OpenSPARC Slide-Cast

In Twelve Chapters
Presented by OpenSPARC designers, developers, and programmers
• to guide users as they develop their own OpenSPARC designs and
• to assist professors as they teach the next generation

Chapter Ten

DEVELOPING APPLICATIONS FOR CMT PROCESSORS

Darryl Gove
Performance Analyst
Author "Solaris Application Programming"
Sun Microsystems

Agenda

- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- System utilisation
- Other resources

Agenda

- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- System utilisation
- Other resources

Sun Studio 12

100% standards compliant

- ANSI C/C++
- C99, IEEE-754 ...
- OpenMP 2.5
- **GCC compatibility with gccfss**

Rapid debugging

- Set breakpoints
- Single-step
- NetBeans IDE
- **Thread analyzer**

Performance tuning

- Advanced optimizations
- Sun Performance Analyzer
- SPOT
- **Automatic Tuning System**

Extensive libraries

- Media and graphics
- Science and math
- Portable performance
- **Parallelized for CMT**

Sun Studio 12

- IDE (based on NetBeans.org)
- Compilers
  - C/C++/Fortran
- Debugger
  - dbx
- Performance Analyzer
- Thread Analyzer
- Solaris and Linux
- SPARC and x86/64

GCC for Sun Systems

- Enables GCC to use Sun Studio code generator
  - > GCC compatibility
  - > Sun Studio optimisations
  - > Compatibility with Sun Studio tools

Mapping Tools to the Development or Migration Lifecycle

- **Application Selection**
  - > cooltst

- **Observing**
  - > SPOT
  - > Corestat

http://cooltools.sunsource.net/ Over 15k

- **Development**
  - > GCC4SS (Compiling)
  - > ATS (tuning)
  - > BIT (instrumenting)
  - > Discover (checking)
  - > Thread Analyzer (checking)

- **Deployment**
  - > CoolTuner
  - > Cool Stack
  - > Consolidation Tool

Sun Studio Express

- Preview of next Sun Studio release
- July 2008 release includes
  - CMT Developer Tools
  - Initial support for OpenMP 3.0

CMT Developer Tools

- Automatic Tuning and Troubleshooting System (`ats`)
- Binary Improvement Tool (`bit`)
- Sun Memory Error Discovery Tool (`discover`)
- Simple Performance Optimisation Tool (`spot`)
- Free download from: http://cooltools.sunsource.net/

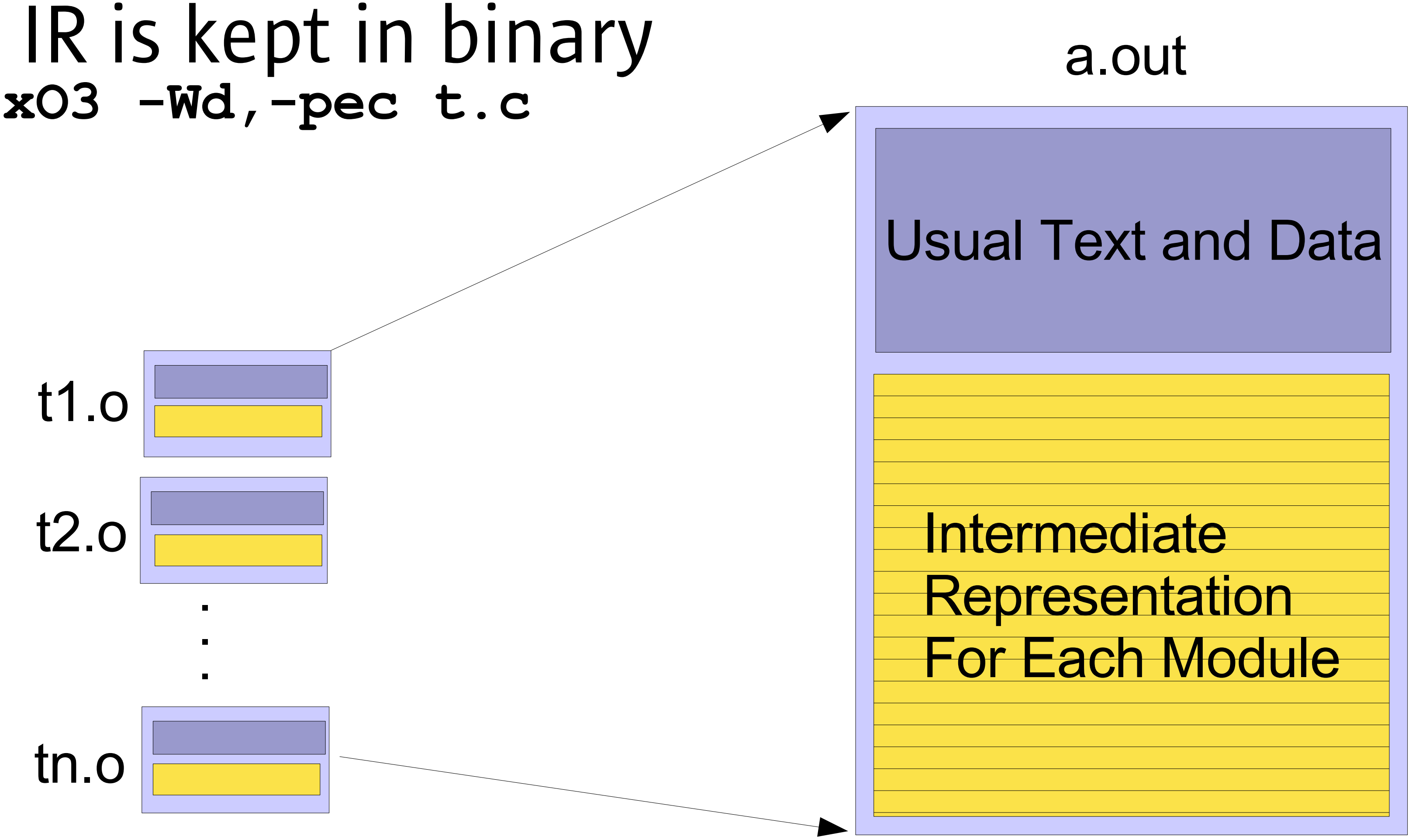
ATS

- Recompile application without access to source
- Automated performance tuning
  - Find the best compiler flags
- Automated application debug
  - Find problem compiler flag
  - Find problem module
- SPARC & x86
- http://cooltools.sunsource.net/ats/

ATS uses PEC

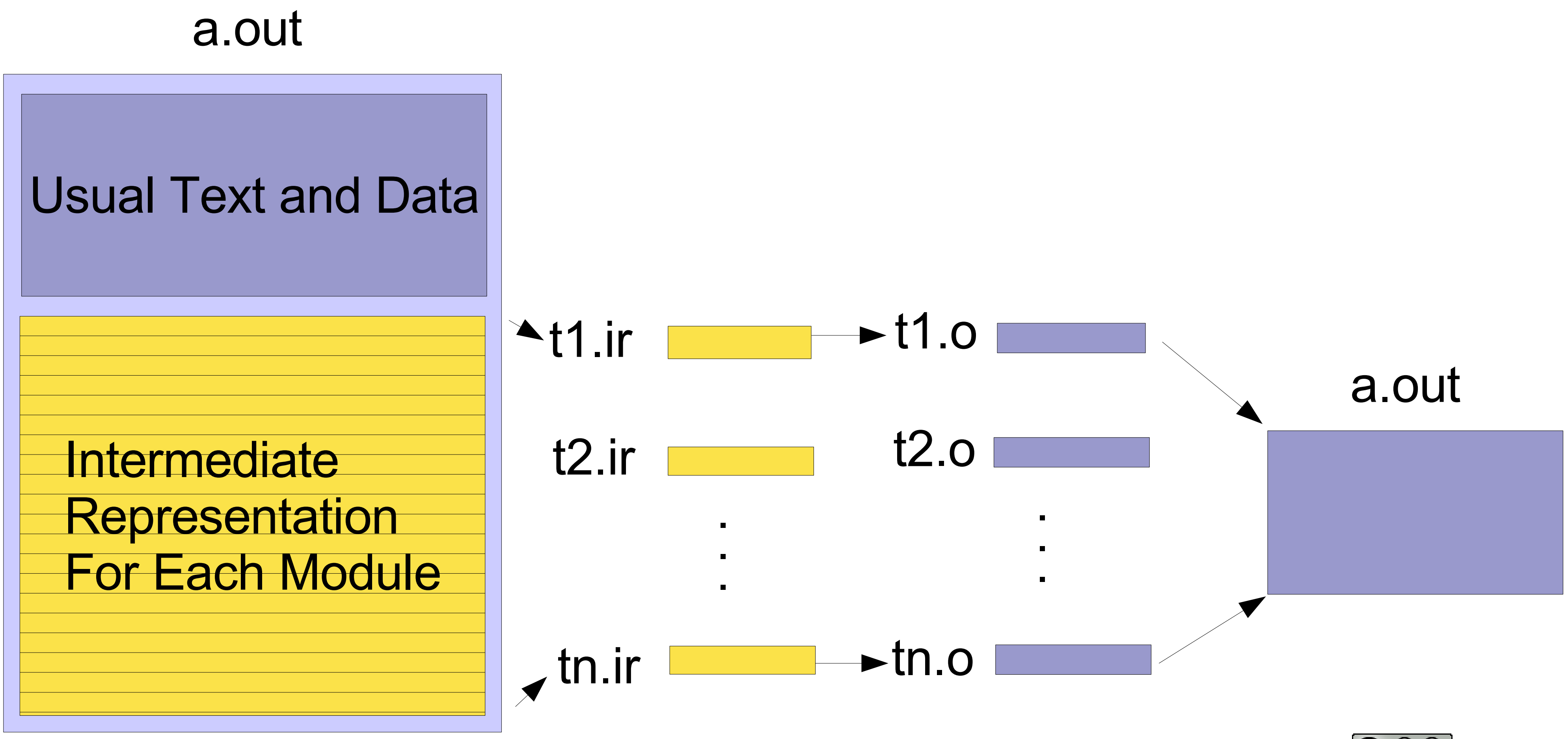
- Portable Executable Code
  - Sun IR is kept in binary

```
cc -xO3 -Wd,-pec t.c
```

