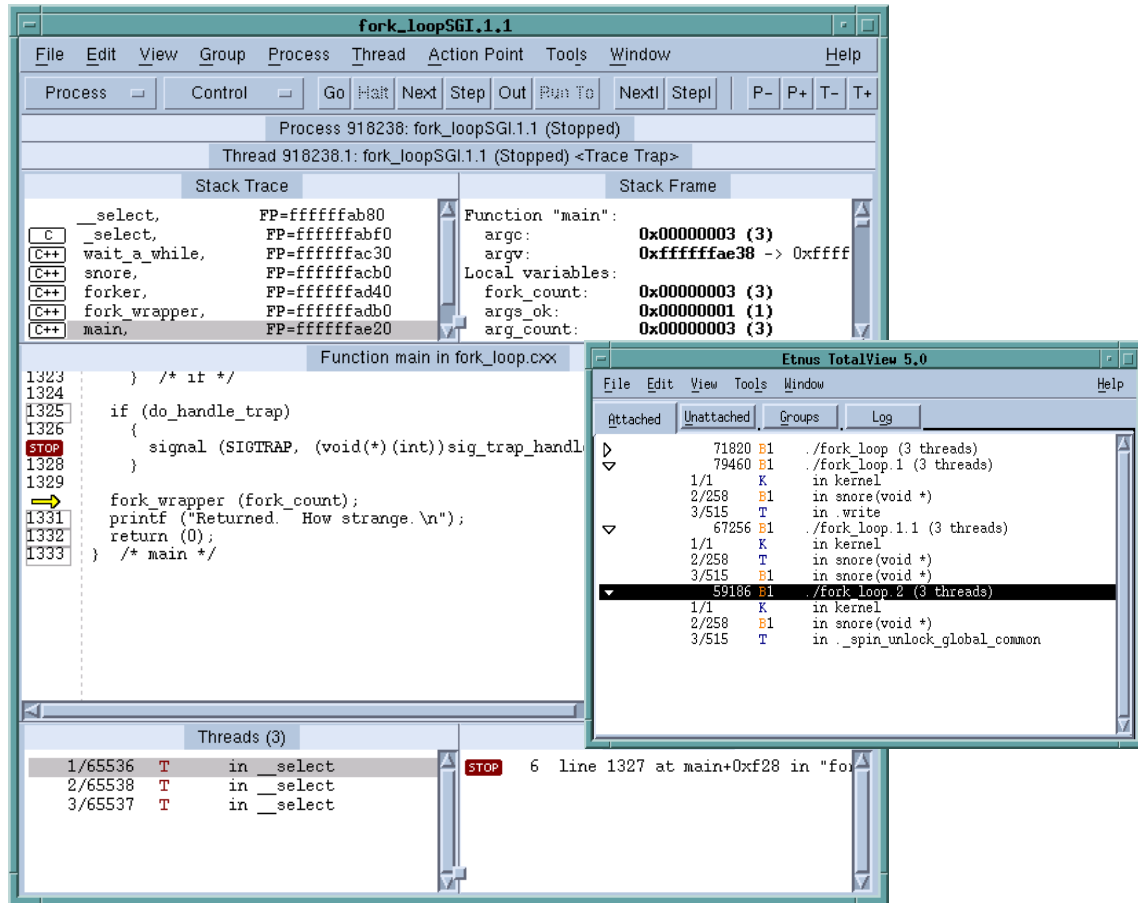
**350x** speed up auf 512 CPUs

# Parallele Programmentwicklung

AS DAWN BROKE,  
CODY REALIZED HE  
WAS ILL-EQUIPPED...



# Paralleler Debugger



The screenshot displays the Pallas parallel debugger interface for a process named `fork_loopSGL1.1`. The process is stopped at a trace trap. The main window shows the following components:

- Process Information:** Process 918238: `fork_loopSGL1.1` (Stopped). Thread 918238.1: `fork_loopSGL1.1` (Stopped) <Trace Trap>.
- Stack Trace:**

Function	FP
<code>__select</code>	<code>FP=ffffffab80</code>
<code>_select</code>	<code>FP=ffffffabf0</code>
<code>wait_a_while</code>	<code>FP=ffffffac30</code>
<code>snore</code>	<code>FP=ffffffacb0</code>
<code>forker</code>	<code>FP=ffffffad40</code>
<code>fork_wrapper</code>	<code>FP=ffffffadb0</code>
<code>main</code>	<code>FP=ffffffae20</code>
- Stack Frame:**

Function "main":  
`argc: 0x00000003 (3)`  
`argv: 0xfffffae38 -> 0xffff`  
 Local variables:  
`fork_count: 0x00000003 (3)`  
`args_ok: 0x00000001 (1)`  
`arg_count: 0x00000003 (3)`
- Source Code:**

```

1323     } /* it */
1324
1325     if (do_handle_trap)
1326     {
1327         signal (SIGTRAP, (void(*) (int)) sig_trap_handl
1328     }
1329
1330     fork_wrapper (fork_count);
1331     printf ("Returned. How strange. \n");
1332     return (0);
1333 } /* main */
    
```
- Threads (3):**

ID	State	Location
1/65536	T	in <code>_select</code>
2/65538	T	in <code>_select</code>
3/65537	T	in <code>_select</code>
- Etnus TotalView 5.0:**

ID	State	Location
71820	B1	./fork_loop (3 threads)
79460	B1	./fork_loop.1 (3 threads)
1/1	K	in kernel
2/258	B1	in <code>snore(void *)</code>
3/515	T	in <code>write</code>
67256	B1	./fork_loop.1.1 (3 threads)
1/1	K	in kernel
2/258	T	in <code>snore(void *)</code>
3/515	B1	in <code>snore(void *)</code>
59186	B1	./Fork loop 2 (3 threads)
1/1	K	in kernel
2/258	B1	in <code>snore(void *)</code>
3/515	T	in <code>._spin_unlock_global_common</code>

The main window also shows a **Threads (3)** panel at the bottom, which is currently empty. A **STOP** button is visible next to the thread information.





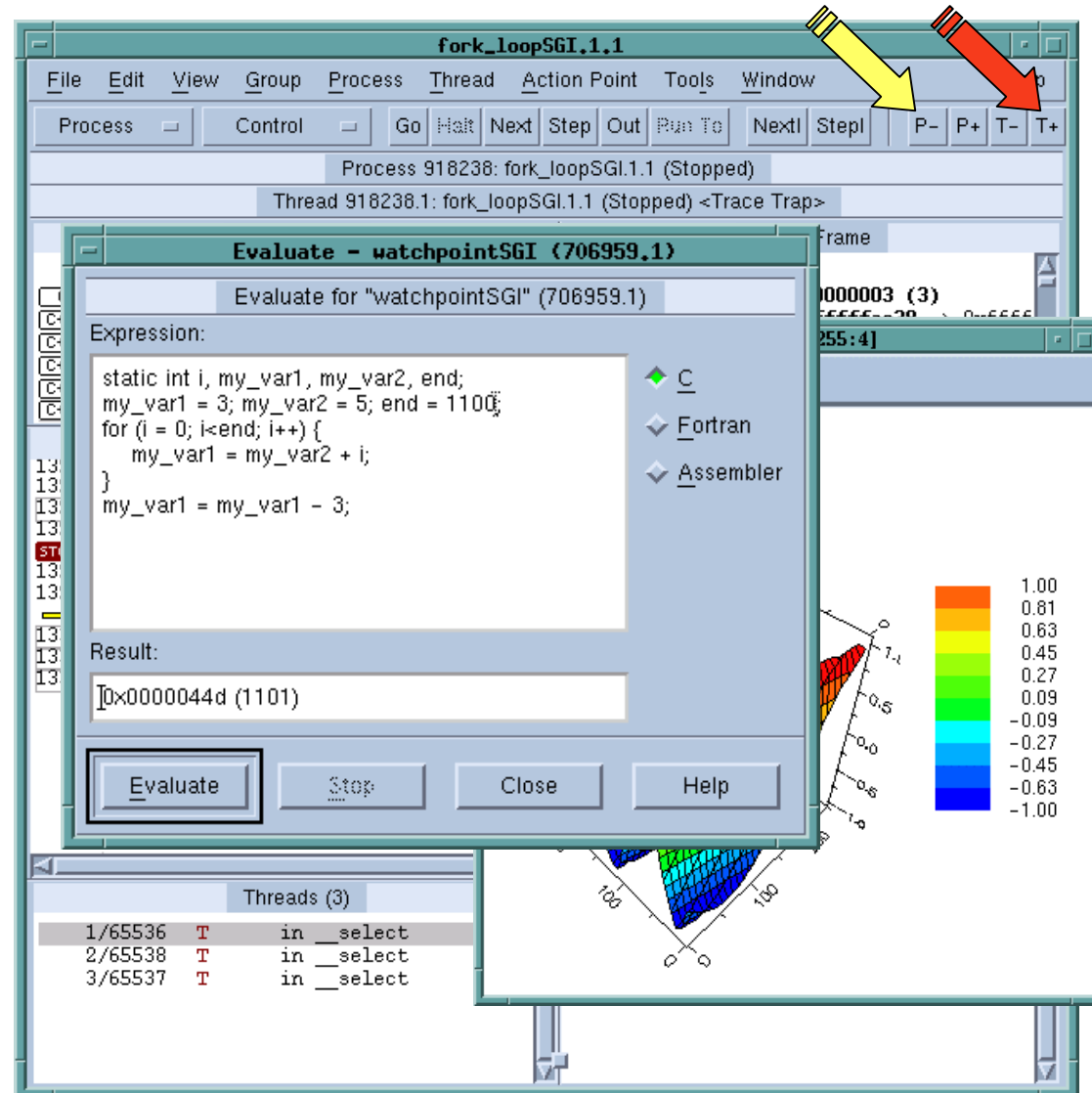
## Eigenschaften eines seriellen *command line* Debuggers plus

- 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

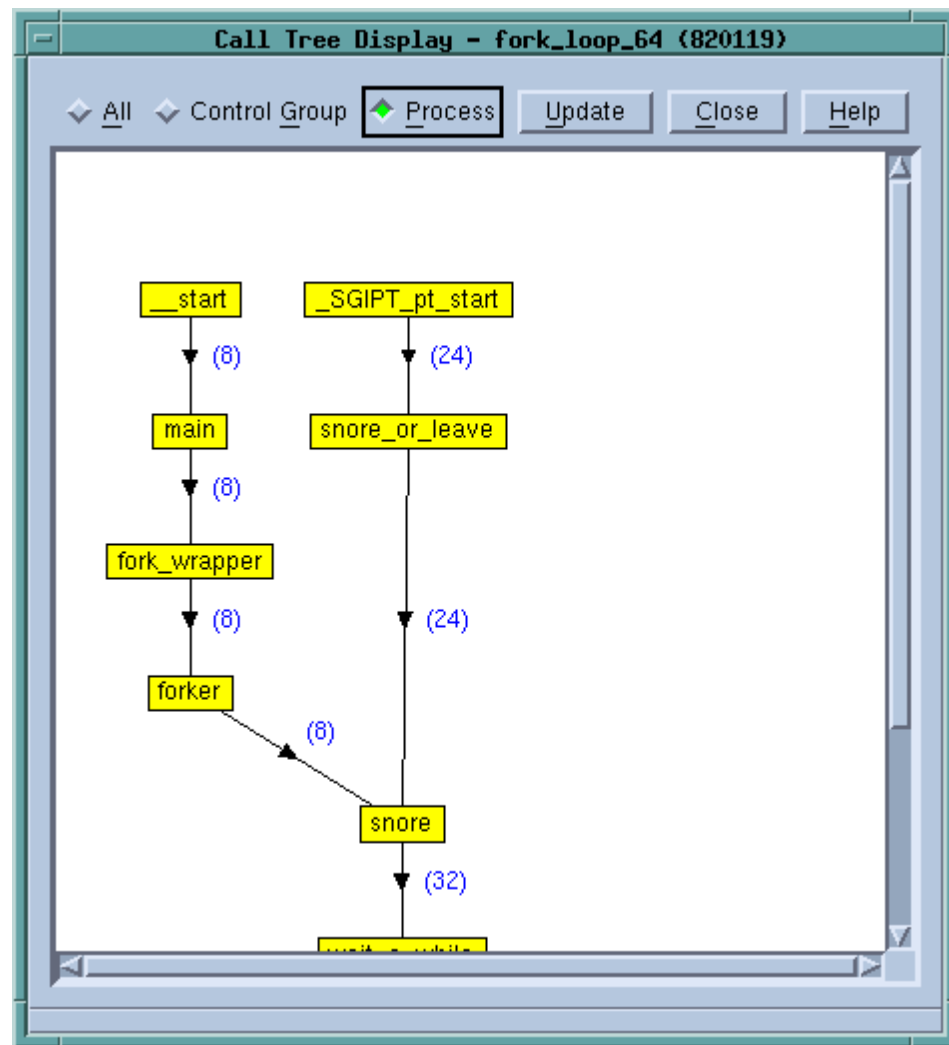


- Debug-Aktionen
  - Breakpoint
  - Evaluationpoint
  - Watchpoint
- Daten Management
  - Darstellung
  - Statistiken
  - Visualisierung
- Änderungen während des Programmlaufs



