Assure and KAP/Pro Toolset: Tools for Parallel Software Engineering

Dave Poulsen

Kuck and Associates, Inc.

poulsen@kai.com

http://www.kai.com

KAI
OPTIMIZING SOFTWARE

Outline

- Introduction
- Assure and its uses in parallel programming
- How to use Assure on a parallel program
- Examples of errors detected by Assure
- Implementation of Assure
- The AssureView GUI

KAI
OPTIMIZING SOFTWARE

Kuck and Associates

- Founded 1979
- ***KAP***: parallelizing compiler, vectorizer
  - Supports most vector, SMP parallel machines
  - Global scalar optimizer for workstations, PCs
- ***KAP/Pro Toolset***: directive parallelism
- ***KAI C++***: state-of-industry C++ optimizer

KAI
OPTIMIZING SOFTWARE

KAP/Pro Toolset

- Shared-memory parallel Fortran
- OpenMP directive-based parallelism
- First full implementation in industry
- ***Guide***: OpenMP for Unix, Windows-NT
- ***GuideView***: performance analysis, tuning
- ***Assure***: correctness, validation, debugging
- ***KAP***: automatic parallelization

KAI
OPTIMIZING SOFTWARE

OpenMP Directives

Parallel Region

```
C$OMP PARALLEL
C$OMP& [IF (if_expression)]
C$OMP& [SHARED(shared_variables)]
C$OMP& [PRIVATE(local_variables)]
C$OMP END PARALLEL
```

Parallel Do

```
C$OMP DO
C$OMP&[SCHEDULE(type,chunk)]
C$OMP END DO [NOWAIT]
```

Parallel Sections

```
C$OMP SECTIONS
C$OMP SECTION
C$OMP END SECTIONS [NOWAIT]
```

Single Processor Sections

```
C$OMP SINGLE
C$OMP END SINGLE [NOWAIT]
```

KAI
OPTIMIZING SOFTWARE

OpenMP Directives

Parallel Data	*Synchronization*
`C$OMP THREAD PRIVATE [/common/,…]`	`C$OMP CRITICAL [(variable)]`
`C$OMP& [COPYIN (variables)]`	`C$OMP END CRITICAL`
`C$OMP& [PRIVATE(variables)]`	`C$OMP ORDERED`
`C$OMP& [SHARED(variables)]`	`C$OMP END ORDERED`
`C$OMP& [FIRSTPRIVATE(variables)]`	`C$OMP MASTER`
`C$OMP& [LASTPRIVATE(variables)]`	`C$OMP END MASTER`
`C$OMP& [REDUCTION(op : variables)]`	`C$OMP BARRIER`
`C$OMP& [DEFAULT` `C$OMP& (PRIVATE|SHARED|NONE)]`	`C$OMP ATOMIC`
	`C$OMP FLUSH`

OpenMP Library, Environment

KAI OPTIMIZING SOFTWARE

Run Time Library Routines	*Environment Variables*
`external omp_set_num_threads(integer)`	`OMP_SCHEDULE`
`integer  omp_get_num_threads()`	`OMP_NUM_THREADS`
`integer  omp_get_max_threads()`	`OMP_DYNAMIC`
`integer  omp_get_thread_num()`	`OMP_NESTED`
`integer  omp_get_num_procs()`	
`external omp_set_dynamic(logical)`	
`logical  omp_get_dynamic()`	
`logical  omp_in_parallel()`	
`external omp_set_nested(logical)`	
`logical  omp_get_nested()`	
`external omp_init_lock(var)`	
`external omp_init_destroy(var)`	
`external omp_set_lock(var)`	
`external omp_unset_lock(var)`	
`logical  omp_test_lock(var)`	

Example -- OpenMP Reduction

```
c$omp parallel
c$omp& shared(a,m,n,sum)
c$omp& private(i,j,suml)
    suml=0.0
c$omp do
    do 10 i=1,n
      do 10 j=1,m
10      suml=suml+a(j,i)
c$omp critical
    sum=sum+suml
c$omp end critical
c$omp end parallel
```

- Private ***suml*** accumulates per-thread
- Final reduction into shared ***sum***

KAI
OPTIMIZING SOFTWARE

Example -- OpenMP Reduction

```
c$omp parallel do
c$omp& shared(a,m,n)
c$omp& private(i,j)
c$omp& reduction (+ : sum)
      do 10 i=1,n
        do 10 j=1,m
10        sum=sum+a(j,i)
```

- OpenMP ***reduction*** keyword:
  - Scalars or array elements
  - Operators such as +, -, max, min

KAI
OPTIMIZING SOFTWARE

Assure: Parallel Software Engineering

- Validates correctness of OpenMP programs
- Detects several classes of problems:
  - Parallelization, logic errors
  - Misuse of OpenMP directives
  - Errors that affect numerical consistency
  - Potential races, deadlock conditions
- Can be used to parallelize large serial codes

KAI
OPTIMIZING SOFTWARE

What Does Assure Do?