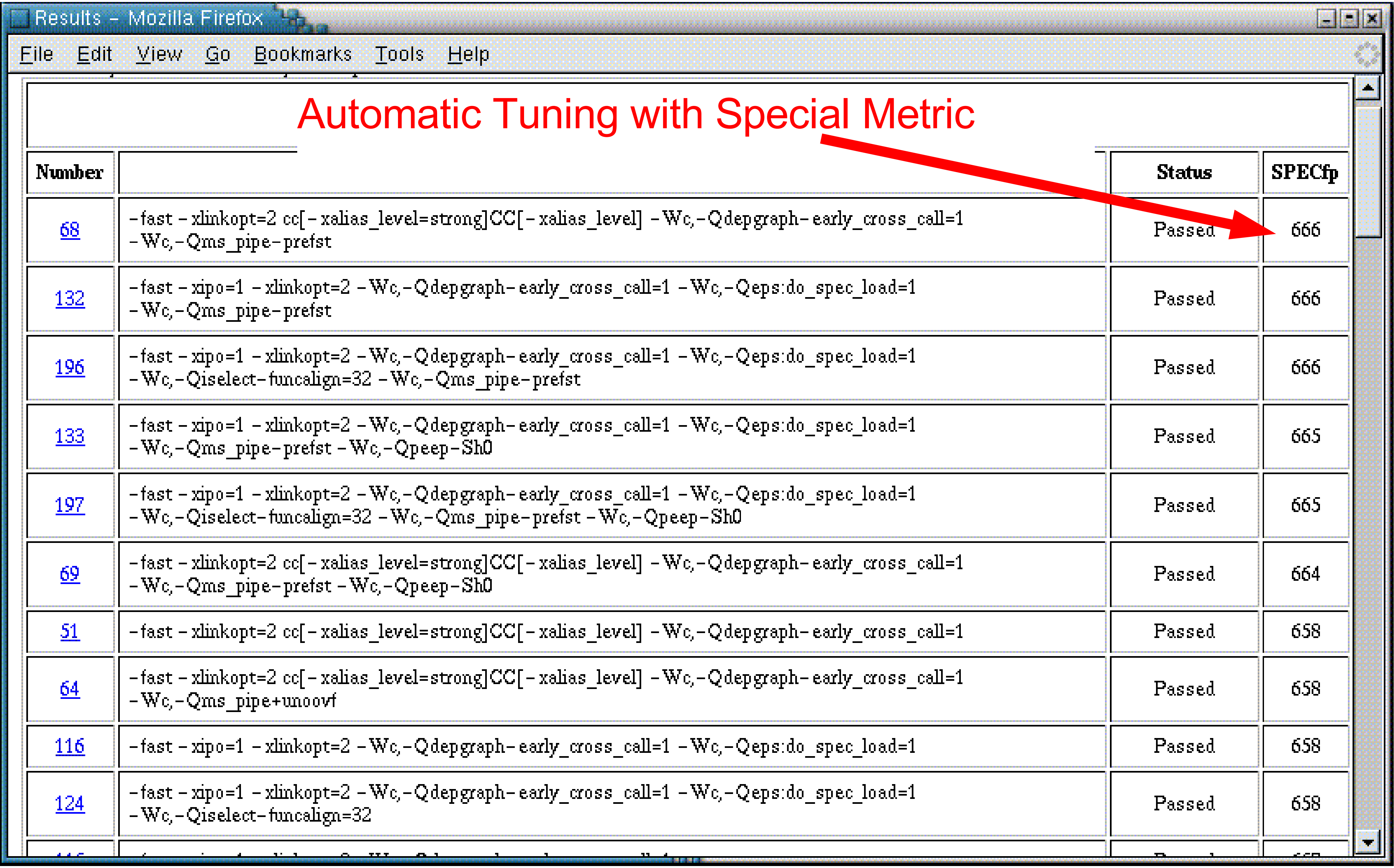


Recompiling Binaries

- IR is extracted and reprocessed



Automatic Tuning with Special Metric

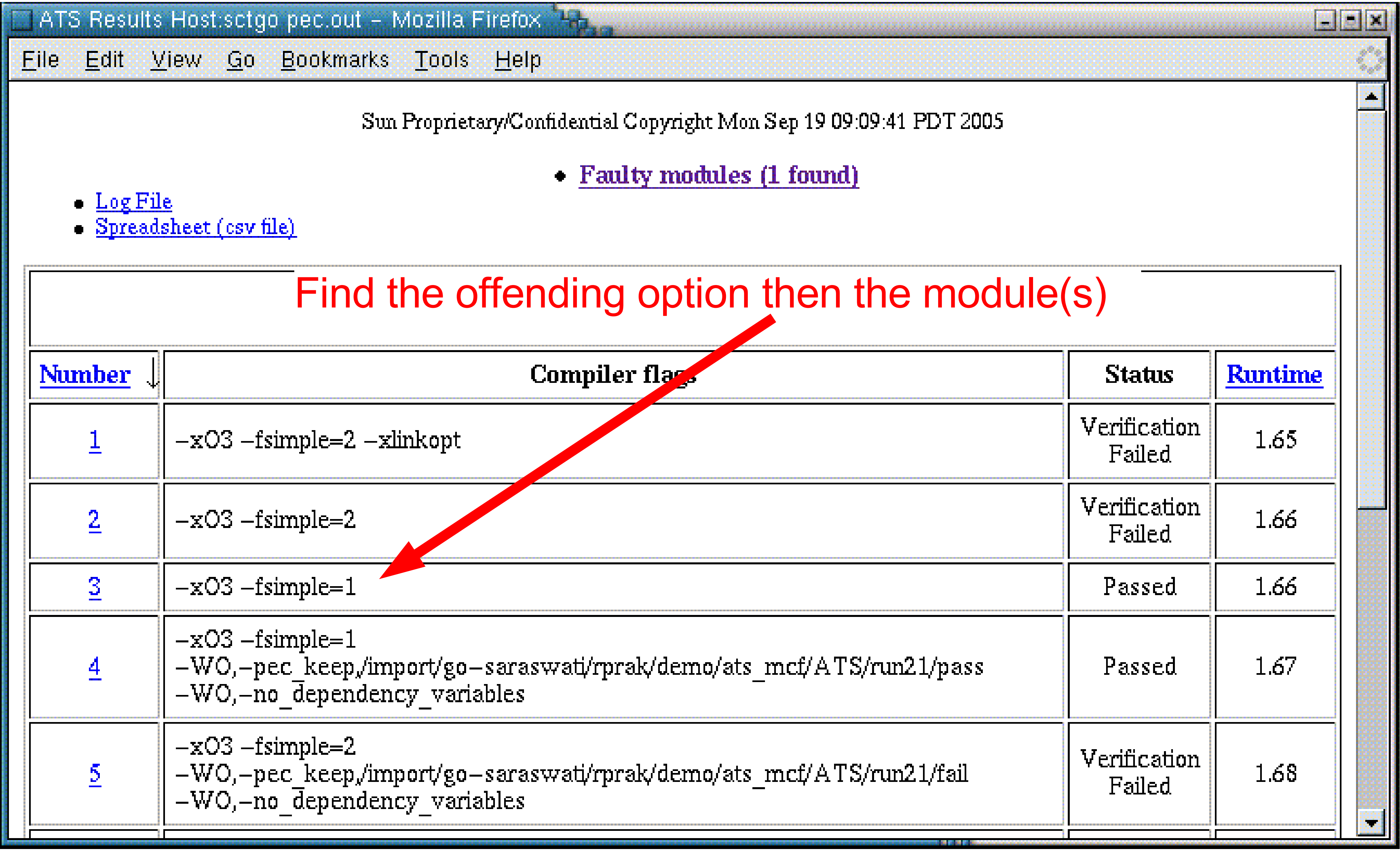

Results - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

Number		Status	SPECfp
68	-fast -xlinkopt=2 cc[-xalias_level=strong]CC[-xalias_level] -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qms_pipe-prefst	Passed	666
132	-fast -xipo=1 -xlinkopt=2 -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qeps:do_spec_load=1 -Wc,-Qms_pipe-prefst	Passed	666
196	-fast -xipo=1 -xlinkopt=2 -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qeps:do_spec_load=1 -Wc,-Qiselect-funcalign=32 -Wc,-Qms_pipe-prefst	Passed	666
133	-fast -xipo=1 -xlinkopt=2 -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qeps:do_spec_load=1 -Wc,-Qms_pipe-prefst -Wc,-Qpeep-Sh0	Passed	665
197	-fast -xipo=1 -xlinkopt=2 -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qeps:do_spec_load=1 -Wc,-Qiselect-funcalign=32 -Wc,-Qms_pipe-prefst -Wc,-Qpeep-Sh0	Passed	665
69	-fast -xlinkopt=2 cc[-xalias_level=strong]CC[-xalias_level] -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qms_pipe-prefst -Wc,-Qpeep-Sh0	Passed	664
51	-fast -xlinkopt=2 cc[-xalias_level=strong]CC[-xalias_level] -Wc,-Qdepgraph-early_cross_call=1	Passed	658
64	-fast -xlinkopt=2 cc[-xalias_level=strong]CC[-xalias_level] -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qms_pipe+unoovf	Passed	658
116	-fast -xipo=1 -xlinkopt=2 -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qeps:do_spec_load=1	Passed	658
124	-fast -xipo=1 -xlinkopt=2 -Wc,-Qdepgraph-early_cross_call=1 -Wc,-Qeps:do_spec_load=1 -Wc,-Qiselect-funcalign=32	Passed	658