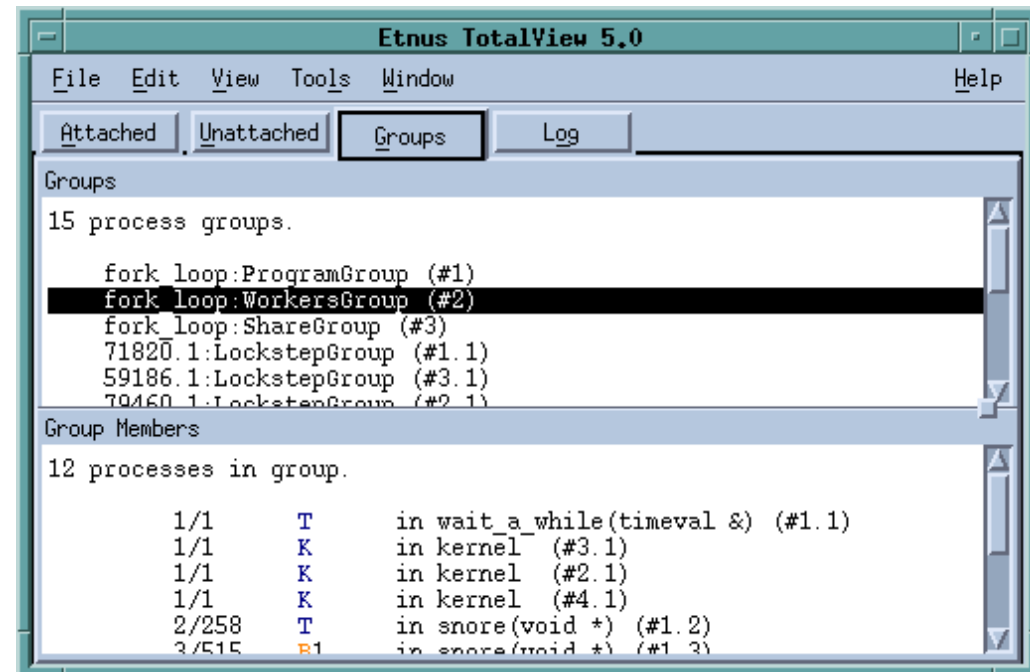


- Dynamisch Darstellung des Programmablaufs
- Potentielle Konflikte können visualisiert werden
- Kontrolle und Abhängigkeiten des Programmablaufs





- 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
  - ...





- Darstellung der Nachrichtenwege zu einem definierten Zeitpunkt
- Inter-Prozess Kommunikation
- Visualisierung von Kommunikations-Problemen (hängende, unerwartet Nachrichten)

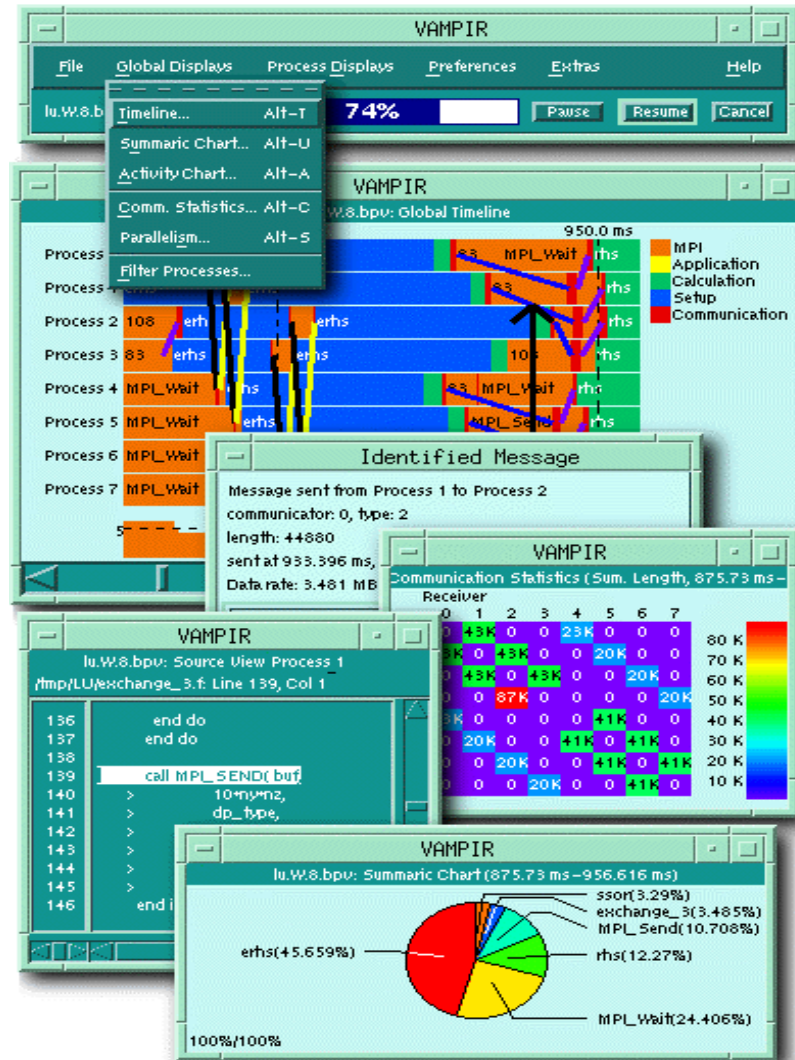
The image shows two windows from an MPI Visualizer. The top window, titled "Message Queue Graph Display - mismatch2 (61504.1)", displays a grid of message states with indices 0 through 7 highlighted in yellow. The bottom window, titled "mismatch.0", shows the "Message State for 'mismatch.0' (14350.1)".

```
MPI_COMM_WORLD
Comm_size          2
Comm_rank          0
Pending receives
[0]
  Status           Pending
  Source           1 (mismatch.1)
  Tag              100 (0x00000064)
  User Buffer       0x2ff21a90 -> 0xc2c80000 (-1027080192)
  Buffer Length     400 (0x00000190)

Unexpected messages : none
Pending sends      : none

MPI_COMM_WORLD_collective
Comm_size          2
Comm_rank          0
Pending receives   : none
Unexpected messages : none
Pending sends      : none
```

# MPI Performance Analyse



The screenshot displays the VAMPIR interface with several overlapping windows:

- Global Timeline:** Shows a horizontal bar chart for processes 1 through 7. The legend indicates: MPI (yellow), Application (orange), Calculation (green), Setup (blue), and Communication (red). A progress indicator shows 74% completion.
- Identified Message:** Displays details for a message sent from Process 1 to Process 2:
  - communicator: 0, type: 2
  - length: 44880
  - sent at 933.396 ms,
  - Data rate: 3.481 MB
- Communication Statistics:** A heatmap showing data exchange between processes 0-7. The color scale ranges from 10K (blue) to 80K (red).
 

Receiver	0	1	2	3	4	5	6	7
0	0	43K	0	0	23K	0	0	0
1	0	0	43K	0	0	0	20K	0
2	0	0	0	87K	0	0	0	20K
3	0	0	0	0	0	0	41K	0
4	0	0	0	0	0	41K	0	0
5	0	0	0	0	0	0	41K	0
6	0	0	0	0	0	0	0	41K
7	0	0	0	0	0	0	0	0
- Source View:** Shows code for process 1:
 

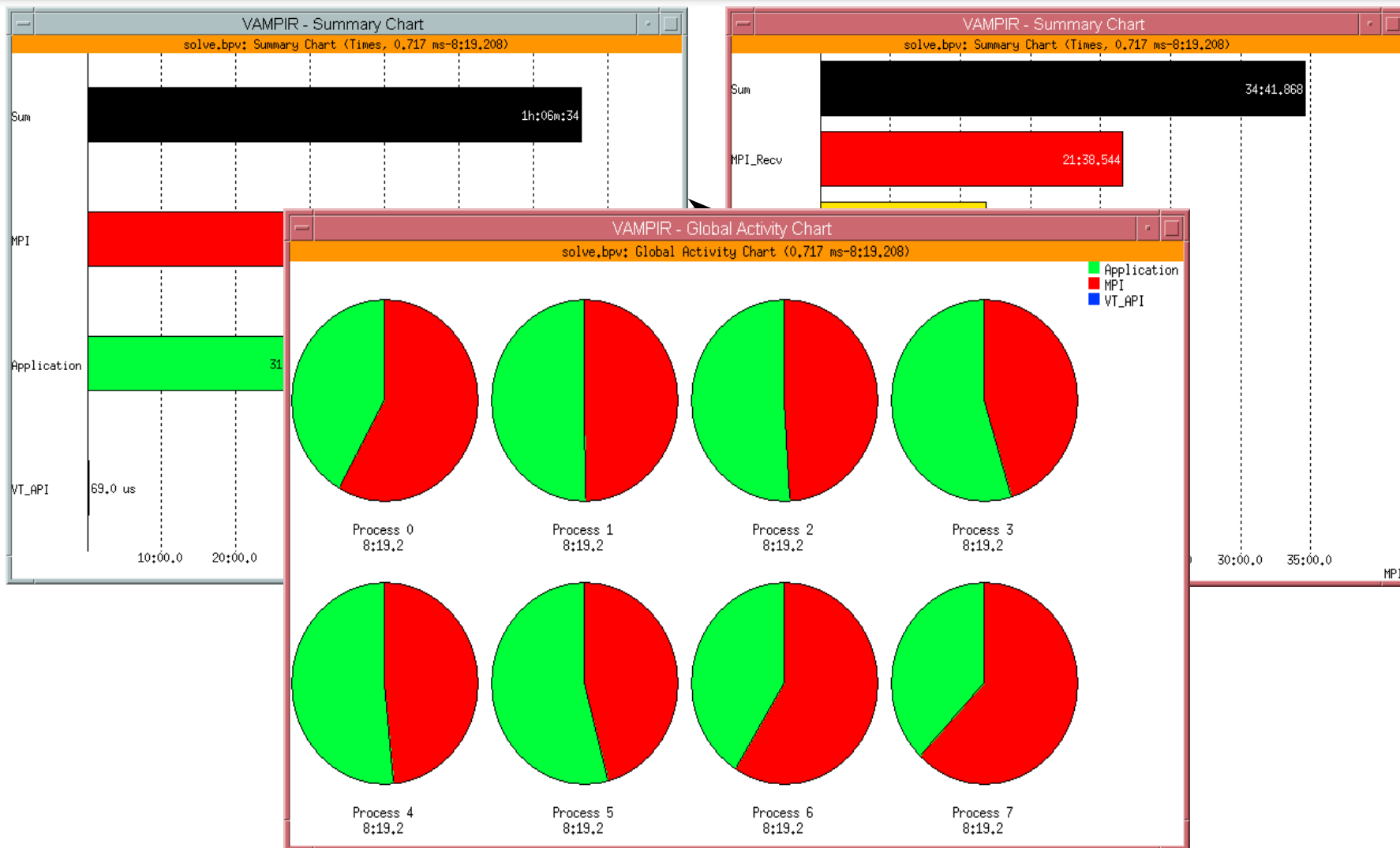
```

136   end do
137   end do
138
139   call MPL_SEND(buf
140   >   10*nmz,
141   >   dp_type,
142   >
143   >
144   >
145   >
146   end i
      
```
- Summaric Chart:** A pie chart showing the distribution of time spent on different activities:
  - erhs (45.659%)
  - MPL\_Wait (24.406%)
  - rhs (12.27%)
  - MPL\_Send (10.708%)
  - exchange\_3 (3.485%)
  - ssort (3.29%)