- Validates correctness of parallel programs
- Defines correct behavior by contained serial program (i.e., with directives removed)
- When validated, a program is correct for:
  - A particular input dataset used during a run
  - Any number of processors used
  - All possible execution timing variations
  - Across all supported platforms

KAI
OPTIMIZING SOFTWARE

Assure-Detected Errors

- Unintended data dependences
  - Unsynchronized access to shared variables
  - Unintended sharing of variables
  - Unintended aliasing
- Incorrect OpenMP directive usage
- Deadlock conditions
- Parallel I/O operations

KAI
OPTIMIZING SOFTWARE

Assure-Detected Errors

- Serial/parallel transition problems
  - Copy-in at beginning of parallel regions
  - Copy-out at end of parallel regions
  - Re-use of private variables across regions
  - Uninitialized reads
- Inconsistent declarations
  - Shared vs. private variables
  - COMMON blocks

KAI
OPTIMIZING SOFTWARE

Assure-Detected Errors

- Numerical consistency
  - Unordered reductions, accumulations
  - Incorrect synchronization
- Parallel stack size problems

KAI
OPTIMIZING SOFTWARE

Assure vs. Traditional Approaches

- Debuggers
  - Time-consuming, postmortem analysis
  - Requires bugs to be reproducible
  - Does not guarantee finding all ***potential*** bugs
  - Threaded debugging is difficult at best
- Parallelize and debug incrementally
  - Human-intensive, not automatable

KAI
OPTIMIZING SOFTWARE

Advantages of Assure

- Finds cases where parallel program ***might*** behave differently than serial program
- Automated, shows all potential errors for a given input dataset
- Detects errors without parallel execution
  - Allows diagnosis of programs that hang or abort when executed in parallel
  - Detects intermittent parallel failures

KAI
OPTIMIZING SOFTWARE

Using Assure

- Single source file
  - Compile source with assuref77, not f77
  - Run resulting executable
  - assureview [-txt]
- Multiple source files
  - Compile sources with assuref77 -pname=pgm
  - Run resulting executable
  - assureview [-txt] pgm

KAI
OPTIMIZING SOFTWARE

Assure Options

- assuref77 –h gives command-line options:
  - –pname=<file> if multiple source files
  - –check={slow,medium,fast} (default slow)
  - –location={exact,approx} (default exact)
- setenv KDD_INTERVAL NNm
  - – Can run assureview every NN minutes
  - – Errors are often caught on initial loop iterations

KAI
OPTIMIZING SOFTWARE

Example -- Reduction

```
c$omp parallel do
c$omp& shared(a,m,n,sum)
c$omp& private(i,j)
      do 10 i=1,n
        do 10 j=1,m
10        sum=sum+a(j,i)
c$omp end parallel do
```

- Naive parallelization of serial algorithm
- Could fix with ***critical*** (inhibits parallelism)

```
Errors in PARALLEL DO:
Write --> Read   SUM  MAIN/6
Write --> Write  SUM  MAIN/6
```

Reduction -- Almost Fixed

```
c$omp parallel
c$omp& shared(a,m,n,sum)
c$omp& private(i,j,suml)
    suml=0.0
c$omp do
    do 10 i=1,n
      do 10 j=1,m
10      suml=suml+a(j,i)
c need c$omp critical here
    sum=sum+suml
c$omp end parallel
```

```
Errors in PARALLEL region:
Read  --> Write  SUM  MAIN/10
Write --> Read   SUM  MAIN/10
Write --> Write  SUM  MAIN/10
Error: MAIN/10: PRIVATE
   symbol SUML used outside
   worksharing construct/
   iteration where defined
   (MAIN/8)
Errors in PDO:
Error: MAIN/8: SUML cannot be
   a PRIVATE variable due to
   reference at (MAIN/8)
```

KAI
OPTIMIZING SOFTWARE

Reduction -- Fixed

```
c$omp parallel
c$omp& shared(a,m,n,sum)
c$omp& private(i,j,suml)
    suml=0.0
c$omp do
    do 10 i=1,n
      do 10 j=1,m
10      suml=suml+a(j,i)
c$omp critical
    sum=sum+suml
c$omp end critical
c$omp end parallel
```

Example -- Flow Dependence

```
c$omp parallel do
c$omp& shared(a,b,c)
c$omp& private(i)
      do 10 i=2,n
        a(i)=b(i)
10      c(i)=a(i-1)
c$omp end parallel do
```

- "True dependence" -- requires restructuring

```
Errors in PARALLEL DO:
Write --> Read   A  MAIN/5-6
```

KAI
OPTIMIZING SOFTWARE

Flow Dependence -- Fixed

```
c$omp parallel
c$omp& shared(a,b,c)
c$omp& private(i)
c$omp do
    do 10 i=2,n
10    a(i)=b(i)
c$omp do
    do 20 i=2,n
20    c(i)=a(i-1)
c$omp end parallel
```

Example -- Output Dependence

```
c$omp parallel do
c$omp& shared(a,b,c)
c$omp& private(i)
    do 10 i=1,n
10    a=b(i)
c$omp end parallel do
    print *,a
```

- "Storage dependence" -- privatize to fix

```
Errors in PARALLEL DO:
Write --> Write  A  MAIN/5
```

Output Dependence -- Almost Fixed

```
c$omp parallel do
c$omp& shared(b,c)
c$omp& private(a,i)
    do 10 i=1,n
10    a=b(i)
c$omp end parallel do
    print *,a
```

- Last value from privatized ***a*** needed after ***parallel do***

```
Errors in PARALLEL DO:
Error: MAIN/7: PRIVATE symbol
   A referenced outside
   parallel construct (write
   in last iteration of PDO)
```

KAI
OPTIMIZING SOFTWARE

Output Dependence -- Fixed

```
c$omp parallel do
c$omp& shared(b,c)
c$omp& private(i)
c$omp& lastprivate(a)
      do 10 i=1,n
10      a=b(i)
c$omp end parallel do
      print *,a
```