Find bug

- Locate problem flags and problem module

```
% ats -i 'script:findbug -xO3 -fsimple=2 -xlinkopt'
      a.out
```


ATS Results Host:sctgo pec.out - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

Sun Proprietary/Confidential Copyright Mon Sep 19 09:09:41 PDT 2005

- Faulty modules (1 found)

- Log File
- Spreadsheet (csv file)

Find the offending option then the module(s)

Number ↓	Compiler flags	Status	Runtime
1	–xO3 –fsimple=2 –xlinkopt	Verification Failed	1.65
2	–xO3 –fsimple=2	Verification Failed	1.66
3	–xO3 –fsimple=1	Passed	1.66
4	–xO3 –fsimple=1 –WO,–pec_keep,/import/go–saraswati/rprak/demo/ats_mcf/ATS/run21/pass –WO,–no_dependency_variables	Passed	1.67
5	–xO3 –fsimple=2 –WO,–pec_keep,/import/go–saraswati/rprak/demo/ats_mcf/ATS/run21/fail –WO,–no_dependency_variables	Verification Failed	1.68

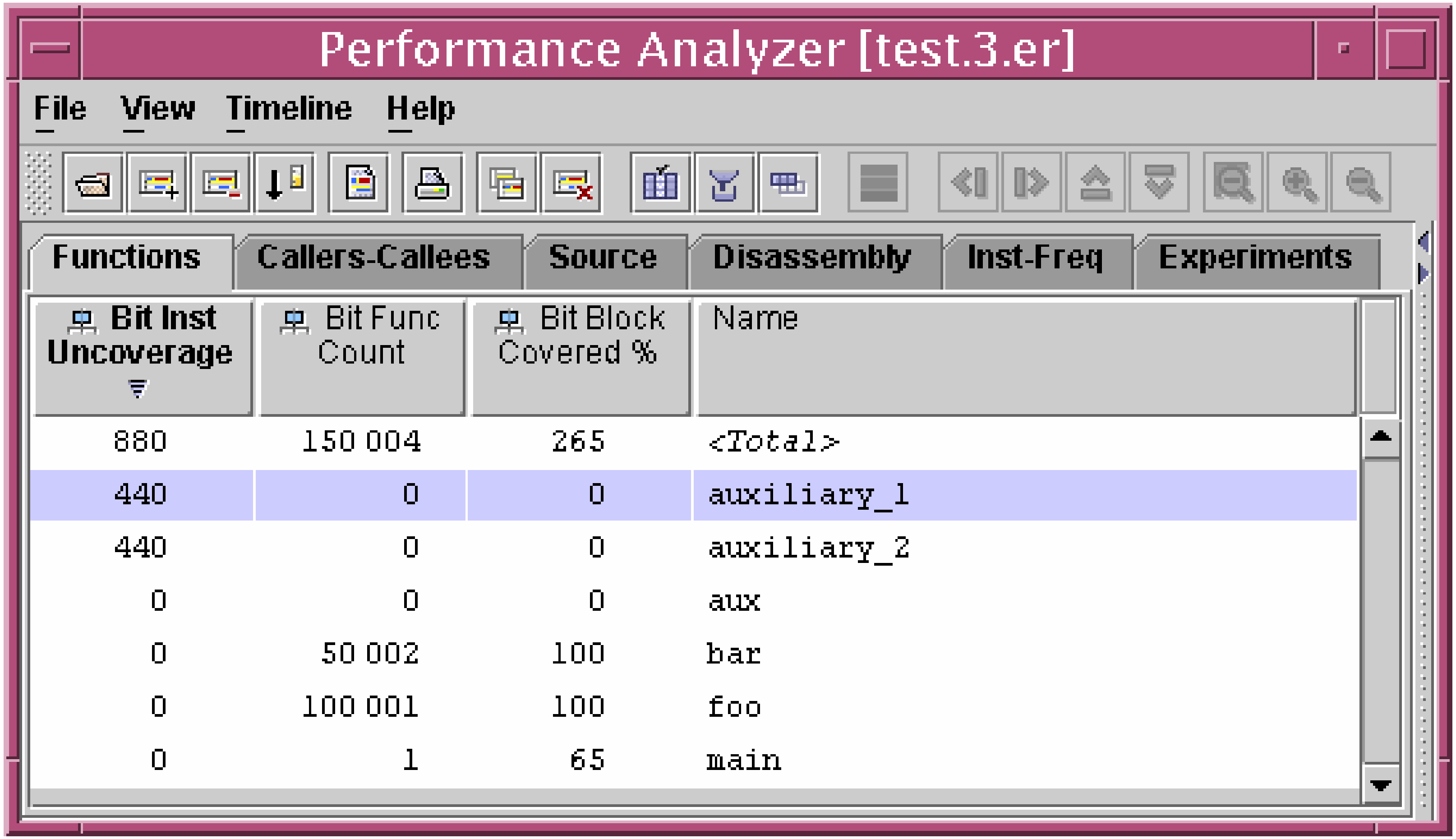
BIT

- Gathers runtime information
  - Instruction execution count
  - Branch taken probabilities
    - Compare behaviour of different workloads
- Generates coverage information
- SPARC only
- http://cooltools.sunsource.net/bit/

BIT coverage results

```
bit coverage -R -d  nmBasic.t.exe
...
BIT Code Coverage
Total Functions: 179
Covered Functions: 19
Function Coverage: 10.6%
Total Basic Blocks: 775
Covered Basic Blocks: 508
Basic Block Coverage: 65.5%
Total Basic Block Executions: 1,296
Average Executions per Basic Block: 1.67
Total Instructions: 3,168
Covered Instructions: 1,719
Instruction Coverage: 54.3%
Total Instruction Executions: 6,373
Average Executions per Instruction: 2.01
Creating experiment database test.1.er
```