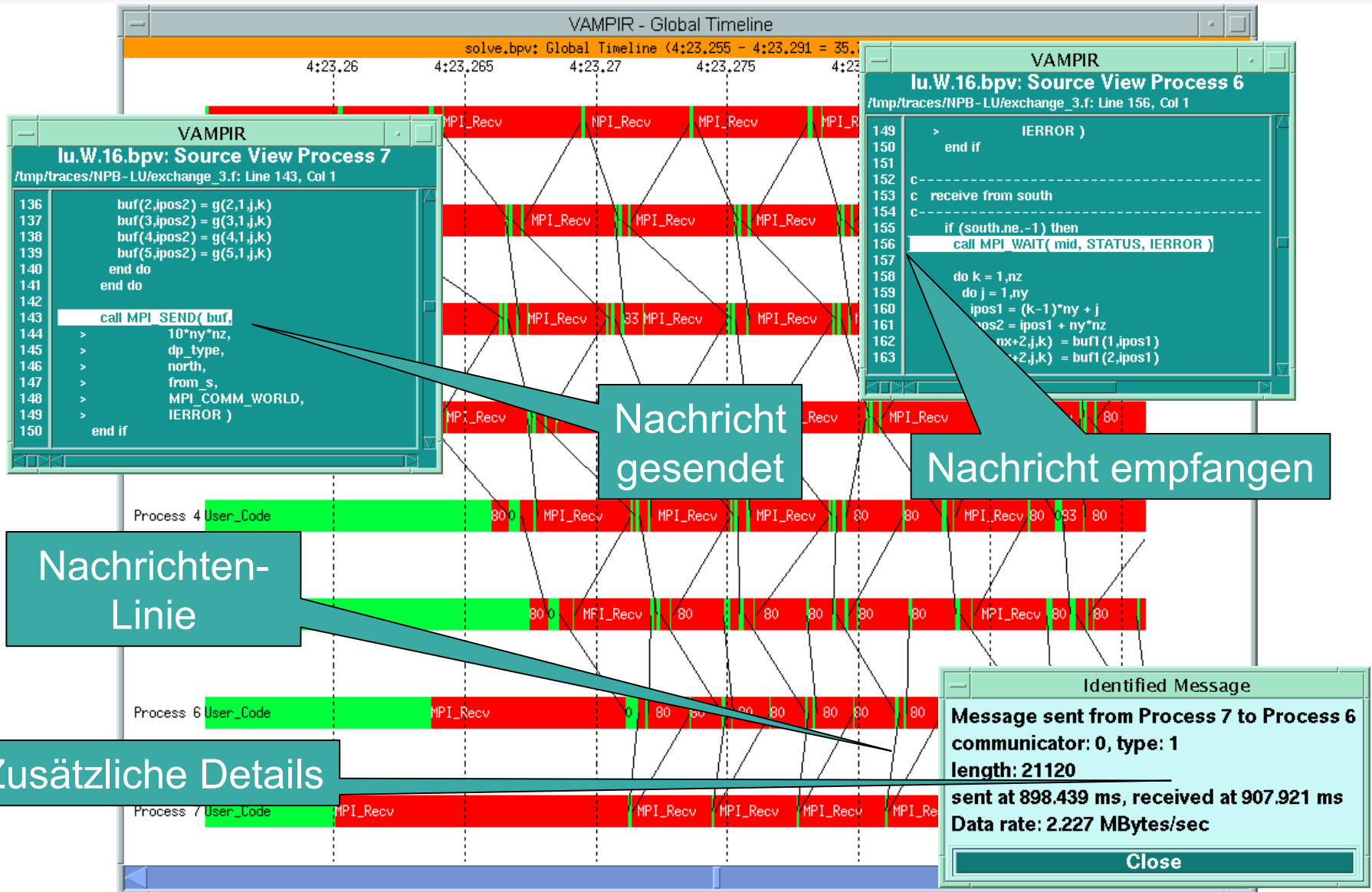


- Analyse von MPI und User Routinen des **optimierten** Programms
- Statistische Analyse von Programm und Kommunikation
- Darstellung der Lastverteilung
- Einfache und schnelle Handhabung
- Filtermöglichkeiten (Prozesse, Nachrichten, kollektive and MPI-I/O Operationen)
- Geringen Einfluss auf die Applikations-Performance

# Statistik des Programmablaufs







Nachricht  
gesendet

Nachricht empfangen

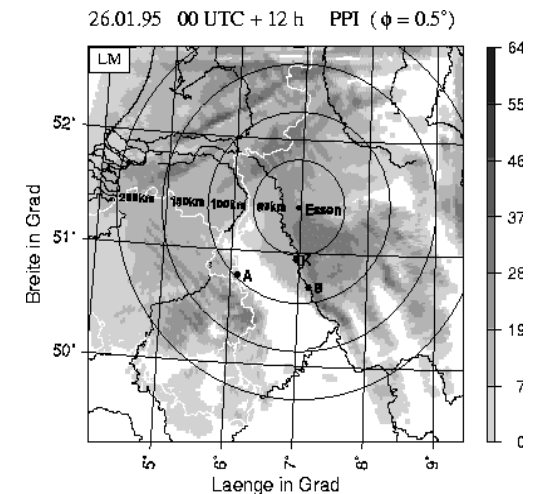
Nachrichten-  
Linie

Zusätzliche Details

Identified Message  
**Message sent from Process 7 to Process 6**  
**communicator: 0, type: 1**  
**length: 21120**  
**sent at 898.439 ms, received at 907.921 ms**  
**Data rate: 2.227 MBytes/sec**  
 Close

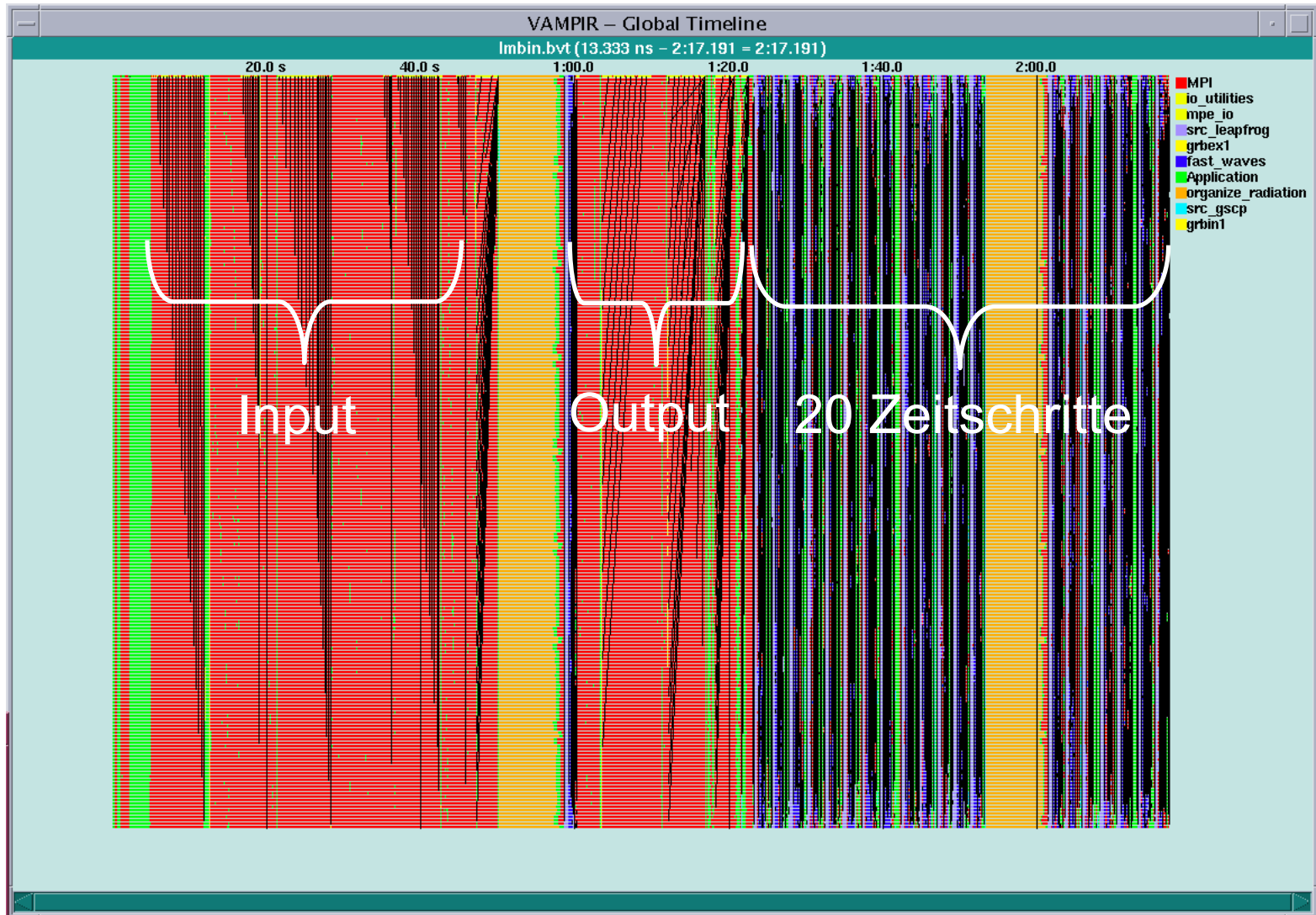


LM (Lokal Modell) ist das Programm des Deutschen Wetterdienstes (DWD) für die Wettervorhersage im Skalenbereich von 1-10 km.



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

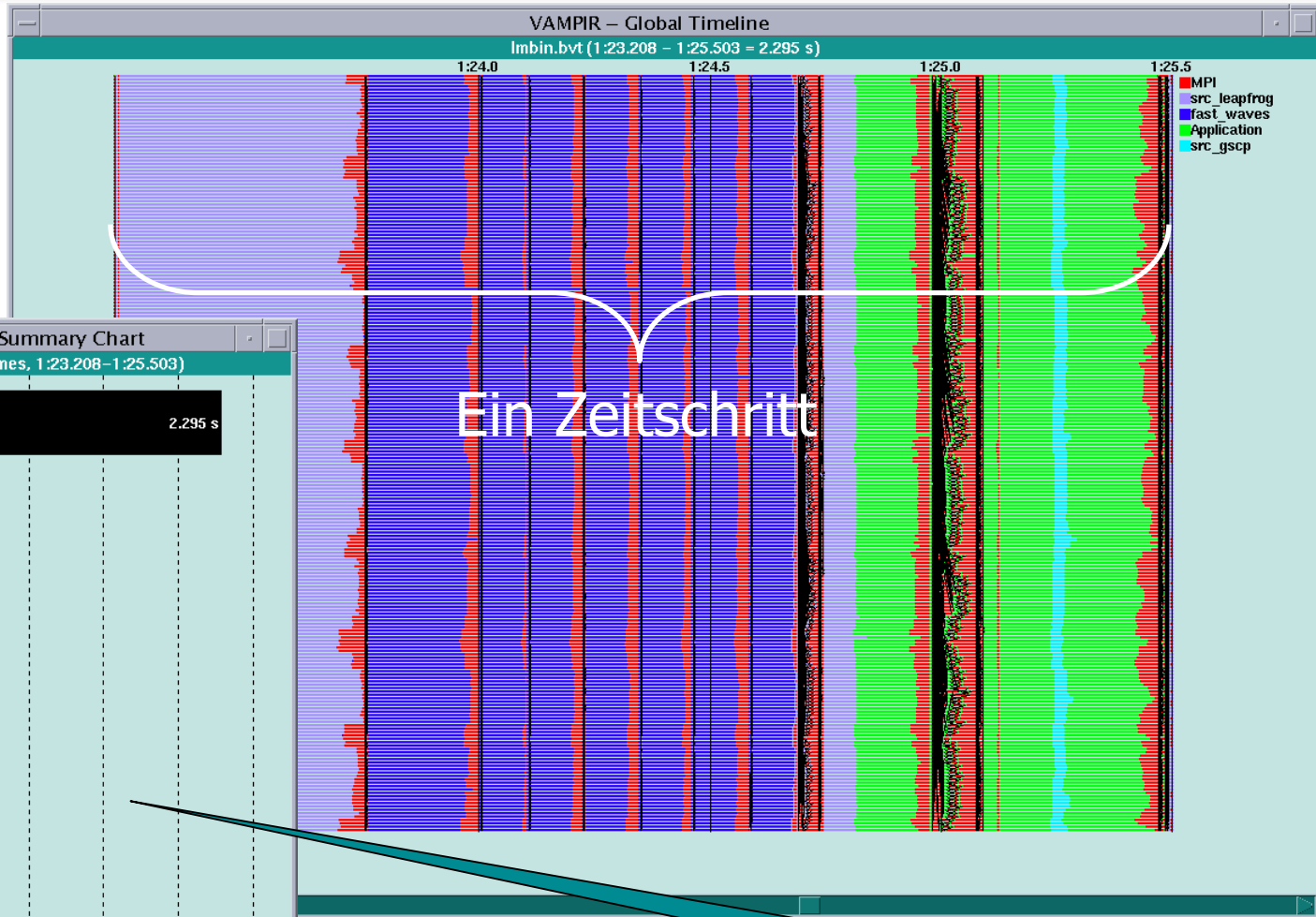
# Trace LM - Test Fall auf 224 Prozessoren



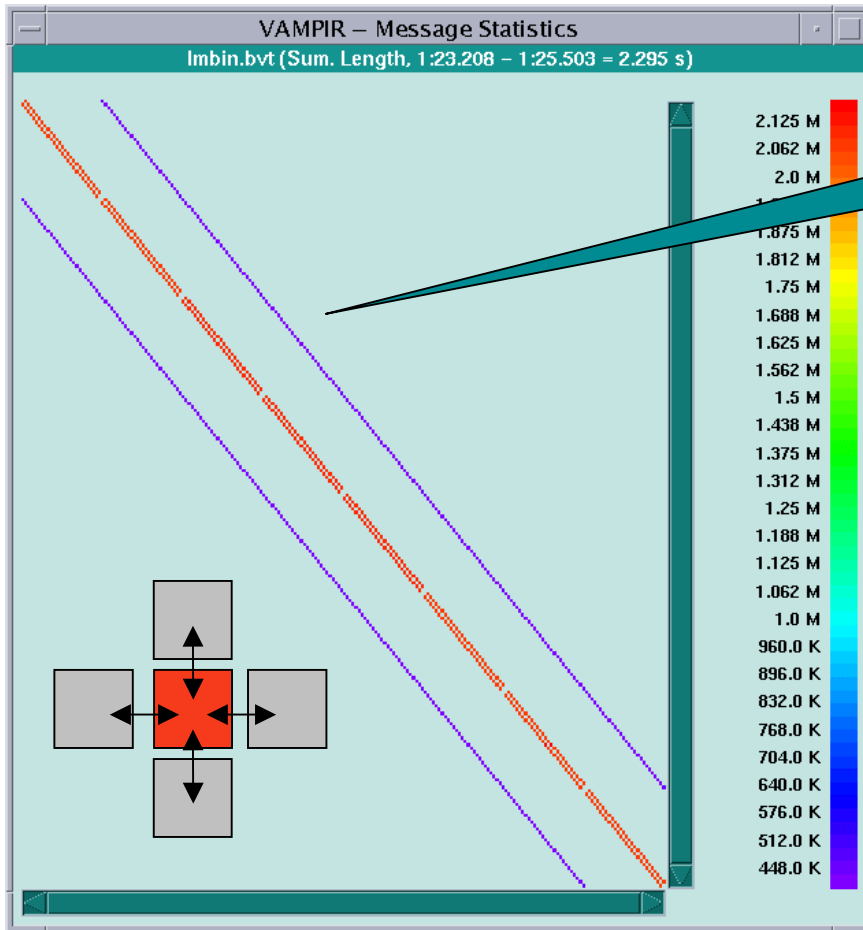


- Input
  - Dauer beim Start von LM : 45 s
- Zeitschritte
  - Dauer ca. 2.3 s
  - Auswertung der Kommunikation Performance
  - Datenverteilung: Prozess→Knoten Aufteilung
- Output
  - Dauer ca. 20 s beim Start
  - Ca. 20 s nach jedem Zeitschritt