Example -- Anti Dependence

```
      a(1)=0.0
c$omp parallel do
c$omp& shared(a,b,c)
c$omp& private(i)
      do 10 i=2,n
        b(i)=a(i-1)
10      a(i)=c(i)
c$omp end parallel do
```

- "Storage dependence" -- privatize to fix

```
Errors in PARALLEL DO:
Read --> Write  A  MAIN/6-7
```

Anti Dependence -- Fixed

```
      c(1)=0.0
c$omp parallel do
c$omp& shared(b,c)
c$omp& private(a,i)
      do 10 i=1,n-1
        a=c(i)
10      b(i+1)=a
c$omp end parallel do
```

KAI
OPTIMIZING SOFTWARE

Example -- Copy-In

```
c$omp parallel
c$omp& shared(a,c)
c$omp& private(b,i)
c$omp do
    do 10 i=1,10
10    a(i)=b(i)
c$omp do
    do 20 i=1,10
20    b(i)=b(i)+c(i)
c$omp do
    do 30 i=1,10
30    c(i)=b(i)
c$omp end parallel
```

- Private copies of ***b*** need initialization for first ***do*** loop

```
Errors in PDO (MAIN/4):
Error: MAIN/6: Uninitialized
   read of PRIVATE symbol B
   in worksharing construct
Errors in PDO (MAIN/7):
Error: MAIN/9: Uninitialized
   read of PRIVATE symbol B
   in worksharing construct
```

Copy-In -- Almost Fixed

```
c$omp parallel
c$omp& shared(a,c)
c$omp& private(i)
c$omp& firstprivate(b)
c$omp do
    do 10 i=1,10
10    a(i)=b(i)
c$omp do
    do 20 i=1,10
20    b(i)=b(i)+c(i)
c$omp do
    do 30 i=1,10
30    c(i)=b(i)
c$omp end parallel
```

- Problems can still occur if the three ***do*** loops are not scheduled identically

```
Errors in PDO (MAIN/11):
Error: MAIN/13: Private
   symbol B used outside
   worksharing construct/
   iteration where defined
   (MAIN/10)
```

KAI
OPTIMIZING SOFTWARE

Other Assure Errors

Warning: suggested KMP_STACKSIZE = nnn

- limit stack unlimited
- setenv KMP_STACKSIZE nnn
- large, SHARED main program arrays in COMMON blocks

Warning: some I/O not analyzed in {parallel,serial} code

- implied DO lists in I/O statements
- usually not serious in serial code

Error: parallel I/O incorrectly synchronized

- use (same) synchronization around these

Error: uninitialized read

- PRIVATE variable read before written within PARALLEL
- change PRIVATE to SHARED or use FIRSTPRIVATE

KAI
OPTIMIZING SOFTWARE

Other Assure Errors

Error: symbol cannot be a PRIVATE variable

- uninitialized READ + cross-iteration anti dependence

Warning: probable reduction, else...

- if properly-coded reduction, no problem

Error: PRIVATE symbol referenced outside parallel construct

- PRIVATE variable read before written after PARALLEL
- use LASTPRIVATE if most recent write was on last DO iteration

Error: LASTPRIVATE symbol referenced outside parallel construct
(write not in last iteration of DO)

- LASTPRIVATE requires write on last DO iteration

KAI
OPTIMIZING SOFTWARE

How Does Assure Work?

- Use ***assuref77*** instead of usual f77 compiler
  - Executable, serial program detects errors
  - *Project file* (.prj) describes input program
- Run program with an input dataset
  - *Output file* (.kdd) summarizes errors found
- Use ***assureview*** to view results
  - GUI displays errors, source code, call tree
  - Optional text report output

KAI
OPTIMIZING SOFTWARE

How Does Assure Really *Work?*

- ***assuref77*** instruments input program
  - Output serial program generates event trace
  - Produces/updates .prj file
- Execution simulates parallel runs from trace
  - Independent of # processors, execution timing
  - Produces .kdd file with error output
- ***assureview*** gathers and displays results by using .prj and .kdd files

KAI
OPTIMIZING SOFTWARE

Instrumentation, Trace Generation: assuref77

- Call ***ProcessEvent***() for events of interest:
  - ***c$omp*** construct entry/exit
  - Variable reads/writes
  - Routine entry/exit (for call tree)
- Parameters to ***ProcessEvent***():
  - Source location
  - Timestamp
  - Variable address