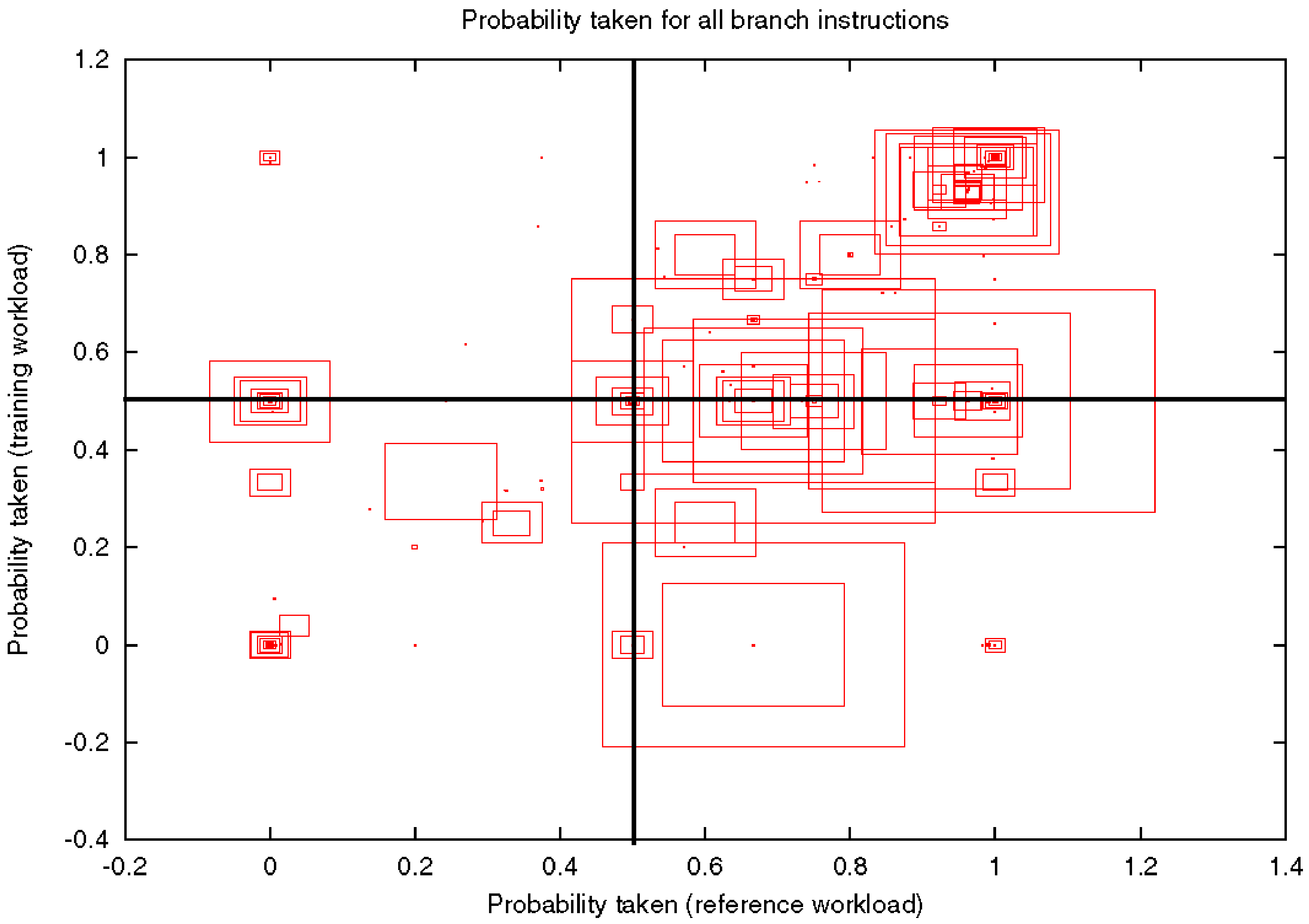
BIT Uncoverage



Performance Analyzer [test.3.er]
File View Timeline Help
Functions Callers-Callees Source Disassembly Inst-Freq Experiments

Bit Inst Uncoverage	Bit Func Count	Bit Block Covered %	Name
880	150 004	265	<Total>
440	0	0	auxiliary_1
440	0	0	auxiliary_2
0	0	0	aux
0	50 002	100	bar
0	100 001	100	foo
0	1	65	main

Branch probability analysis



http://www.spec.org/workshops/2006/papers/10_Darryl_Gove.pdf

Discover

- Memory access error detection
  - Write past end of array
  - Read of uninitialised data
  - Use of freed memory
- SPARC only
- http://cooltools.sunsource.net/discover/

Discover - example

```
$ more memerr.c
#include<stdlib.h>
void main()
{
  int* a;
  int i;
  a=(int*)malloc(sizeof(int)*5);
  for (i=0; i<6; i++)
  {
   a[i]=0;
  }
}

$ cc -O -xbinopt=prepare memerr.c

% discover a.out
```

```
% a.out

ERROR (ABW):
writing to memory beyond array
bounds at:

        main() + 0x158
        _start() + 0x108

block  was allocated at:

        malloc() + 0x144
        main() + 0x1c
        _start() + 0x108

DISCOVER SUMMARY:
 unique errors   : 1 (1 total)
 unique warnings : 0 (0 total)
```

Agenda

- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- System utilisation
- Other resources

Optimisation flags

- No optimisation flags = no optimisation
- **`-O`** = good degree of optimisation
- **`-fast`** = aggressive optimisation

Debug flags

- `-g` for C/Fortran
- `-g0` for C++
  - `-g` disables front-end inlining in C++
- Minor changes to code at low optimisation
  - Tail call optimisation
- Allows attribution of time to lines of source

32-bit or 64-bit

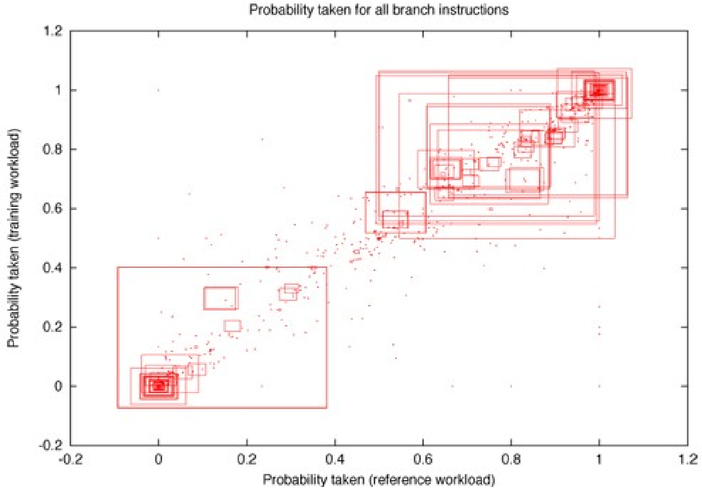
- Compiler flags: **`-m32 | -m64`**
- 64-bit:
  - > Larger address space
  - > Pointers and longs 64-bits
  - > Larger memory footprint
  - > Potentially lower performance

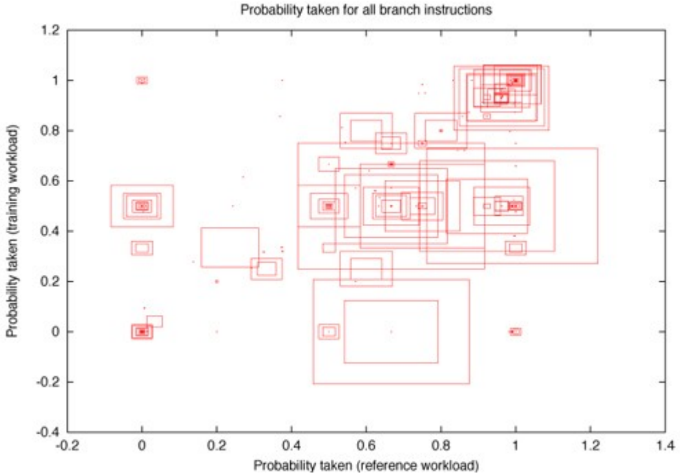
Inlining

- Inlining
  - Within file **`-xO4`**
  - Across files **`-xipo`**
- Avoid cost of calling routine
- Expose further performance opportunities

Profile feedback

- Two compile passes (complicates build)
- Good for “branchy” code
- Helps inlining decisions
- Profile feedback
  - > `-xprofile=[collect:|use:]`





http://developers.sun.com/solaris/articles/coverage.h

Target architecture

- General
  - > `-xtarget=generic`
- If build and run machine is the same
  - > `-xtarget=native`
- Flags evaluated from left to right

Aliasing

- Compiler has to assume pointers alias
  - > Unless it can prove otherwise
  - > Or it is told to assume otherwise
- Specify degree of aliasing
  - > `-xalias_level=<level>`
- Specify pointers passed into functions don't alias
  - > `-xrestrict`
- Restrict qualify pointers
  - > `int * restrict p`

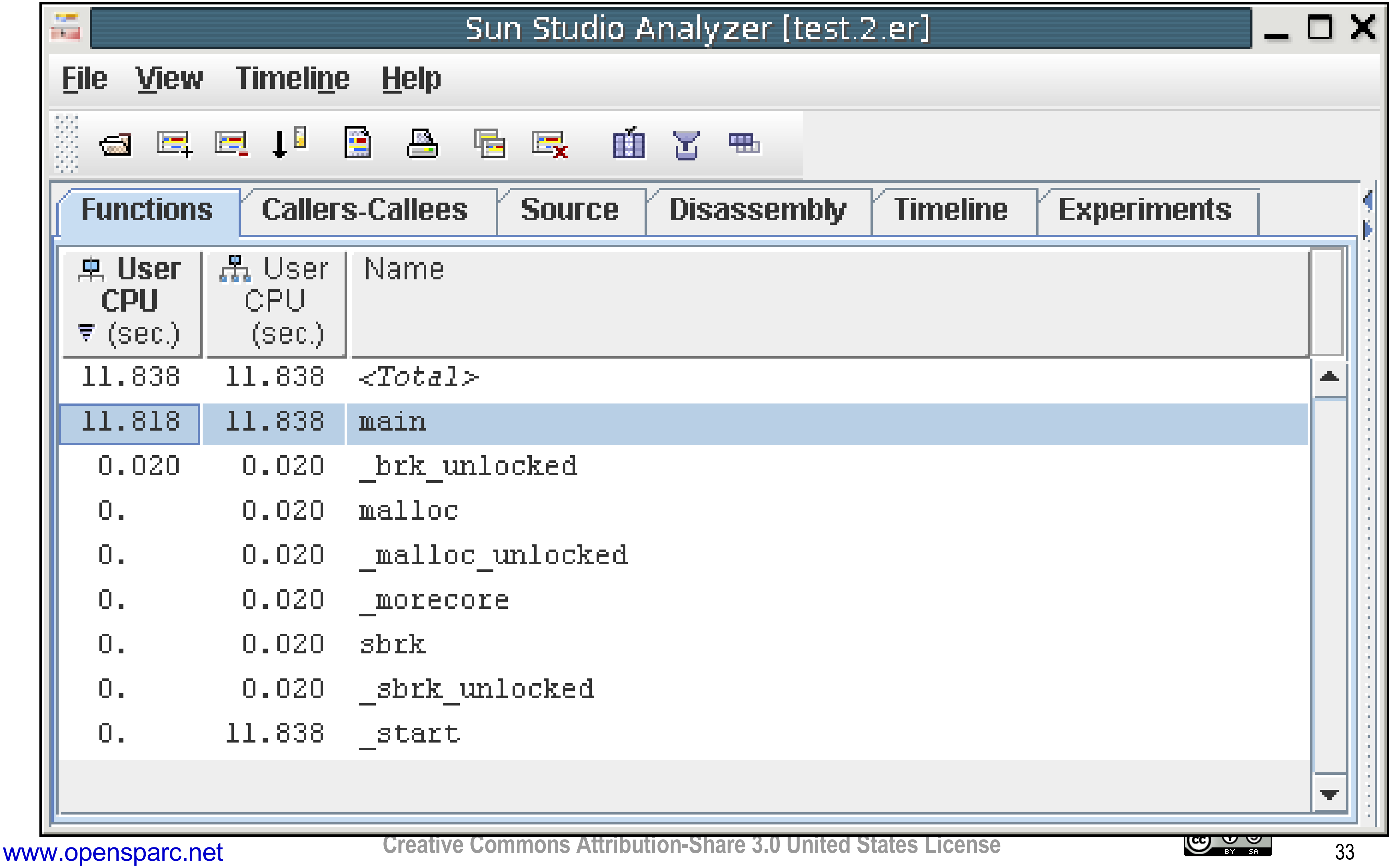
Agenda

- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- System utilisation
- Other resources