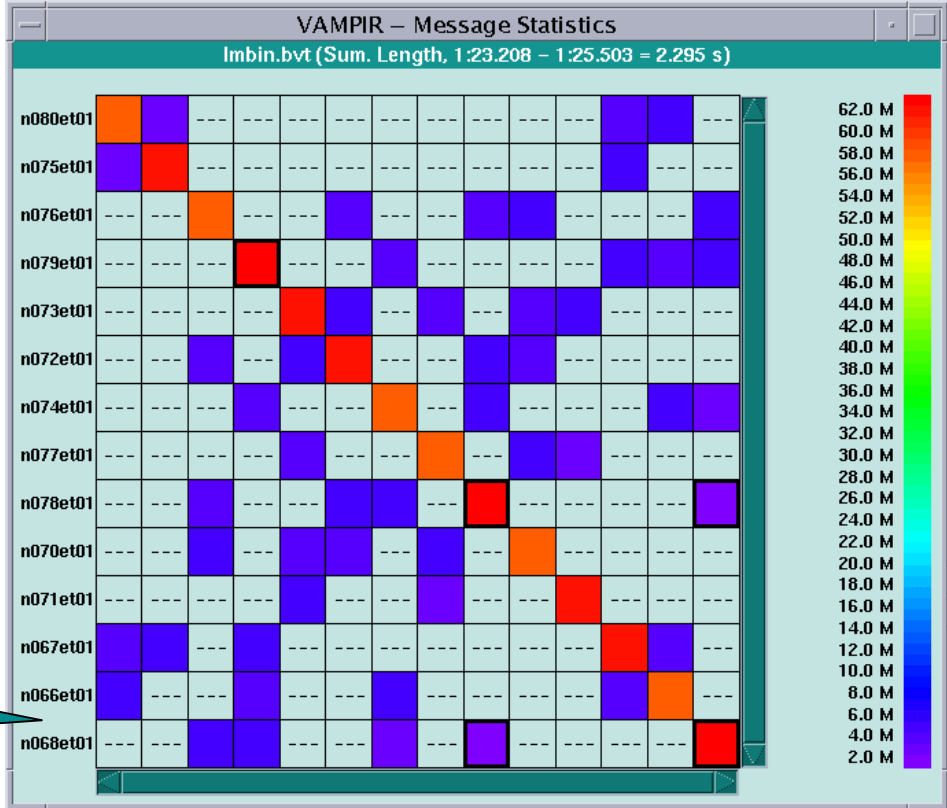
# LM – Zeitschritt Analyse



Statistik über alle Prozesse



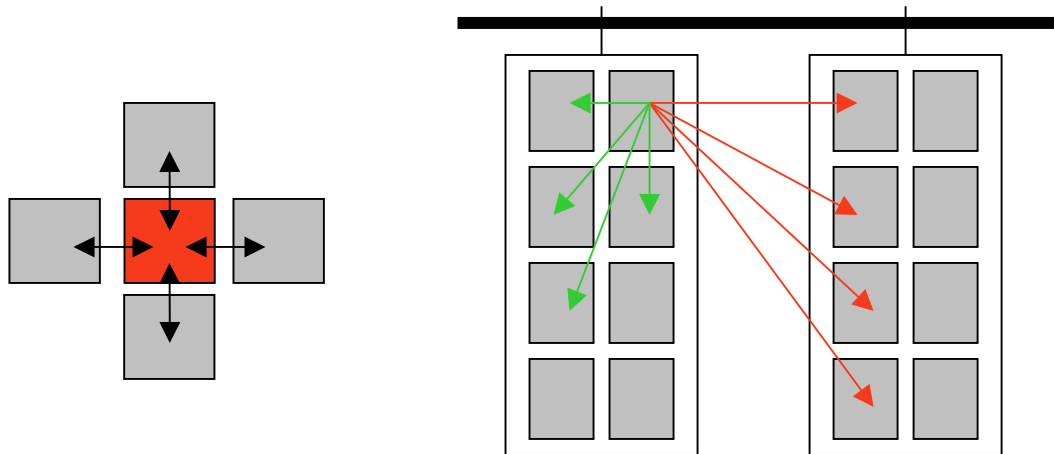
Kommunikation zwischen den Prozessen



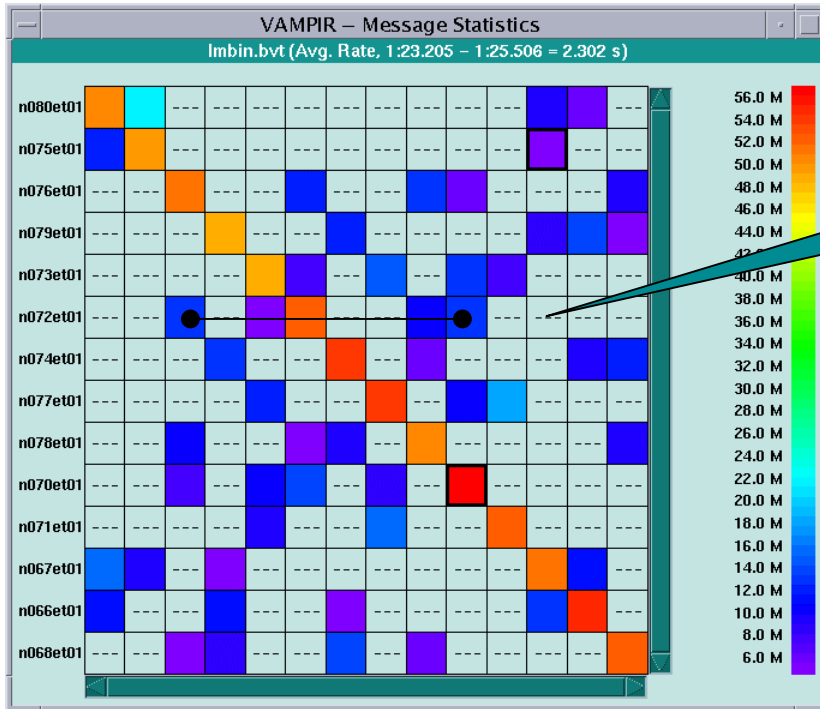
Kommunikation zwischen den SMP Knoten



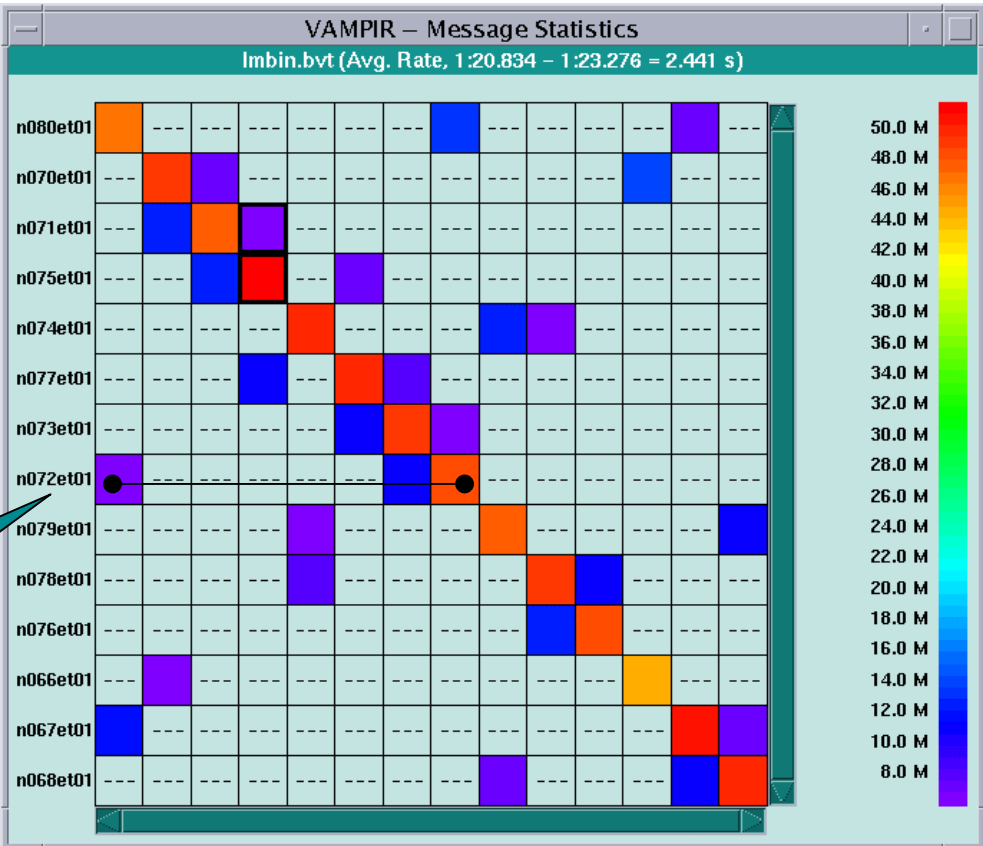
- MPI Kommunikation beträgt ca. 16% der Laufzeit
- Lastverteilung ist gut (geringer Kommunikationsoverhead)
- Kommunikations-Matrix
  - Typische 2–D Nachbarkommunikation
  - **Zuviel** off–node Kommunikation
- Lösungsansatz: Verbessertes Mapping durch Inner-Knoten-Kommunikation (*shared memory **statt**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