KAI
OPTIMIZING SOFTWARE

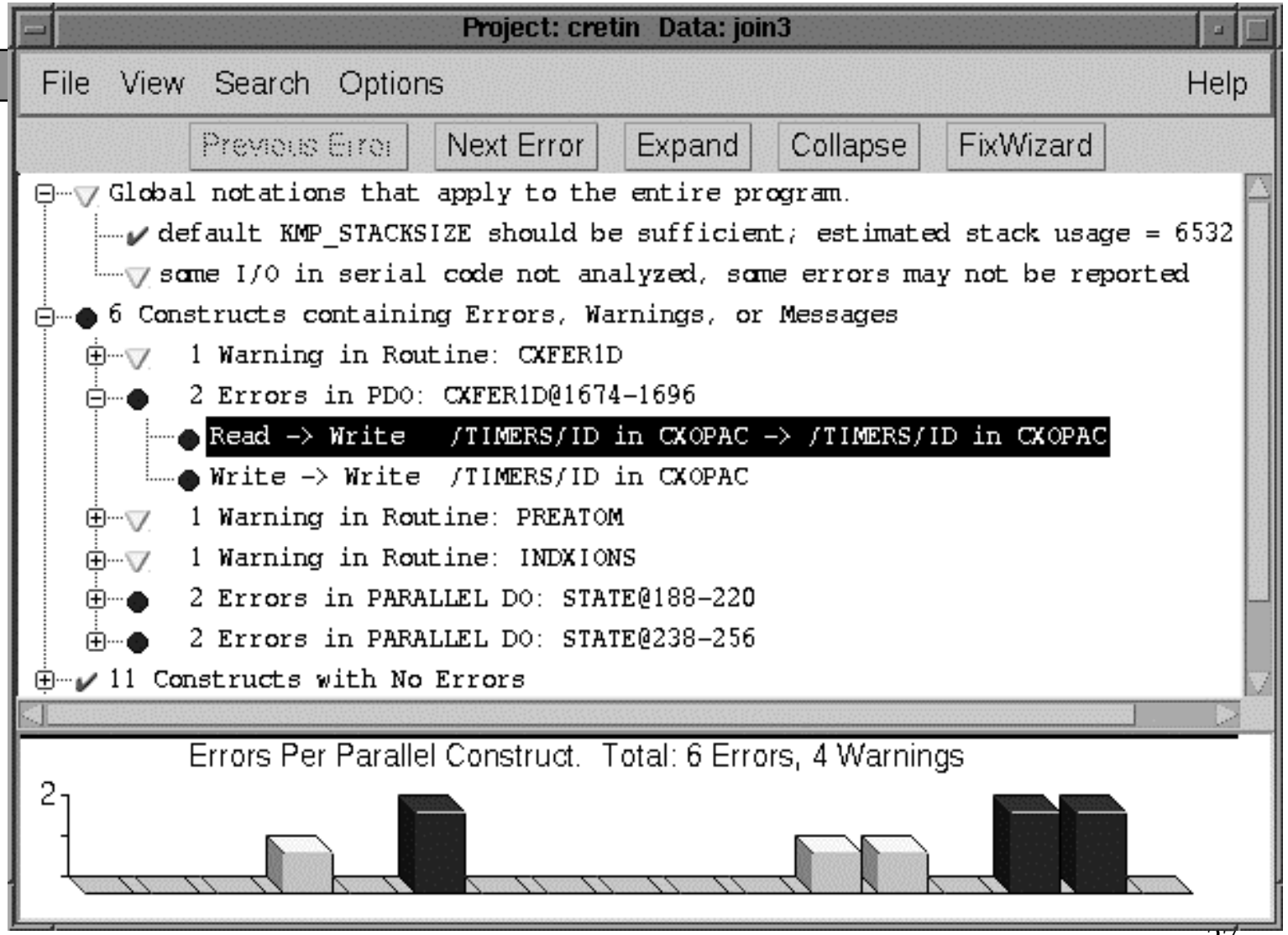
Trace Simulation: ProcessEvent() Routine

- Shadow memory
  - ***twA*** = timestamp of last write to ***A***
  - ***trA*** = timestamp of last read from ***A***
- Cross-iteration data dependence (***shared A***):
  - Two accesses in two different ***do*** iterations
  - At least one access is a write
  - Flow: ***twA*** < ***trA***
  - Anti: ***trA*** < ***twA***