Profiling

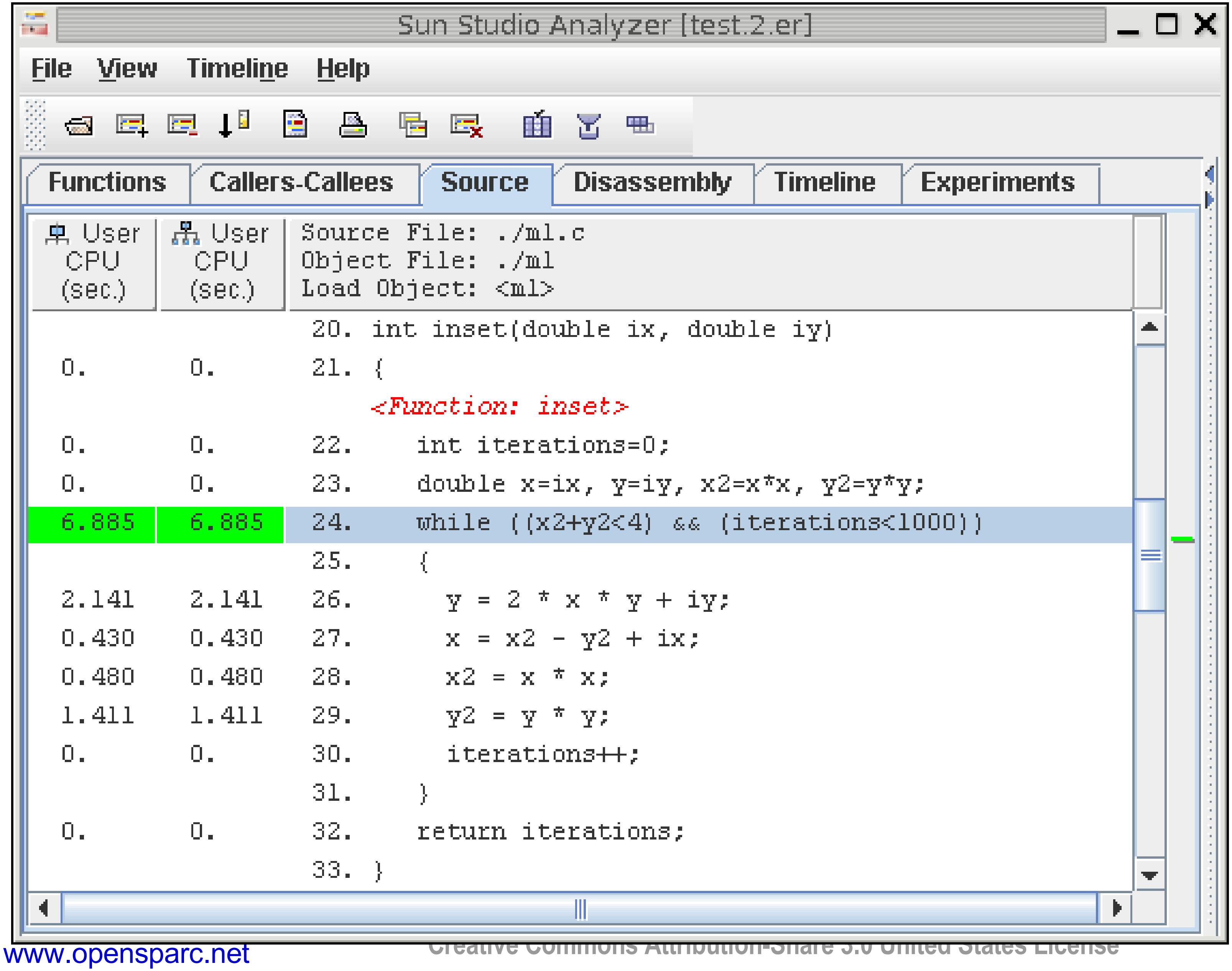
- Performance Analyzer
  - Gather
    - `collect <app> <params>`
    - `collect -P <pid>`
  - Analyse
    - `analyzer <test.N.er>`
    - `er_print <test.N.er>`
- spot
  - Generate html report
    - `spot <app> <params>`
    - `spot -P <pid>`
- Compiler flags: `-g -g0 -xbinopt=prepare`

Application profile



User CPU (sec.)	User CPU (sec.)	Name
11.838	11.838	*<Total>*
11.818	11.838	main
0.020	0.020	_brk_unlocked
0.	0.020	malloc
0.	0.020	_malloc_unlocked
0.	0.020	_morecore
0.	0.020	sbrk
0.	0.020	_sbrk_unlocked
0.	11.838	_start

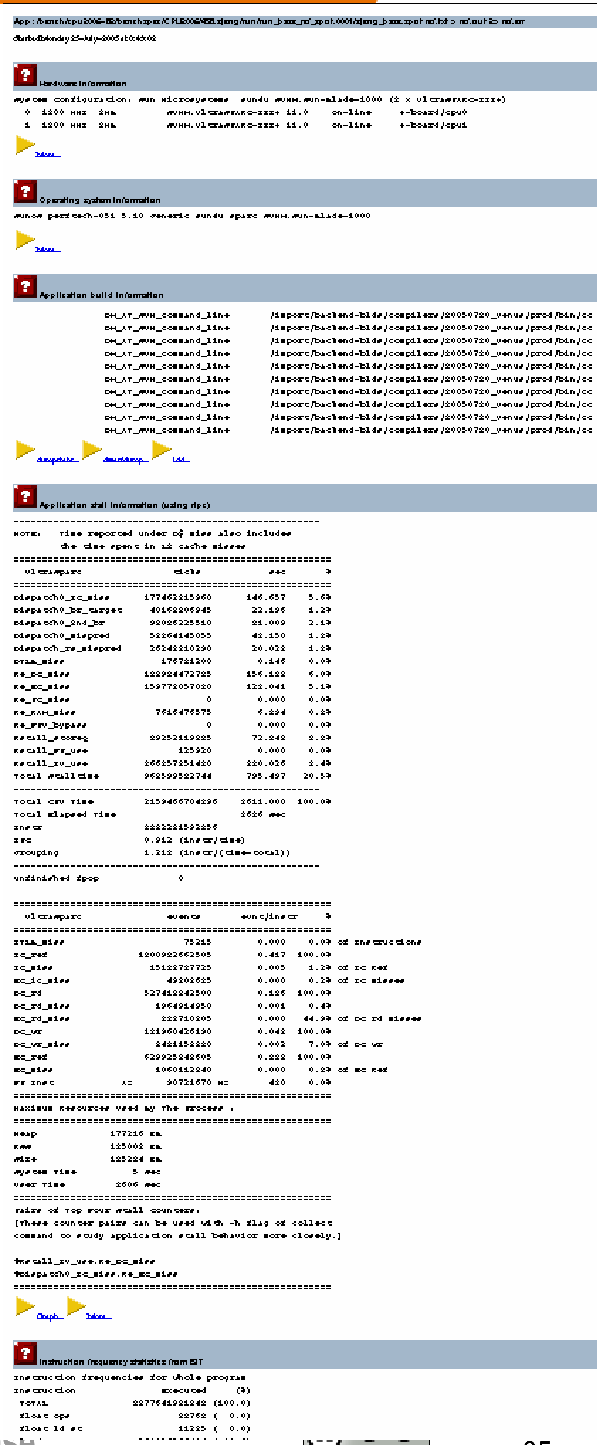
Source level profile



```
User      User     Source File: ./ml.c
CPU       CPU      Object File: ./ml
(sec.)    (sec.)   Load Object: <ml>
                    20. int inset(double ix, double iy)
0.        0.        21. {
                        <Function: inset>
0.        0.        22.    int iterations=0;
0.        0.        23.    double x=ix, y=iy, x2=x*x, y2=y*y;
6.885     6.885     24.    while ((x2+y2<4) && (iterations<1000))
                    25.    {
2.141     2.141     26.      y = 2 * x * y + iy;
0.430     0.430     27.      x = x2 - y2 + ix;
0.480     0.480     28.      x2 = x * x;
1.411     1.411     29.      y2 = y * y;
0.        0.        30.      iterations++;
                    31.    }
0.        0.        32.    return iterations;
                    33. }
```