# Verbesserung des Prozessor Mappings



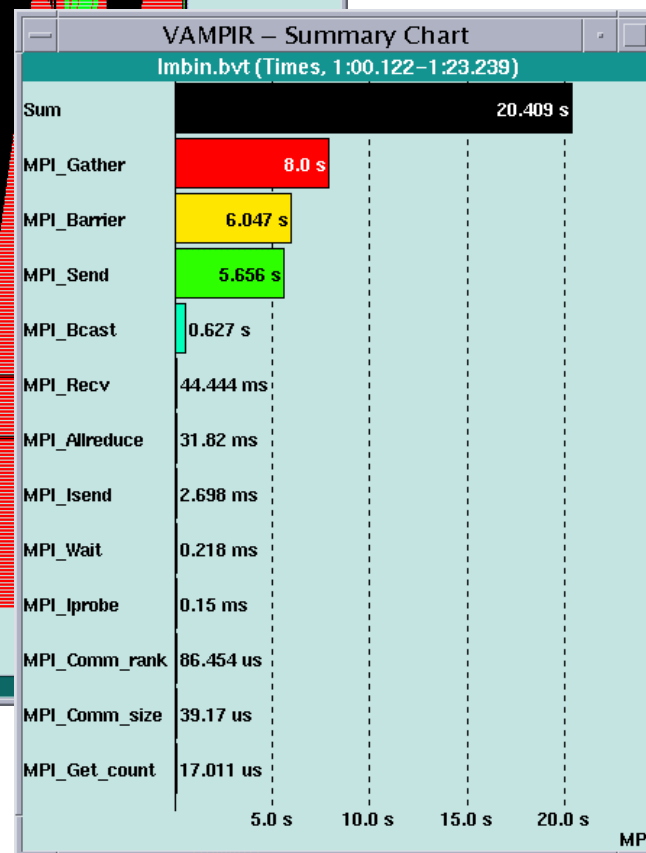
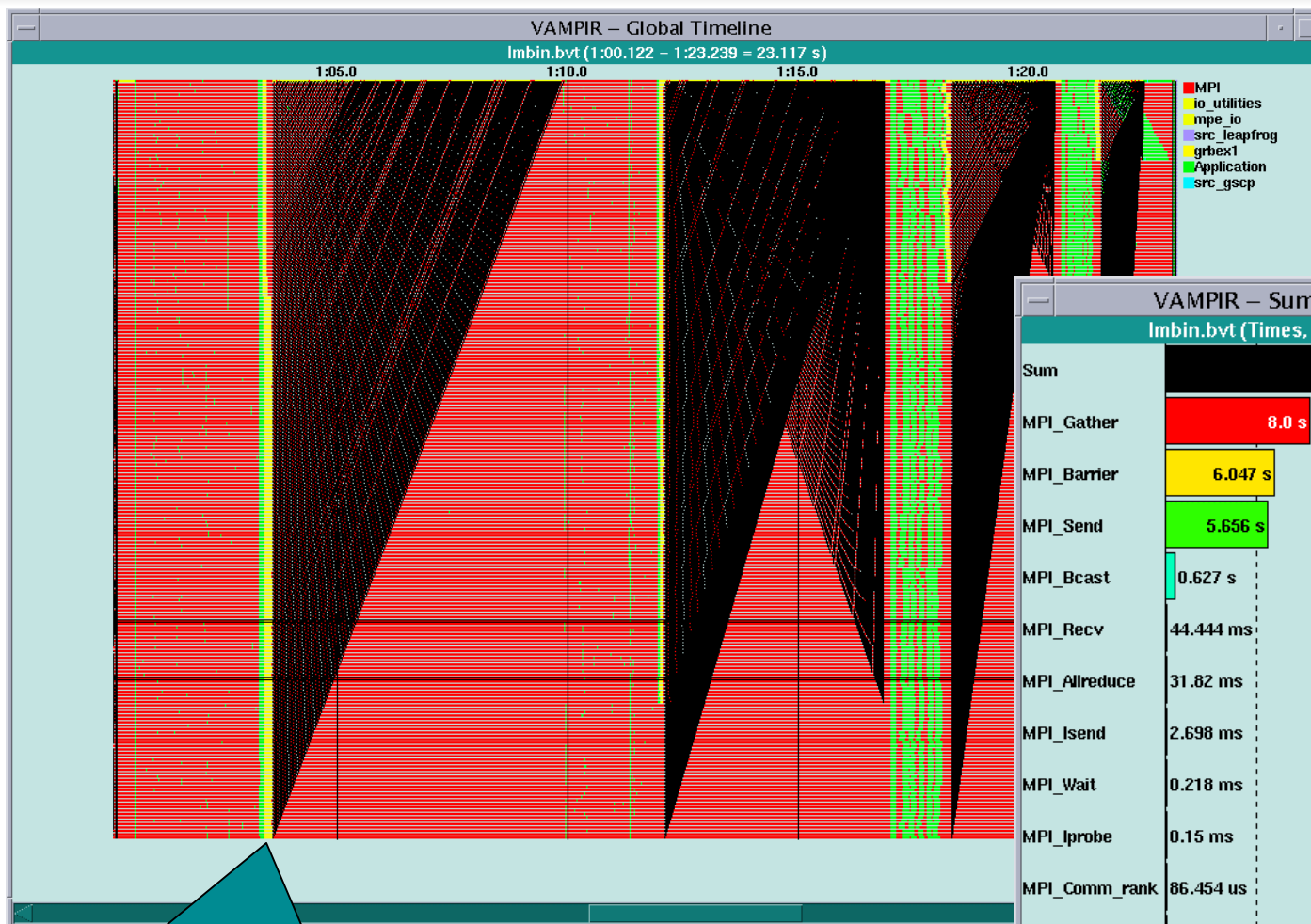
Ursprüngliches Knotenmapping



Verbessertes Knotenmapping



# Analyse der Output Phase

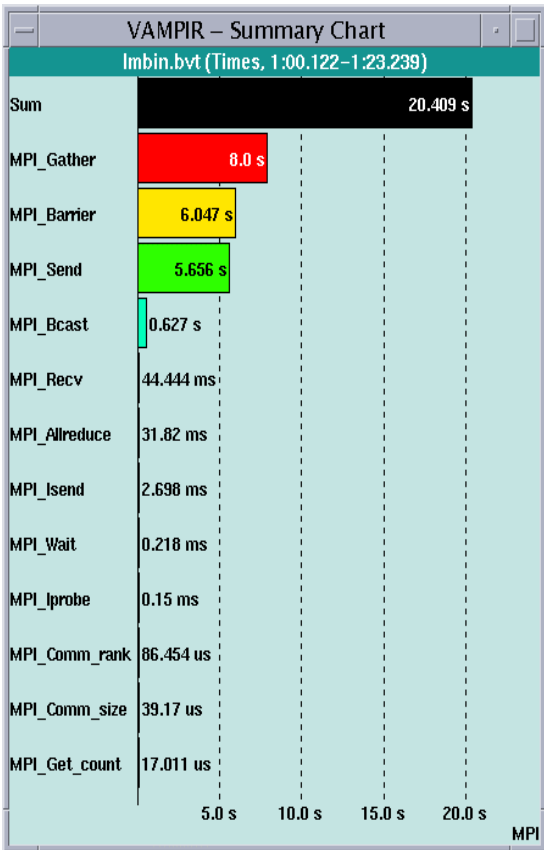
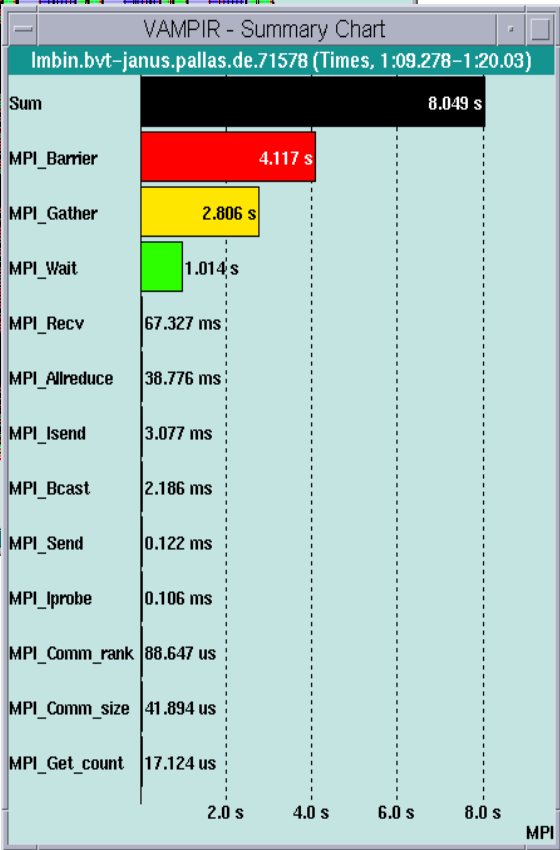
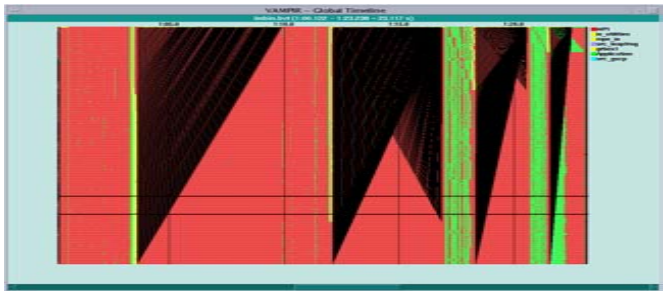
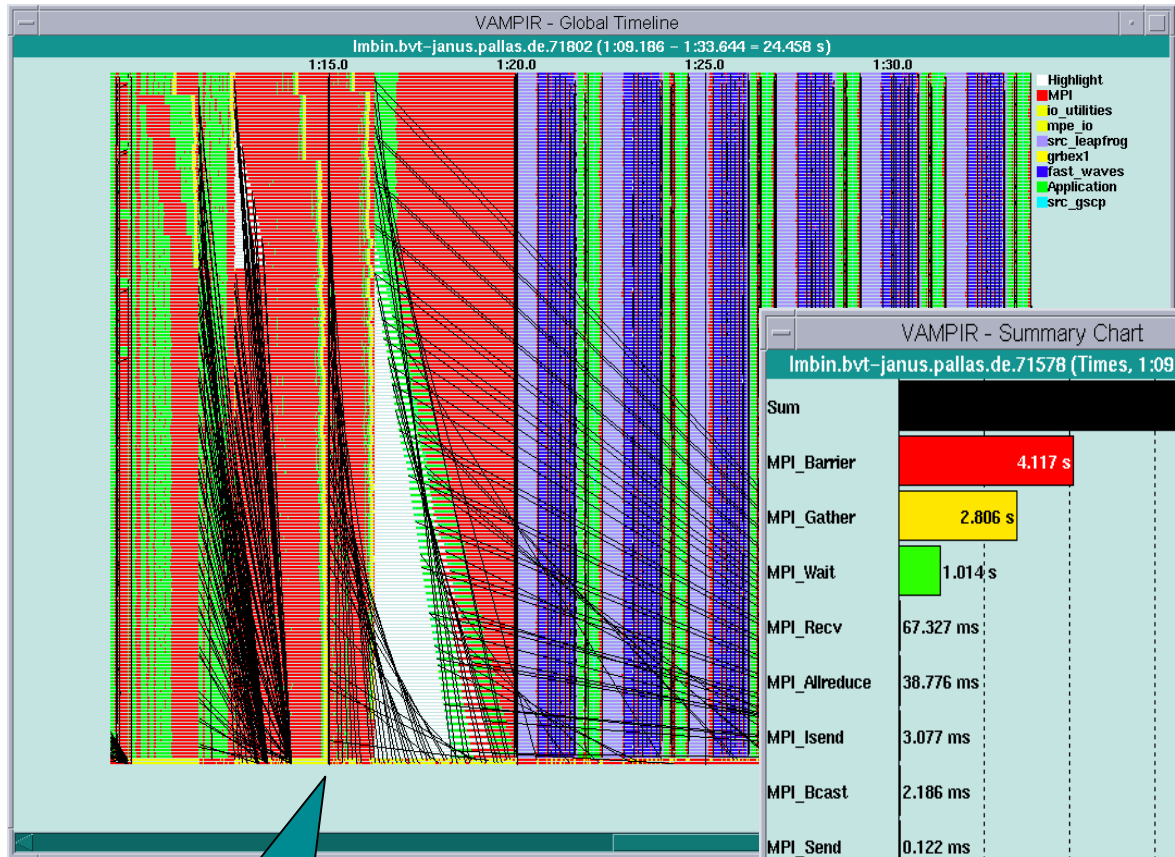


Alle Prozesse senden an Prozess 0



- 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



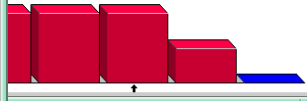
Überlagerung Berechnung- I/O

# OpenMP Analyse

Project: parbugs Data File: parbugs

- Write -> Write I1 in PARBUGS
- 2 Errors in PARALLEL region: PARBUGS846-55
- Write -> Read ISQMAX in PARBUGS -> ISQMAX in PARBUGS
- Parallel I/O incorrectly synchronized in PARBUGS
- 2 Errors in PDO: PARBUGS847-51
- Write -> Read ISQMAX in PARBUGS
- Write -> Write ISQMAX in PARBUGS

File: parbugs.f Routine: PARBUGS @ 46



Source & Sink: parbugs.f

```


34 ! Add "reduction(max: isqmax)" to correct these problems.
35
36 ! The nowait clause means that the printout of
37 ! isqmax could occur before isqmax gets its final values.
38 ! Assume will report a write->read conflict. Remove the "nowait"
39 ! clause to correct.
40
41 ! The I/O here should be synchronized. Assume will report this.
42 ! Fix by placing the write statement inside a "!$omp single"/
43 ! "$omp end single" pair.
44
45 isqmax = 0
46 !$omp parallel private(i)
47 !$omp do
48 do i = 1, imax
49 isqmax = max(isquared(i))
50 end do
51 !$omp end do nowait
52
53 write(*,*) "Maximum value"
54
55 !$omp end parallel
56
57 ! When iinit is made priv
58 ! not -1. Assume will rec
59 ! Fix by placing iinit ir
60 ! a "lastprivate()" claus
61 iinit = -1
62 !$omp parallel do private(iinit)
63 do i = 1, imax

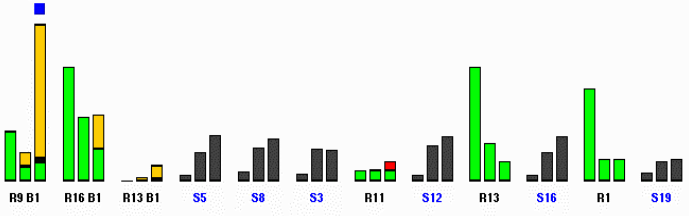
```

Thread Average Region Times

0.0 s. sequential time	24.9 s. barriers time
0.0 s. sequential ovh.	0.6 s. imbalance time
0.0 s. synchronized time	0.4 s. parallel ovh.
0.3 s. locks time	3.5 s. parallel time

R9:B1 in file apsi.f, routine HVD,  
at line 2952, # invocations 0. Total time = 29.8 s.  
Input file apsi\_kpts.8.stat.





Input file order: F1 F2 F0



- 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 !