Trace Simulation: ProcessEvent() Routine

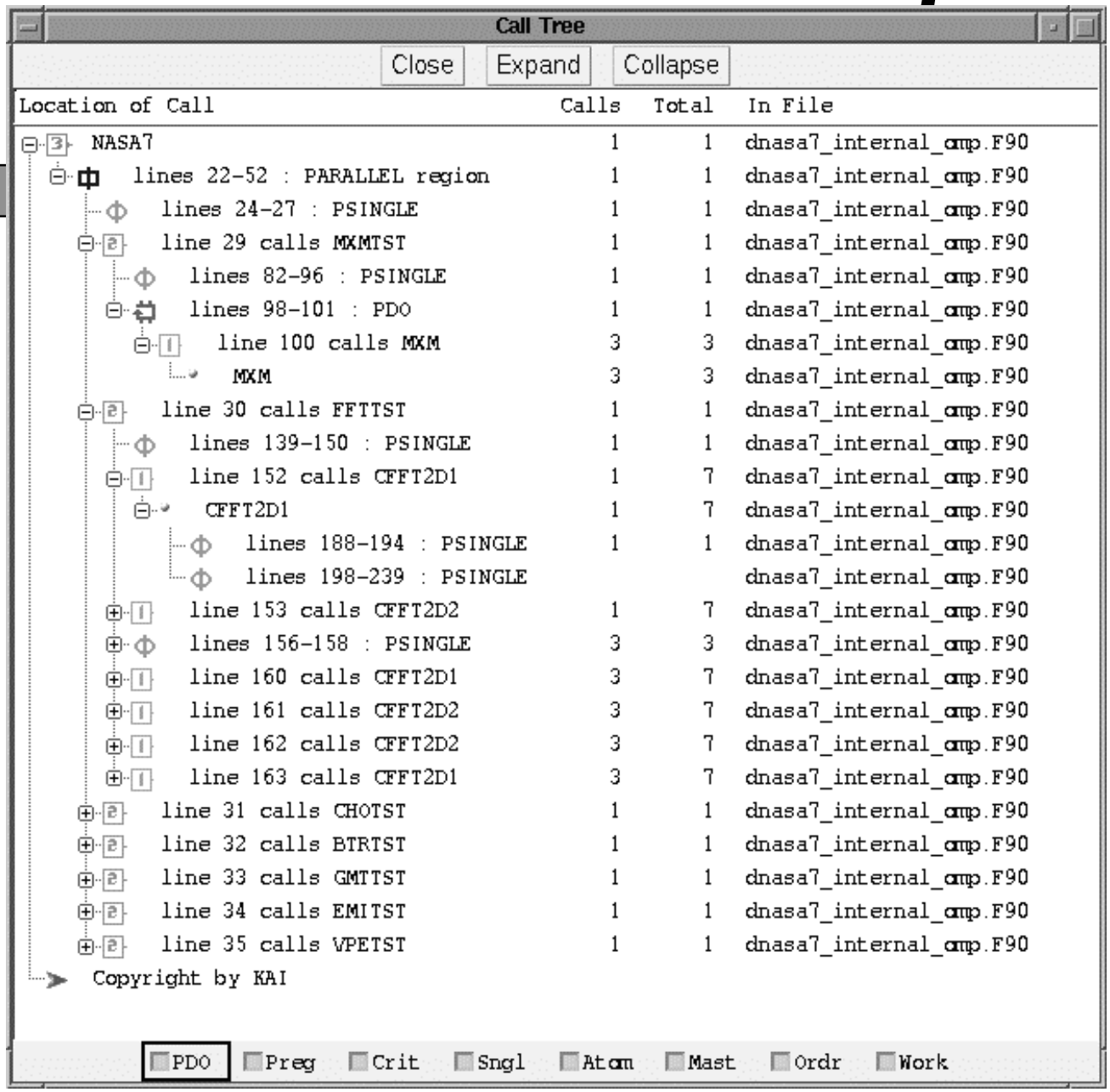
- Handling follows from definitions of errors
- Uninitialized read (***private*** ***A***):
  - Write occurs before start of ***parallel***
  - First reference within ***parallel*** is a read
- Reference outside parallel construct:
  - Write occurs within ***parallel***
  - First reference after ***parallel*** is a read