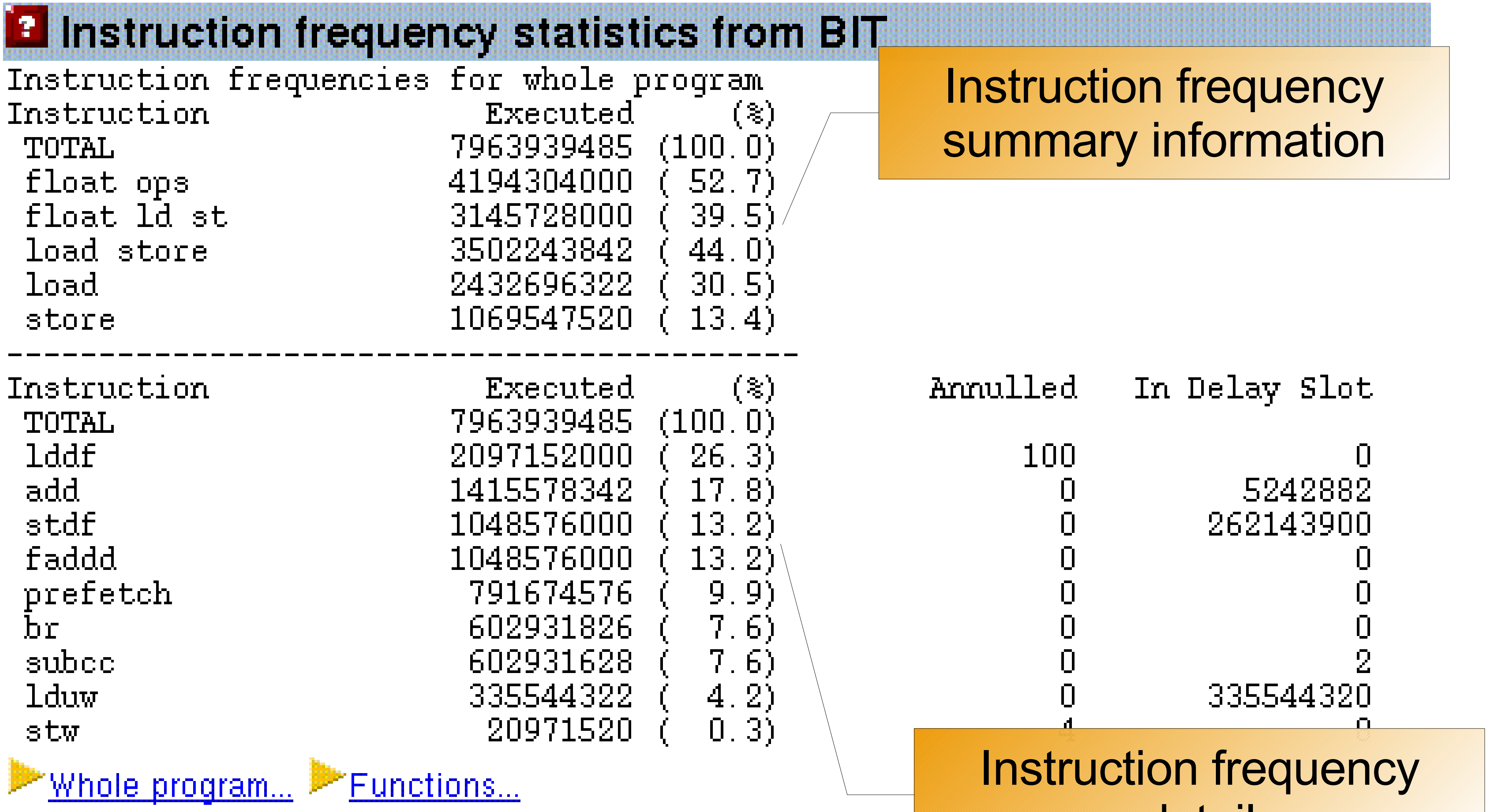
spot

- Generates detailed html report on application performance
  - > Performance counter events
  - > Time-based profile
  - > Event-based profile
  - > Bandwidth utilisation
  - > ...
- Helps in remote collaboration
- SPARC & x86
- http://cooltools.sunsource.net/spot/



```
App : /bench/cpu2006-[illegible]/benchspec/CPU2006/[illegible]/run/run_base_ref_spot.0001/[illegible]_base.spot ref.txt > ref.out 2> ref.err
Started Monday 25-July-2005 at [illegible]

Hardware Information
System Configuration: Sun Microsystems  sun4u SUNW,Sun-Blade-1000 (2 X UltraSPARC-III+)
   0  1200 MHz  8MB        SUNW,UltraSPARC-III+ 11.0    on-line     +-board/cpu0
   1  1200 MHz  8MB        SUNW,UltraSPARC-III+ 11.0    on-line     +-board/cpu1
More...

Operating system information
SunOS perftech-051 5.10 Generic sun4u sparc SUNW,Sun-Blade-1000
More...

Application build information
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
              CH_AT_SUN_command_line      /import/backend-bld/compilers/20050720_venus/prod/bin/cc
dumpstabs   dwarfdump   ldd

Application stall information (using ripc)
-------------------------------------------------------------
NOTE:    Time reported under DC miss also includes
         the time spent in L2 cache misses
===============================================================
  UltraSPARC                  cycles            sec       %
===============================================================
Dispatch0_IC_miss        177462215960       146.657    5.6%
Dispatch0_br_target       40162206945        22.196    1.2%
Dispatch0_2nd_br          92026225510        21.009    2.1%
Dispatch0_mispred         52264145055        42.150    1.2%
Dispatch_rs_mispred       26242210290        20.022    1.2%
ITLB_miss                   176721200         0.146    0.0%
Re_DC_miss               122924472725       156.122    6.0%
Re_EC_miss               159772057020       122.041    5.1%
Re_PC_miss                          0         0.000    0.0%
Re_RAW_miss                7616476575         6.294    0.2%
Re_FPU_bypass                       0         0.000    0.0%
Rstall_storeQ             29252119225        72.242    2.2%
Rstall_FP_use                  125920         0.000    0.0%
Rstall_IU_use            266257251420       220.026    2.4%
Total Stalltime          962599522744       795.497   20.5%
-------------------------------------------------------------
Total CPU Time           2159466704296     2611.000  100.0%
Total Elapsed Time                         2626 sec
Instr                    2222221592256
IPC                      0.912  (instr/time)
Grouping                 1.212  (instr/(time-total))
-------------------------------------------------------------
Unfinished FPop                 0


===============================================================
  UltraSPARC                  events       events/instr     %
===============================================================
ITLB_miss                       75215         0.000    0.0% of instructions
IC_ref                  1200922662505         0.417  100.0%
IC_miss                   15122727725         0.005    1.2% of IC ref
EC_ic_miss                   49202625         0.000    0.2% of IC misses
DC_rd                    527412242500         0.126  100.0%
DC_rd_miss                 1964914950         0.001    0.4%
EC_rd_miss                  222710205         0.000   44.9% of DC rd misses
DC_wr                    121960426190         0.042  100.0%
DC_wr_miss                 2421152220         0.002    7.0% of DC wr
EC_ref                   629925242605         0.222  100.0%
EC_miss                    1060112240         0.000    0.2% of EC ref
FP Inst             AZ       90721670 HZ        420    0.0%
===============================================================
Maximum Resources used by The process :
===============================================================
Heap              177216 KB
RSS               125002 KB
Size              125224 KB
System Time            5 sec
User Time           2606 sec
===============================================================
Pairs of Top four stall counters:
[These counter pairs can be used with -h flag of collect
command to study application stall behavior more closely.]

#Rstall_IU_use,Re_DC_miss
#Dispatch0_IC_miss,Re_EC_miss
===============================================================
Graph   More...

Instruction frequency statistics from BIT
Instruction frequencies for whole program
Instruction                       executed      (%)
 TOTAL                       2277641921242  (100.0)
 float ops                           22762  (  0.0)
 float ld st                         11225  (  0.0)
```

Instruction Frequency (from BIT)



Instruction frequency statistics from BIT

```
Instruction frequencies for whole program
Instruction                   Executed      (%)
 TOTAL                      7963939485  (100.0)
 float ops                  4194304000  ( 52.7)
 float ld st                3145728000  ( 39.5)
 load store                 3502243842  ( 44.0)
 load                       2432696322  ( 30.5)
 store                      1069547520  ( 13.4)
-------------------------------------------
Instruction                   Executed      (%)       Annulled   In Delay Slot
 TOTAL                      7963939485  (100.0)
 lddf                       2097152000  ( 26.3)            100               0
 add                        1415578342  ( 17.8)              0         5242882
 stdf                       1048576000  ( 13.2)              0       262143900
 faddd                      1048576000  ( 13.2)              0               0
 prefetch                    791674576  (  9.9)              0               0
 br                          602931826  (  7.6)              0               0
 subcc                       602931628  (  7.6)              0               2
 lduw                        335544322  (  4.2)              0       335544320
 stw                          20971520  (  0.3)              4               0
```