- Information pro Thread
- Statistik (Bereich/Gesamt)

The screenshot displays the KAI GuideView interface for analyzing an OpenMP program. It is divided into three main windows:

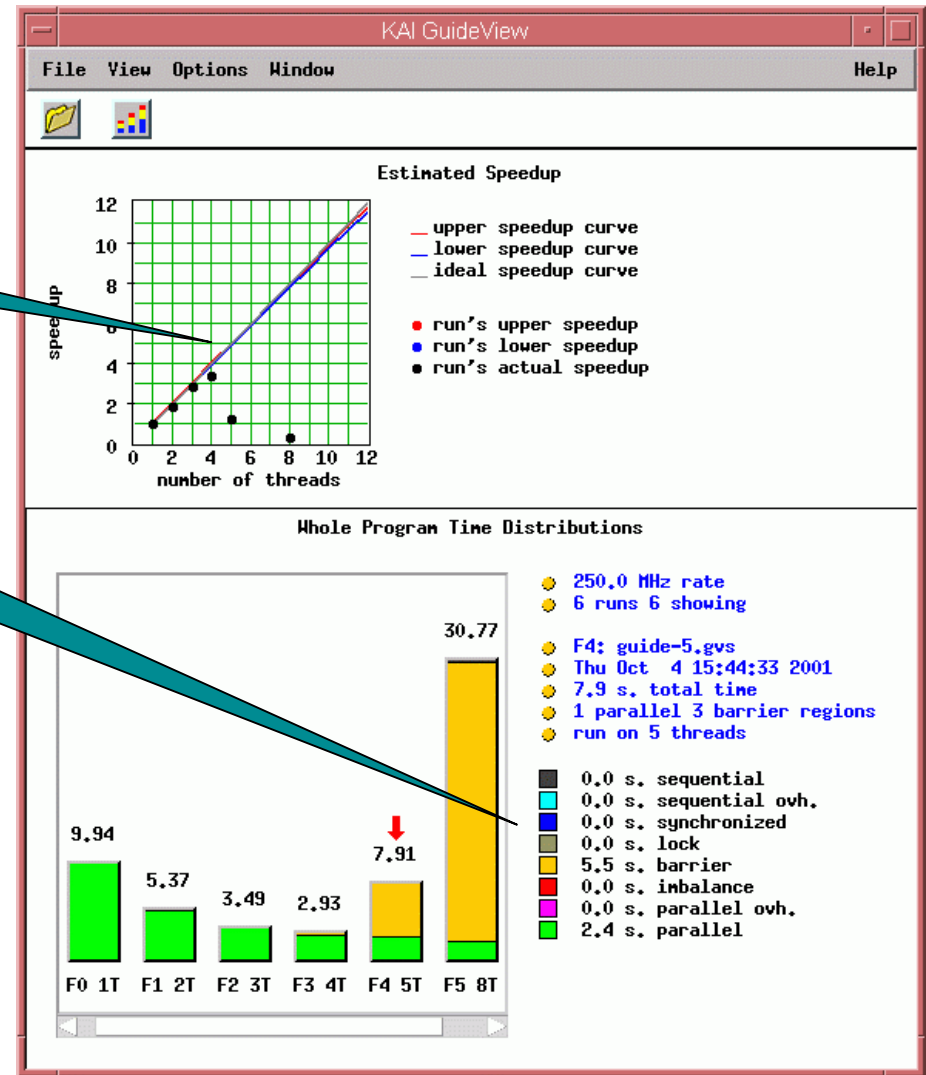
- Whole Program:** Shows a bar chart of execution times for different components: idle time (0.0 s), sequential (0.0 s), synchronize (0.0 s), and lock time (0.0 s). A red arrow points to the 'Total time' bar.
- Thread-Spec:** Contains a 'speedup' graph with 'number of threads' on the x-axis (0 to 8) and 'speedup' on the y-axis (0 to 8). A blue line shows a linear increase in speedup with the number of threads. Below the graph is a diagram of thread order: S1, R1, R1 B1, R1 B2, R1 B3, S2, S3. A red arrow points to S1, and a blue arrow points to R1 B2.
- Code Editor (File cppG\_matmul.f):** Shows the source code for a matrix multiplication. A region starting at line 54 is highlighted in black, indicating the region of interest for the analysis. The code includes nested loops for j, i, and k, and a parallel region for the innermost loop.

# OpenMP Beispiel - Matrixmultiplikation

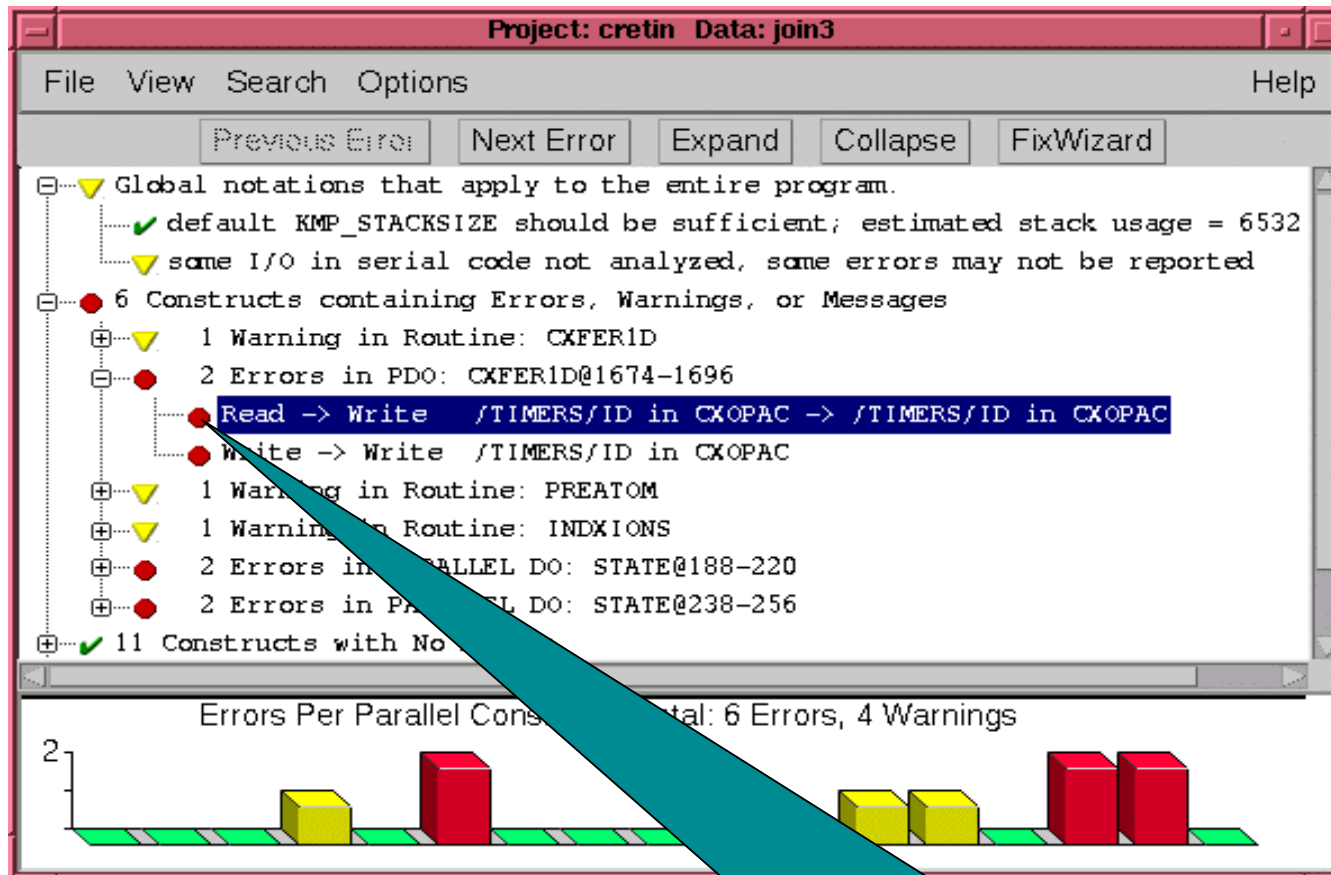


Vergleich idealer-tatsächlicher Performance – *speed up*

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



# Thread Analyse – parallele Probleme



Synchronisationsprobleme (*memory leaks*)



# Thread Analyse - *deadlock*



Project: demo Data File: dead

File View Search Print Preferences Reorder Windows Help

- Global notations that apply to the entire program.
- Deadlock in main was detected.
  - Thread in work\_1 wants lock on object held in work\_2.**
  - Thread in work\_2 wants lock on object held in work\_1.
- Execution data for Thread main.
- Execution data for Thread deadlock\_test.
- Execution data for Thread deadlock\_test.

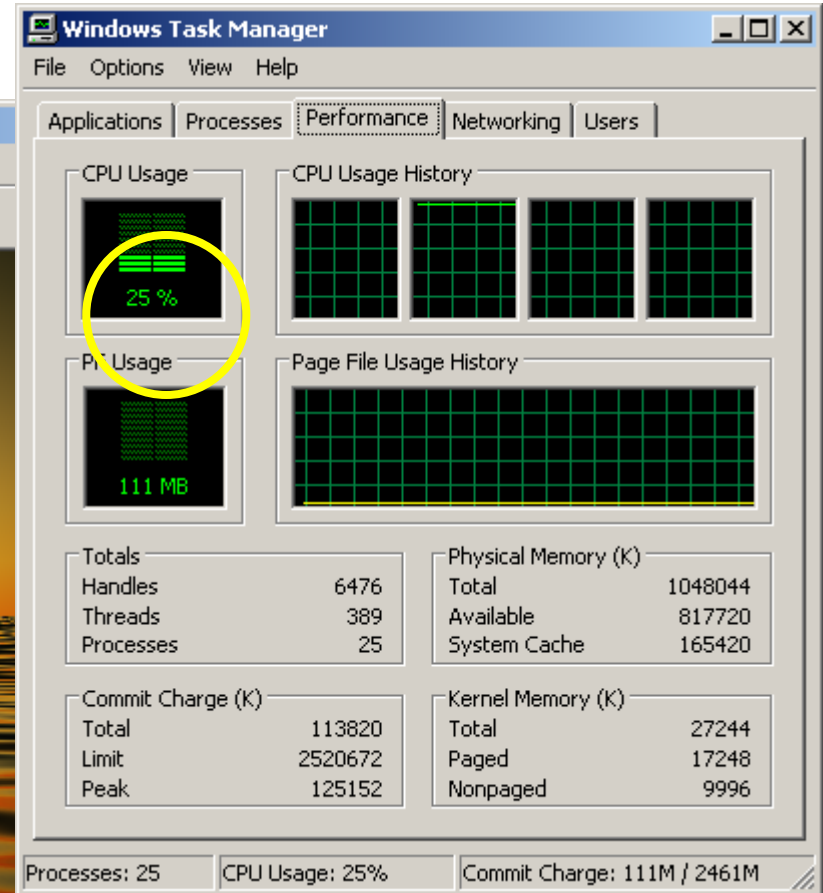
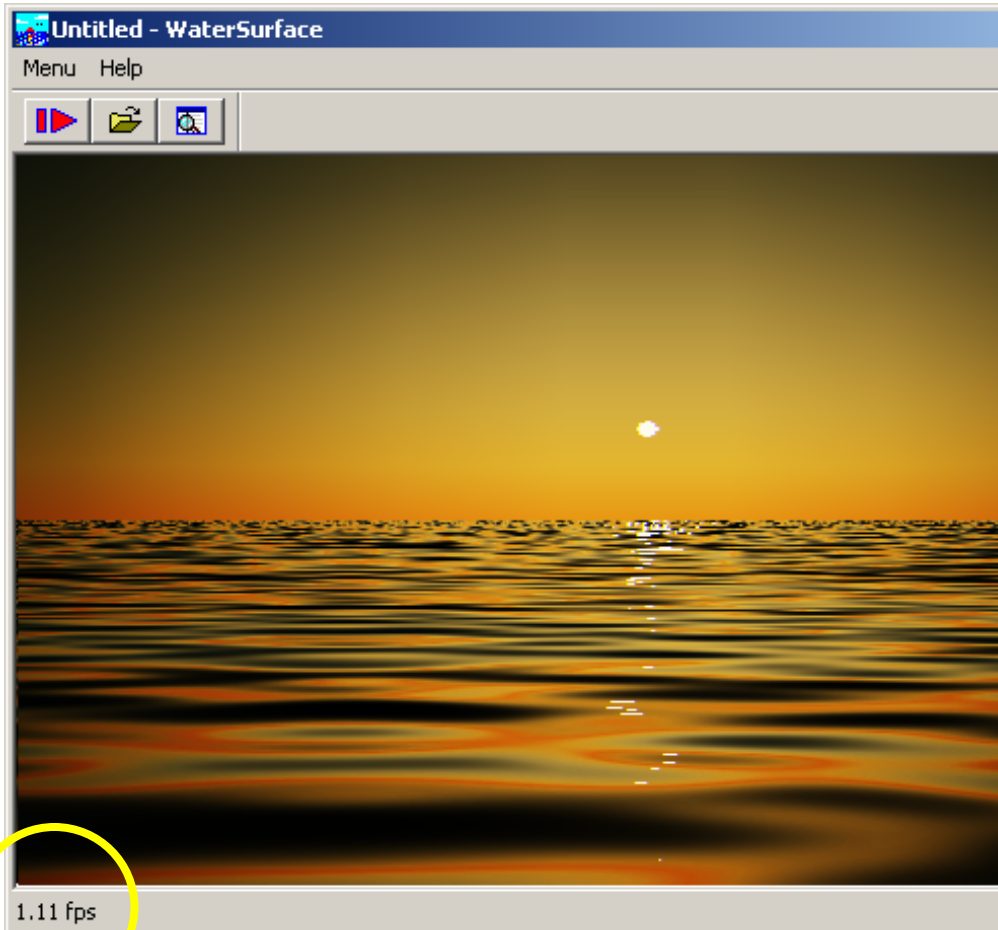
Source: dead.c

```
15
16 void *work_1( void *arg )
17 {
18     int i;
19
20     pthread_mutex lock( & lock 1 );
21     pthread_mutex lock( & lock 2 );
22
23     for ( i = 0; i < ITERS; ++i)
24         sum = sum + 1;
25
26     pthread_mutex_unlock( & lock_2 );
27     pthread_mutex_unlock( & lock_1 );
28
29     return 0;
30 }
```