AssureView GUI

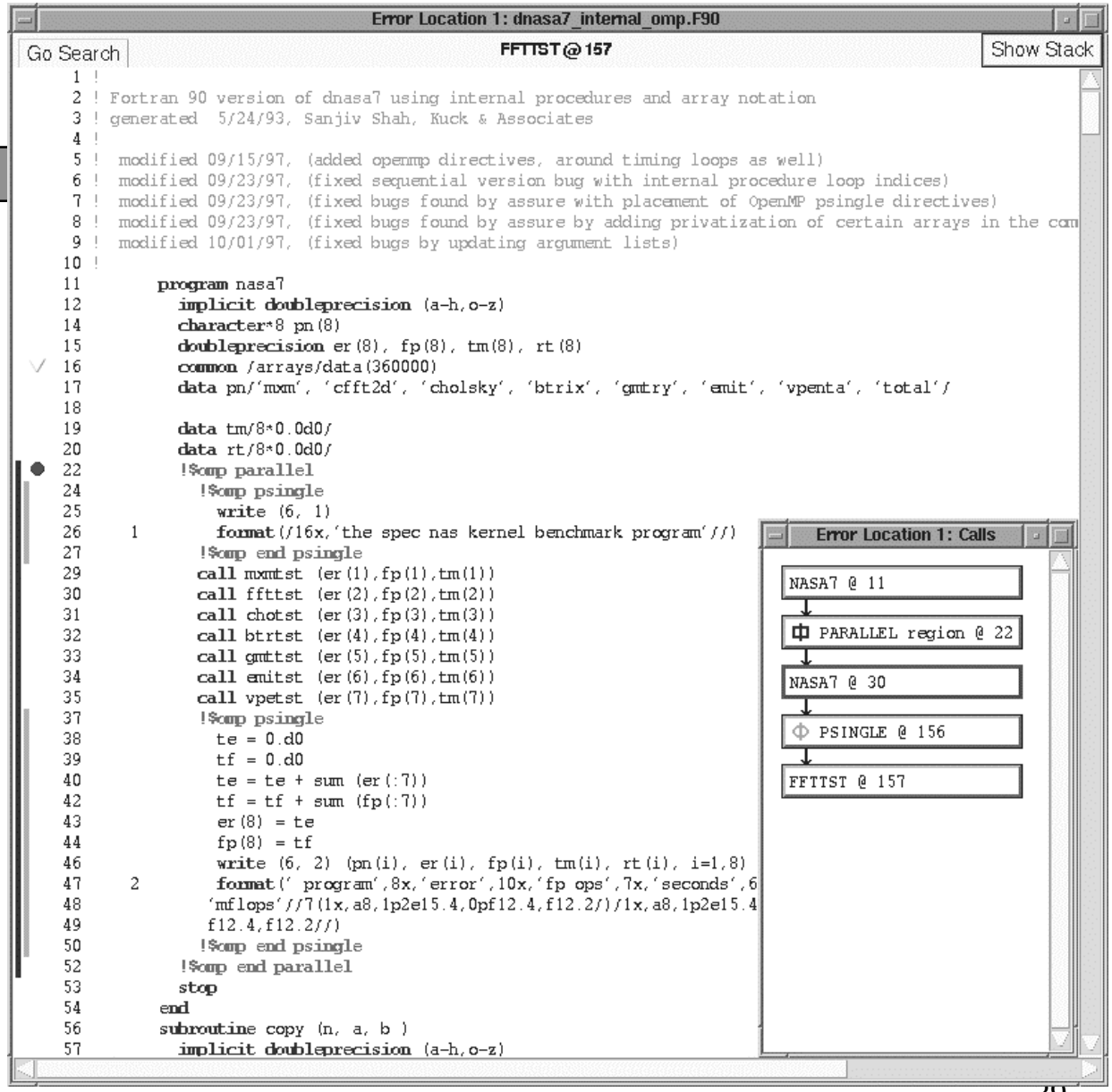


KAI
OPTIMIZING SOFTWARE

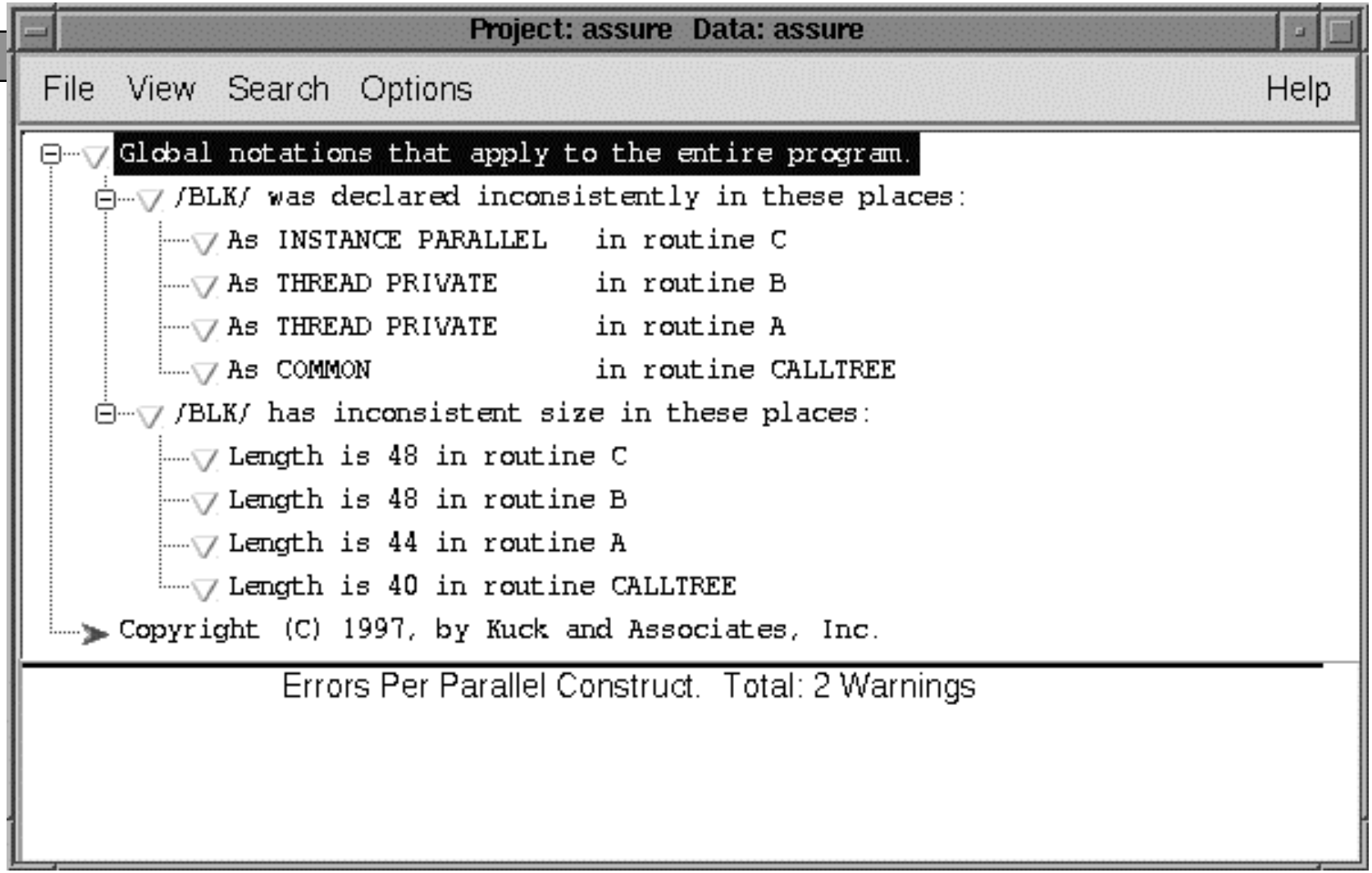
AssureView: Call Graph



AssureView: Source

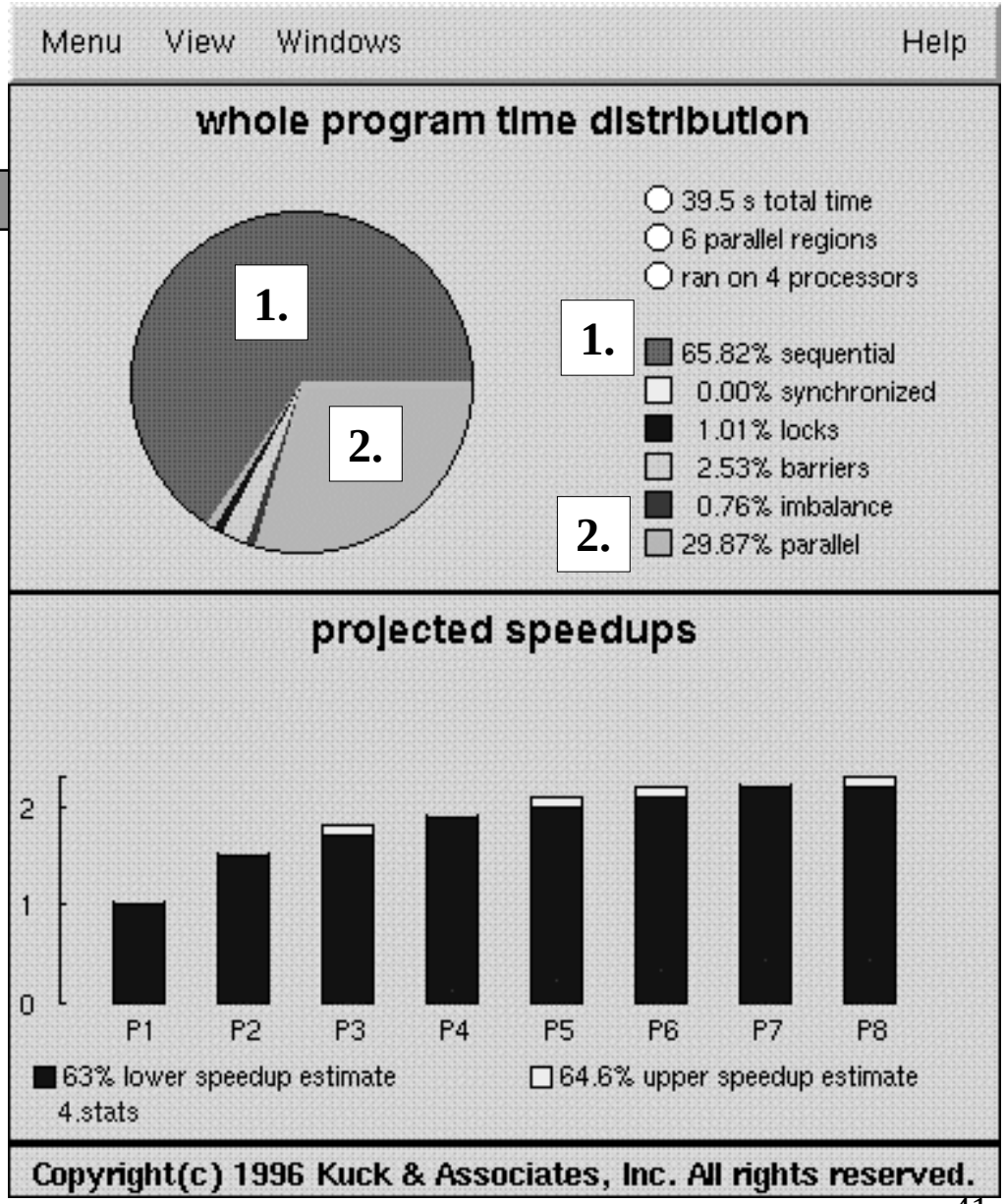


AssureView: COMMON Mismatch



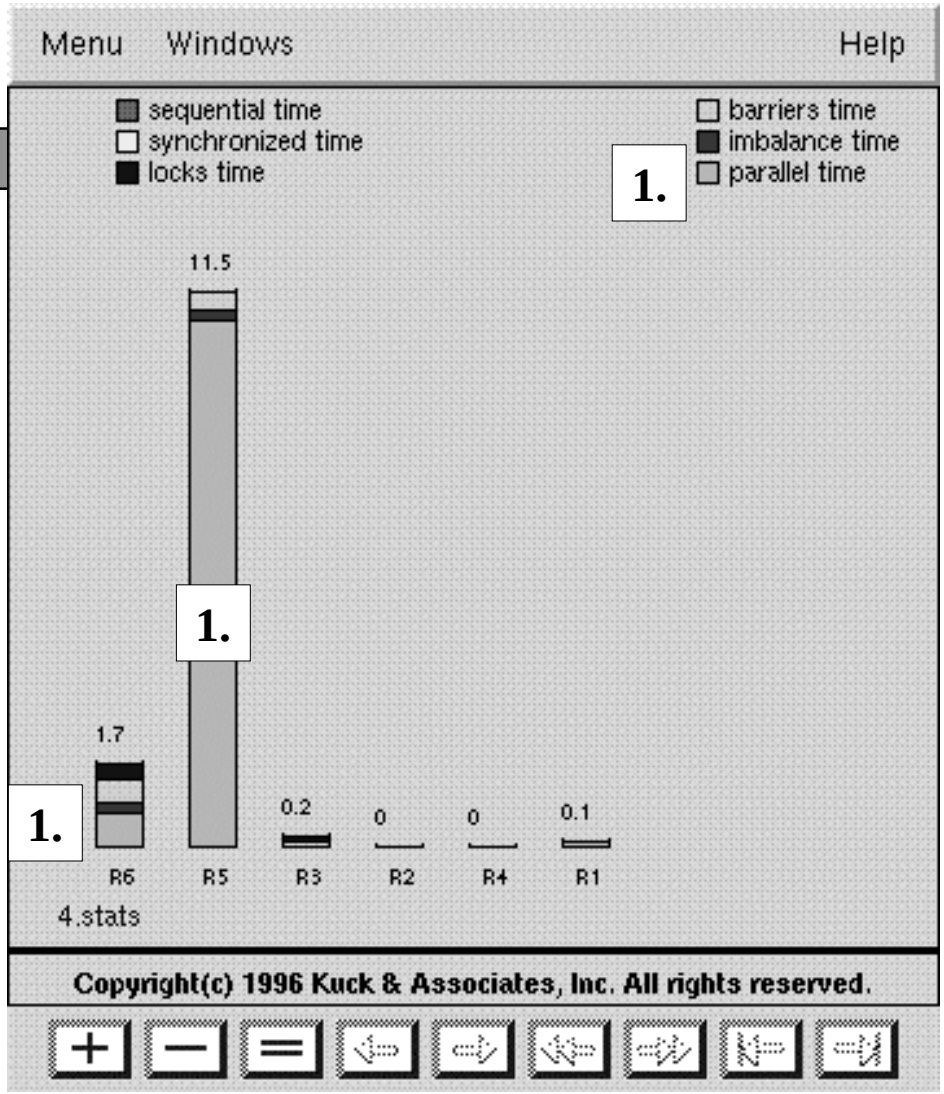
Example -- GuideView

- Pie chart
  - Percent parallel
  - Overhead
    - » Locks