Instruction frequency summary information

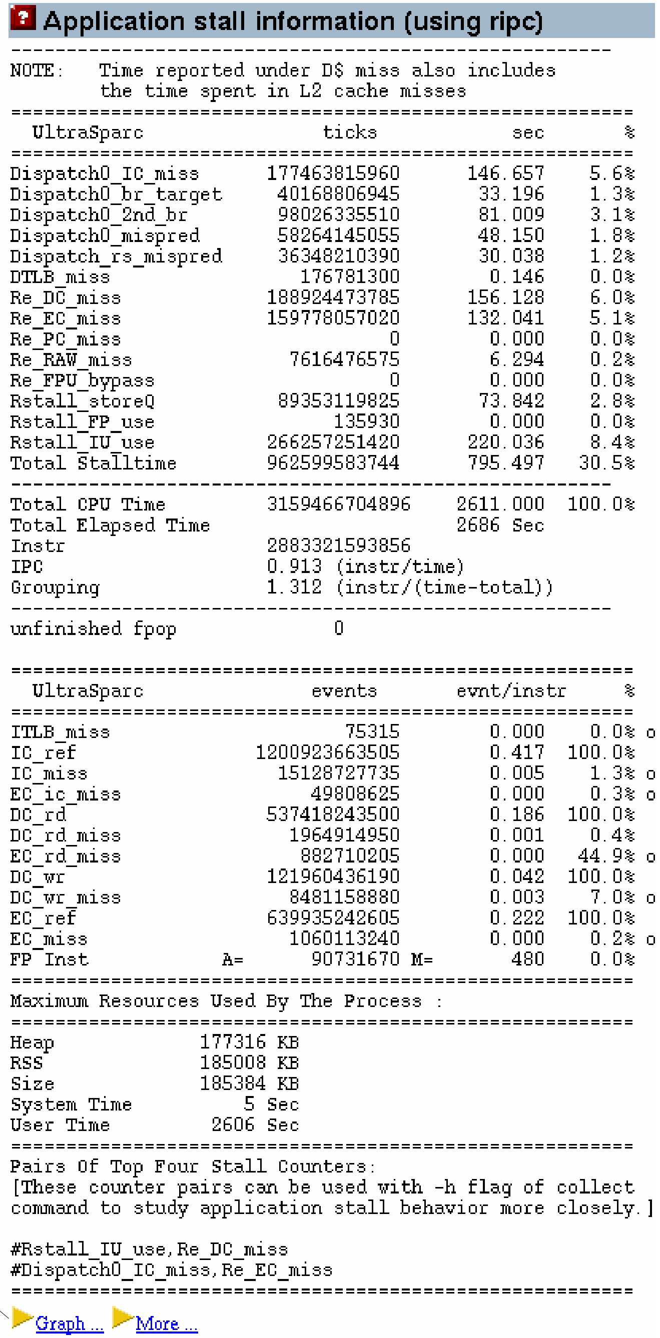
Instruction frequency detail

Performance Counters

Time lost due to various processor stall conditions

Memory consumption & system time

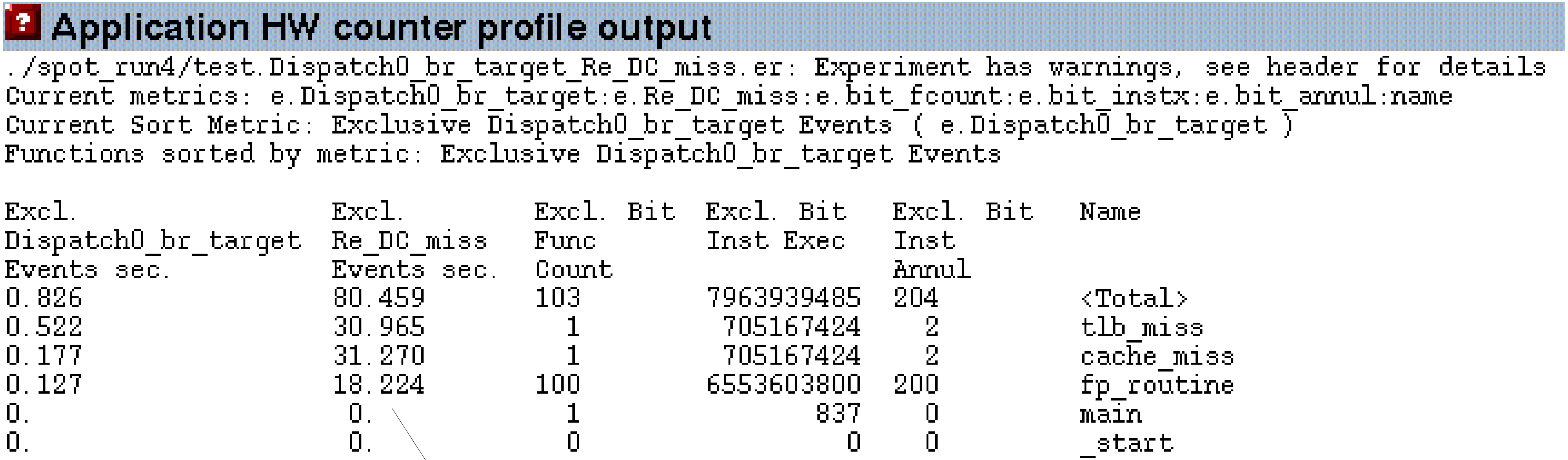
Graph of events over time



Application stall information (using ripc)

```
------------------------------------------------------
NOTE:    Time reported under D$ miss also includes
         the time spent in L2 cache misses
========================================================
  UltraSparc                ticks              sec       %
========================================================
Dispatch0_IC_miss        177463815960      146.657    5.6%
Dispatch0_br_target       40168806945       33.196    1.3%
Dispatch0_2nd_br          98026335510       81.009    3.1%
Dispatch0_mispred         58264145055       48.150    1.8%
Dispatch_rs_mispred       36348210390       30.038    1.2%
DTLB_miss                   176781300        0.146    0.0%
Re_DC_miss               188924473785      156.128    6.0%
Re_EC_miss               159778057020      132.041    5.1%
Re_PC_miss                          0        0.000    0.0%
Re_RAW_miss                7616476575        6.294    0.2%
Re_FPU_bypass                       0        0.000    0.0%
Rstall_storeQ             89353119825       73.842    2.8%
Rstall_FP_use                  135930        0.000    0.0%
Rstall_IU_use            266257251420      220.036    8.4%
Total Stalltime          962599583744      795.497   30.5%
------------------------------------------------------
Total CPU Time           3159466704896    2611.000  100.0%
Total Elapsed Time                        2686 Sec
Instr                    2883321593856
IPC                      0.913 (instr/time)
Grouping                 1.312 (instr/(time-total))
------------------------------------------------------
unfinished fpop                   0

========================================================
  UltraSparc                events         evnt/instr    %
========================================================
ITLB_miss                        75315        0.000    0.0% o
IC_ref                   1200923663505        0.417  100.0%
IC_miss                    15128727735        0.005    1.3% o
EC_ic_miss                    49808625        0.000    0.3% o
DC_rd                     537418243500        0.186  100.0%
DC_rd_miss                  1964914950        0.001    0.4%
EC_rd_miss                   882710205        0.000   44.9% o
DC_wr                     121960436190        0.042  100.0%
DC_wr_miss                  8481158880        0.003    7.0% o
EC_ref                    639935242605        0.222  100.0%
EC_miss                     1060113240        0.000    0.2% o
FP Inst               A=      90731670 M=       480    0.0%
========================================================
Maximum Resources Used By The Process :
========================================================
Heap                 177316 KB
RSS                  185008 KB
Size                 185384 KB
System Time               5 Sec
User Time              2606 Sec
========================================================
Pairs Of Top Four Stall Counters:
[These counter pairs can be used with -h flag of collect
command to study application stall behavior more closely.]

#Rstall_IU_use,Re_DC_miss
#Dispatch0_IC_miss,Re_EC_miss
========================================================
```

Profile - hardware events (–x flag)

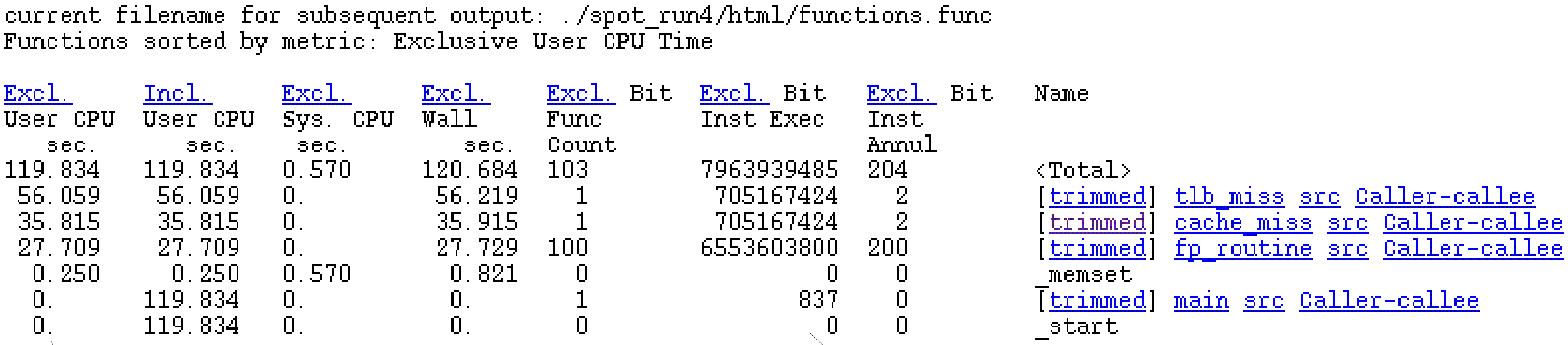


Application HW counter profile output

```
./spot_run4/test.Dispatch0_br_target_Re_DC_miss.er: Experiment has warnings, see header for details
Current metrics: e.Dispatch0_br_target:e.Re_DC_miss:e.bit_fcount:e.bit_instx:e.bit_annul:name
Current Sort Metric: Exclusive Dispatch0_br_target Events ( e.Dispatch0_br_target )
Functions sorted by metric: Exclusive Dispatch0_br_target Events

Excl.                    Excl.          Excl. Bit   Excl. Bit    Excl. Bit    Name
Dispatch0_br_target      Re_DC_miss     Func        Inst Exec    Inst
Events sec.              Events sec.    Count                    Annul
0.826                    80.459         103         7963939485   204          <Total>
0.522                    30.965           1          705167424     2          tlb_miss
0.177                    31.270           1          705167424     2          cache_miss
0.127                    18.224         100         6553603800   200          fp_routine
0.                        0.              1                837     0          main
0.                        0.              0                  0     0          _start
```