Sink: dead.c

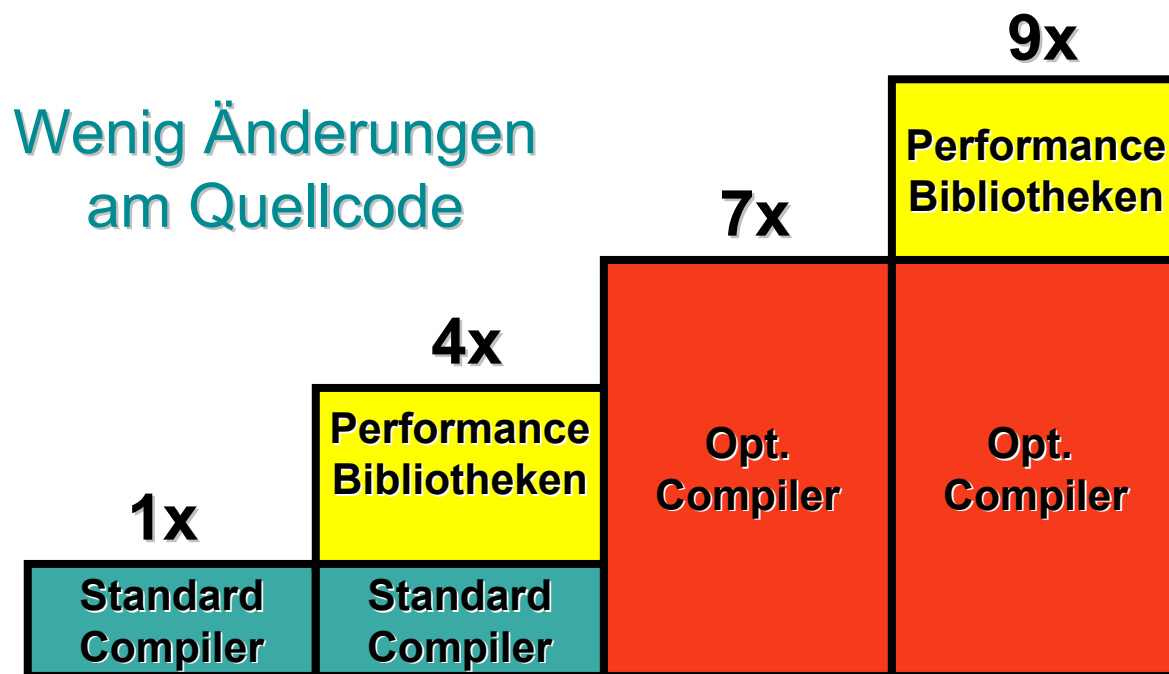
```
31
32 void *work_2( void *arg )
33 {
34     int i;
35
36     pthread_mutex lock( & lock 2 );
37     pthread_mutex_lock( & lock_1 );
38
39     for ( i = 0; i < ITERS; ++i)
40         sum = sum + 1;
41
42     pthread_mutex_unlock( & lock_1 );
43     pthread_mutex_unlock( & lock_2 );
44
45     return 0;
46 }
```

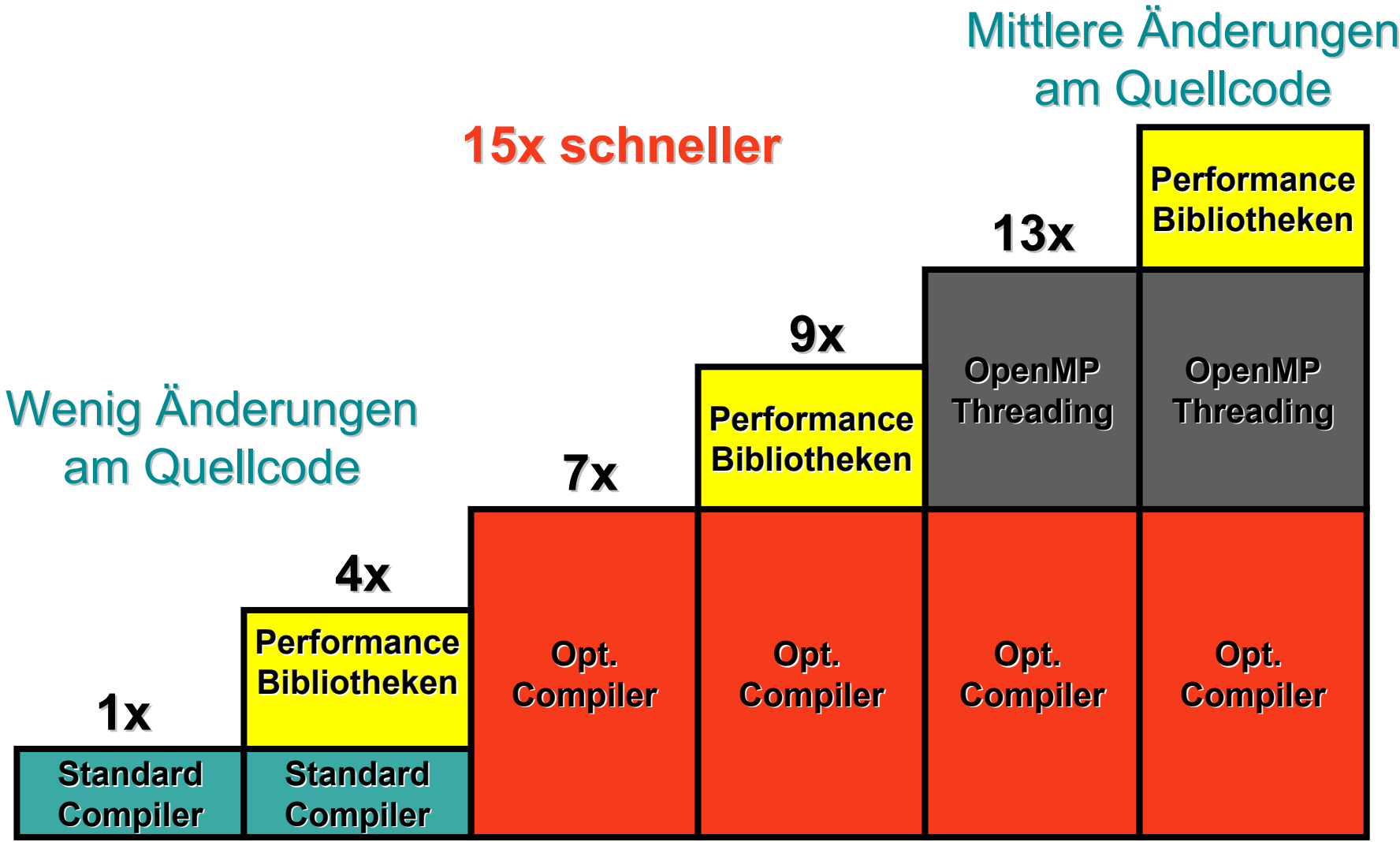
# Basis Performance – nicht optimierter Code





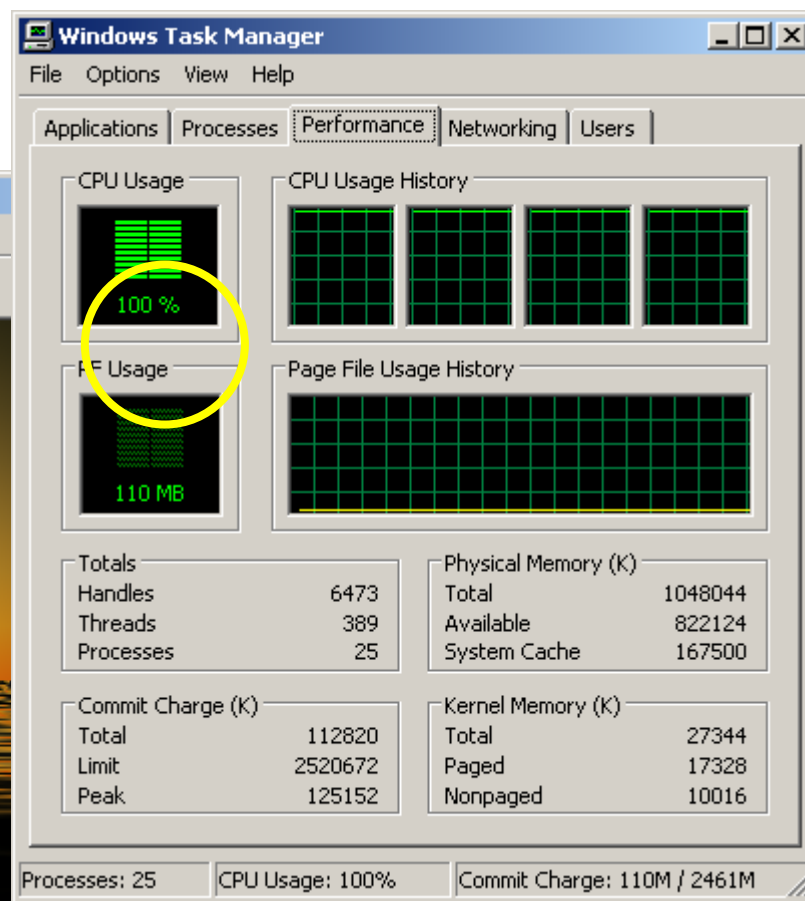
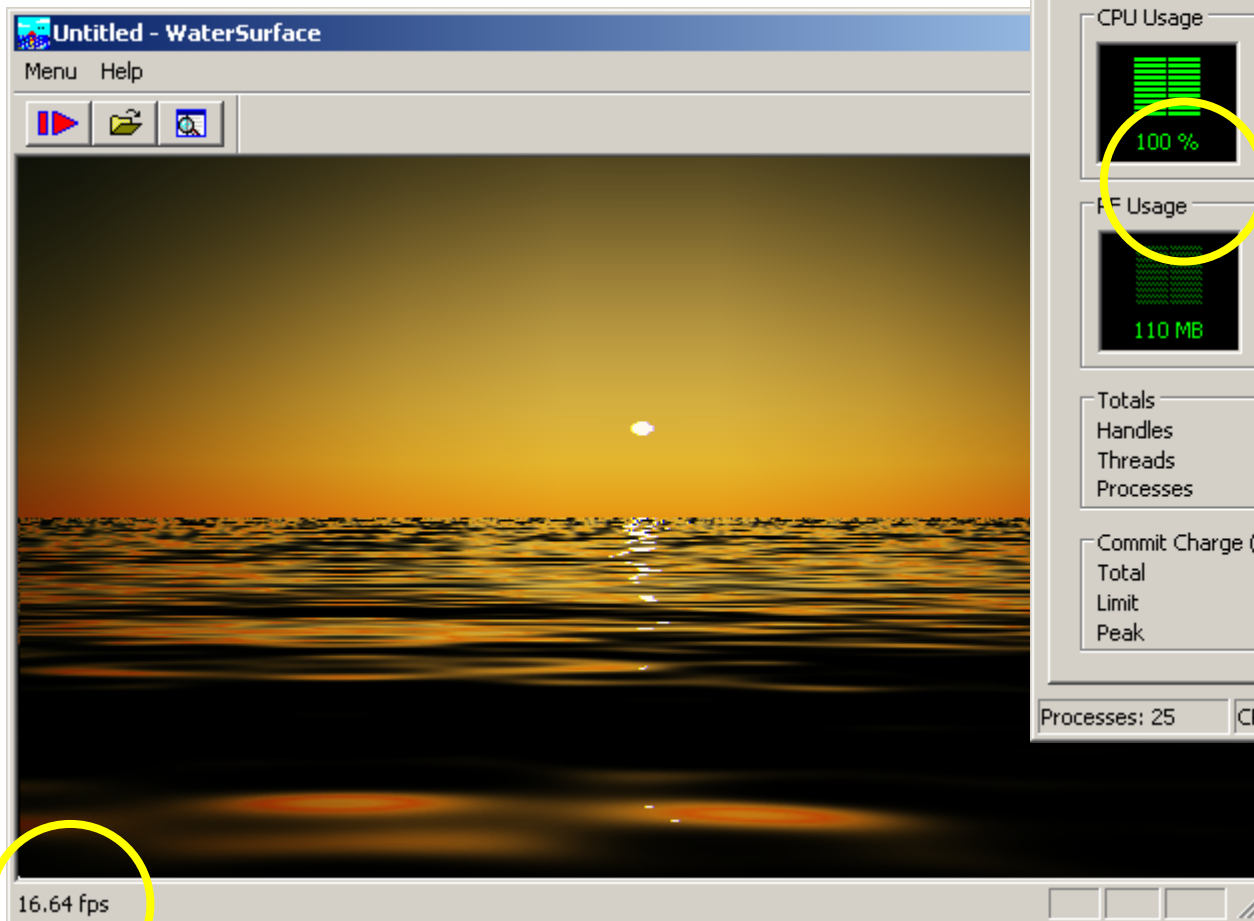
**9x schneller**