    - » Barriers
    - » Imbalance
- Bar chart
  - Amdahl's Law



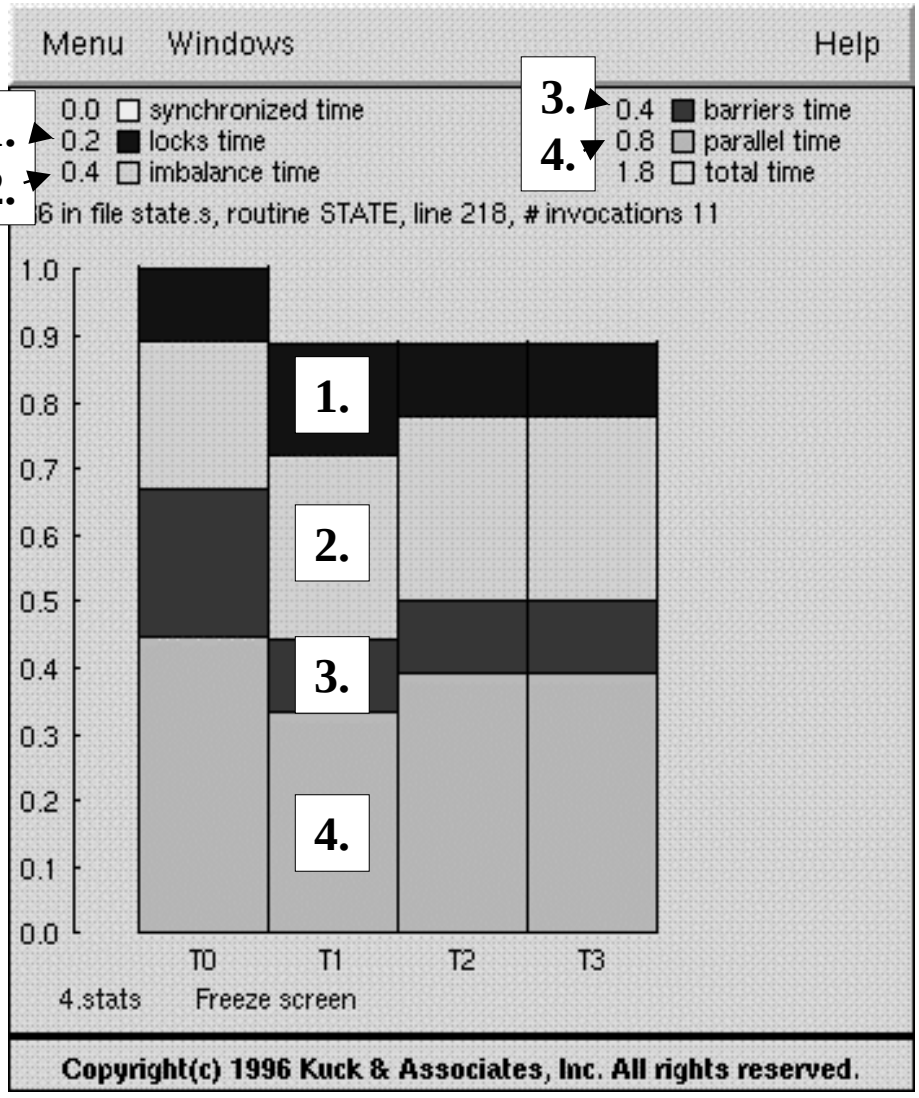
Example -- GuideView

- Parallel regions
- Sorted by
  - Overhead
  - Time
- Compare
  - Parallel time
  - Synched time
  - Lock time
  - Barrier time



Example -- GuideView

- Per-region, per-thread view



KAI
OPTIMIZING SOFTWARE

Assure as a Parallelization Tool

- Use when automatic parallelization yields insufficient parallelism, performance
- Find outer loop known to be “parallel” (yet cannot be parallelized automatically)
- Annotate loop with OpenMP directives
- Use Assure to detect, correct errors
- http://www.kai.com/kpts/pres_intro.html

KAI
OPTIMIZING SOFTWARE

Questions?

Contact me at KAI:

poulsen@kai.com

http://www.kai.com