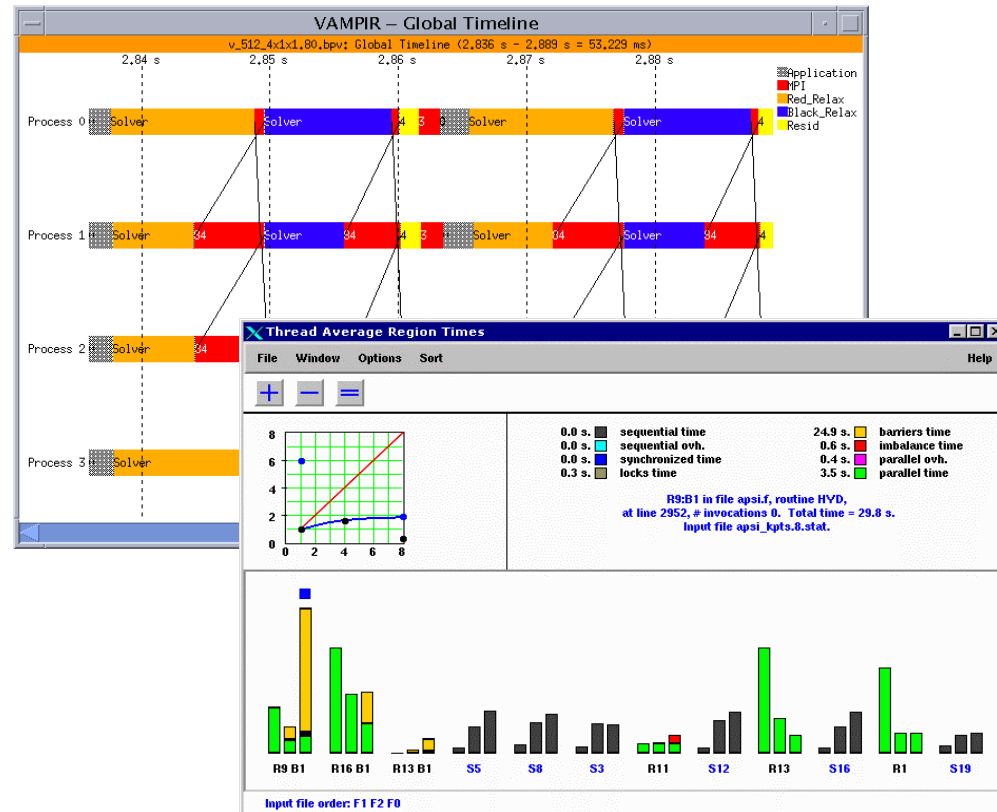




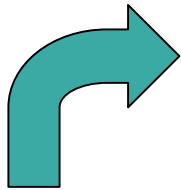


## 15x schneller

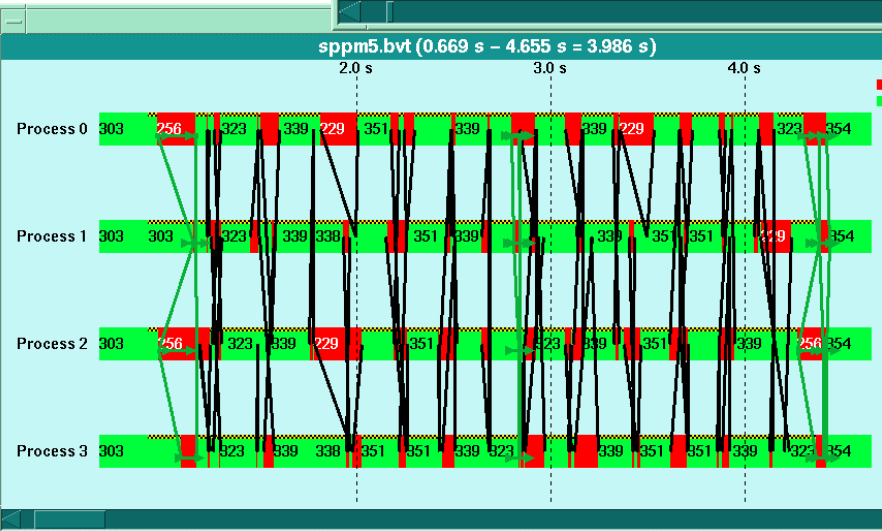
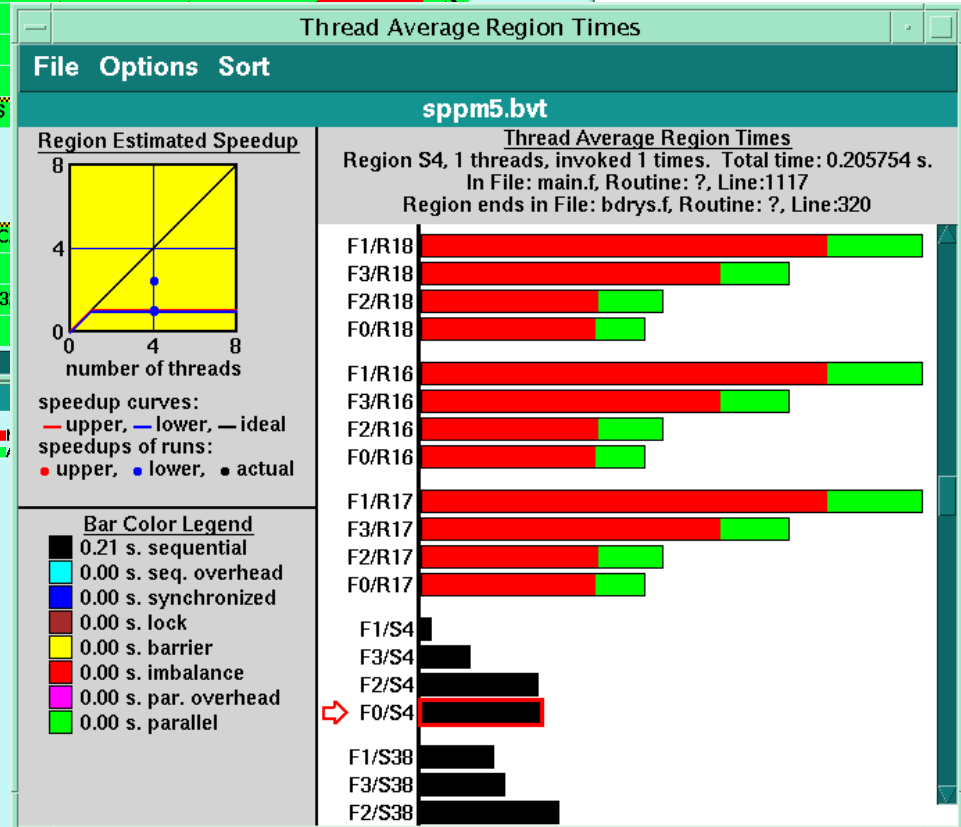
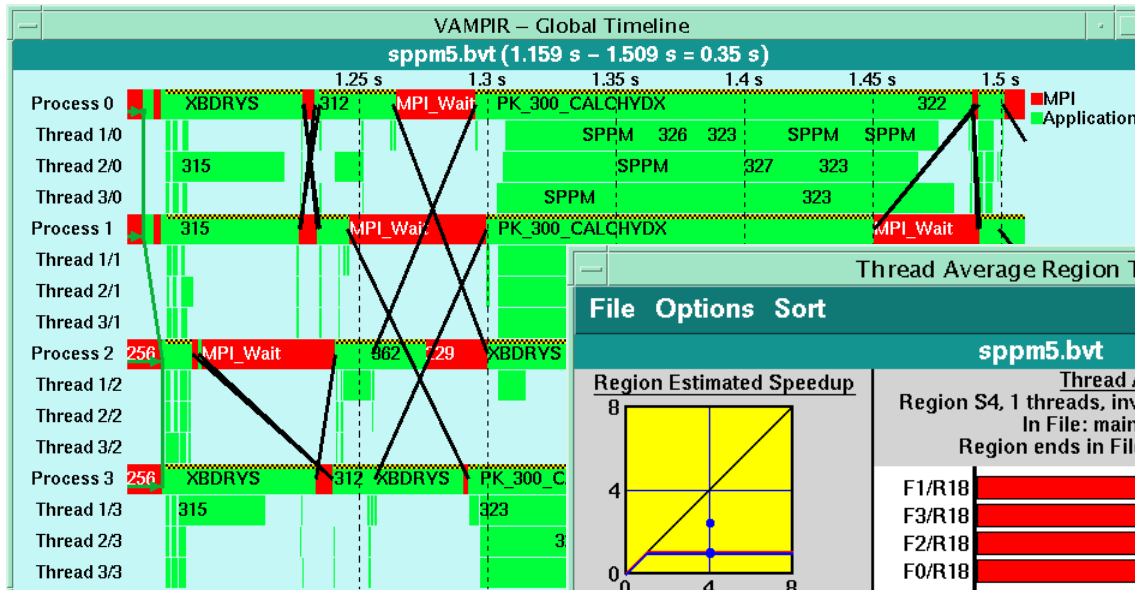




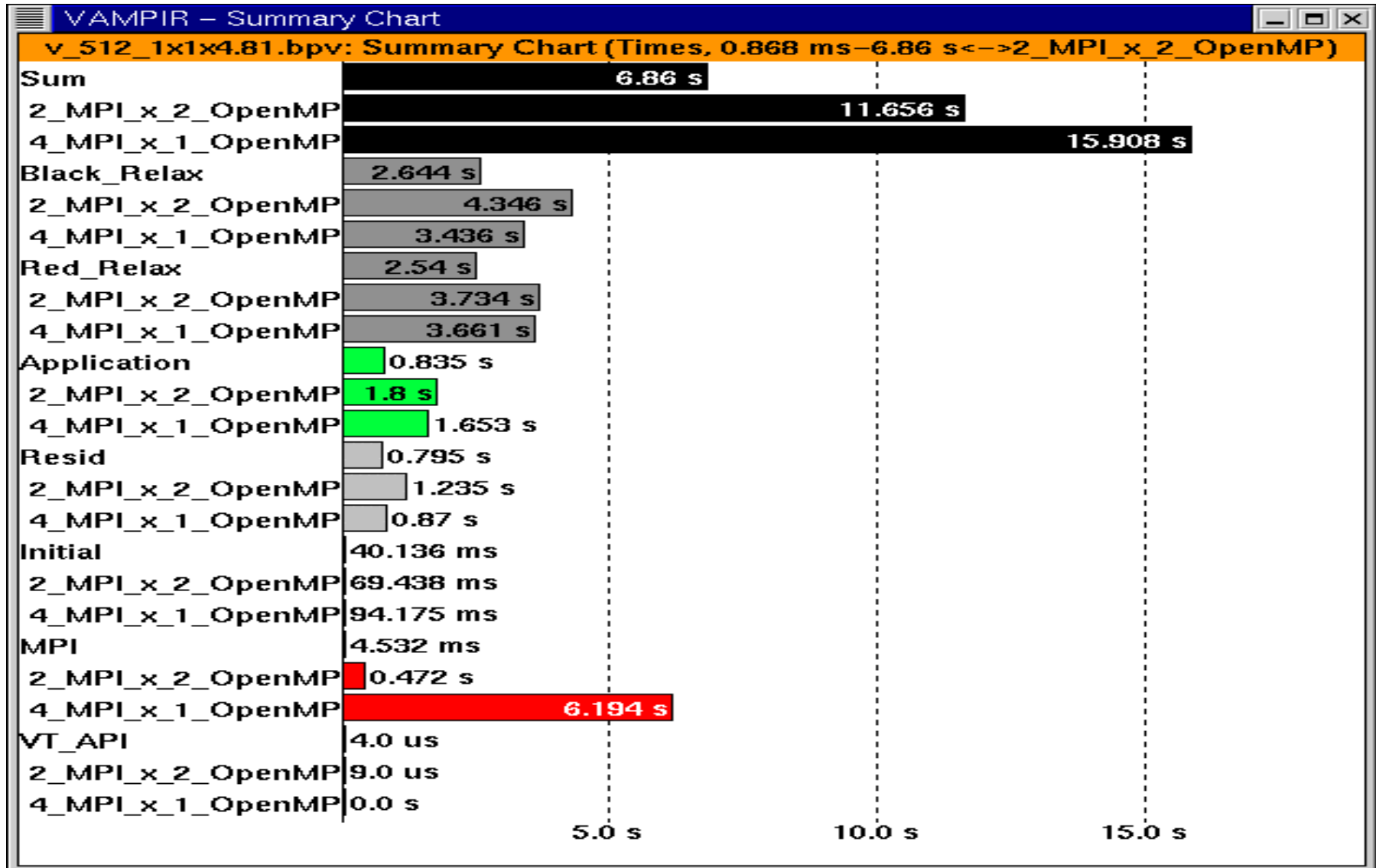
# MPI – OpenMP Analyse



expandieren



# MPI+OpenMP – verschiedene Programmläufe







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](mailto:info@pallas.com)  
[www.pallas.com](http://www.pallas.com)