Time lost to Data Cache miss events

Profile - time



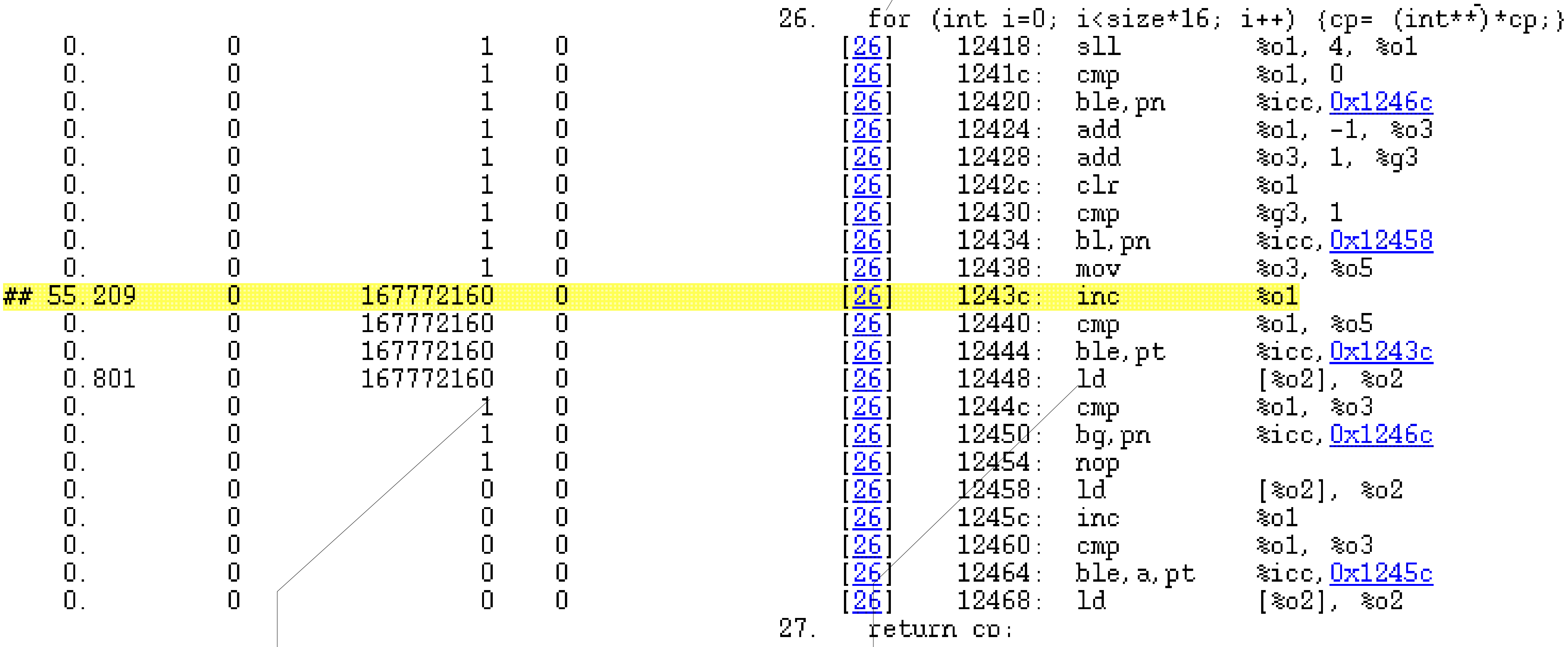
```
current filename for subsequent output: ./spot_run4/html/functions.func
Functions sorted by metric: Exclusive User CPU Time

Excl.     Incl.     Excl.     Excl.     Excl. Bit  Excl. Bit   Excl. Bit   Name
User CPU  User CPU  Sys. CPU  Wall      Func       Inst Exec   Inst
   sec.      sec.     sec.       sec.   Count                  Annul
119.834   119.834   0.570     120.684   103        7963939485  204        <Total>
 56.059    56.059   0.         56.219     1         705167424    2        [trimmed] tlb_miss src Caller-callee
 35.815    35.815   0.         35.915     1         705167424    2        [trimmed] cache_miss src Caller-callee
 27.709    27.709   0.         27.729   100        6553603800  200        [trimmed] fp_routine src Caller-callee
  0.250     0.250   0.570       0.821     0                 0    0        _memset
  0.      119.834   0.          0.        1               837    0        [trimmed] main src Caller-callee
  0.      119.834   0.          0.        0                 0    0        _start
```

Instructions executed in each routine

Number of times routine was called

Time spent in each routine

Assembly level profile



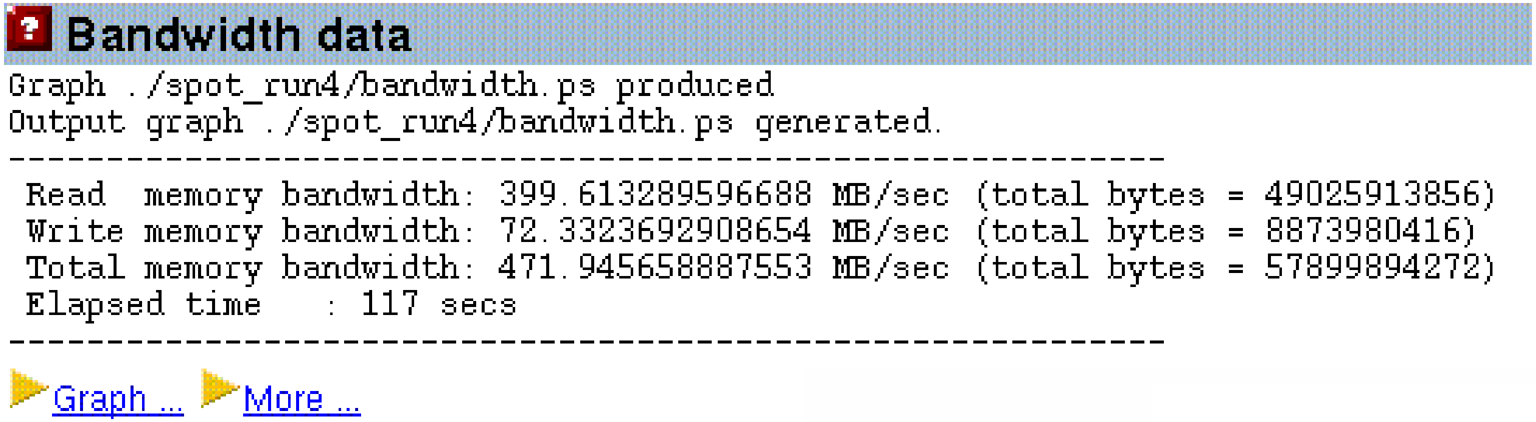
```
                                                   26.   for (int i=0; i<size*16; i++) {cp= (int**)*cp;}
   0.         0                1    0                    [26]    12418:  sll          %o1, 4, %o1
   0.         0                1    0                    [26]    1241c:  cmp          %o1, 0
   0.         0                1    0                    [26]    12420:  ble,pn       %icc,0x1246c
   0.         0                1    0                    [26]    12424:  add          %o1, -1, %o3
   0.         0                1    0                    [26]    12428:  add          %o3, 1, %g3
   0.         0                1    0                    [26]    1242c:  clr          %o1
   0.         0                1    0                    [26]    12430:  cmp          %g3, 1
   0.         0                1    0                    [26]    12434:  bl,pn        %icc,0x12458
   0.         0                1    0                    [26]    12438:  mov          %o3, %o5
## 55.209     0        167772160    0                    [26]    1243c:  inc          %o1
   0.         0        167772160    0                    [26]    12440:  cmp          %o1, %o5
   0.         0        167772160    0                    [26]    12444:  ble,pt       %icc,0x1243c
   0.801      0        167772160    0                    [26]    12448:  ld           [%o2], %o2
   0.         0                1    0                    [26]    1244c:  cmp          %o1, %o3
   0.         0                1    0                    [26]    12450:  bg,pn        %icc,0x1246c
   0.         0                1    0                    [26]    12454:  nop
   0.         0                0    0                    [26]    12458:  ld           [%o2], %o2
   0.         0                0    0                    [26]    1245c:  inc          %o1
   0.         0                0    0                    [26]    12460:  cmp          %o1, %o3
   0.         0                0    0                    [26]    12464:  ble,a,pt     %icc,0x1245c
   0.         0                0    0                    [26]    12468:  ld           [%o2], %o2
                                                   27.   return cp;
```

Source code for loop

Loop entered once, trip count = ~170M

Load instruction that takes the time

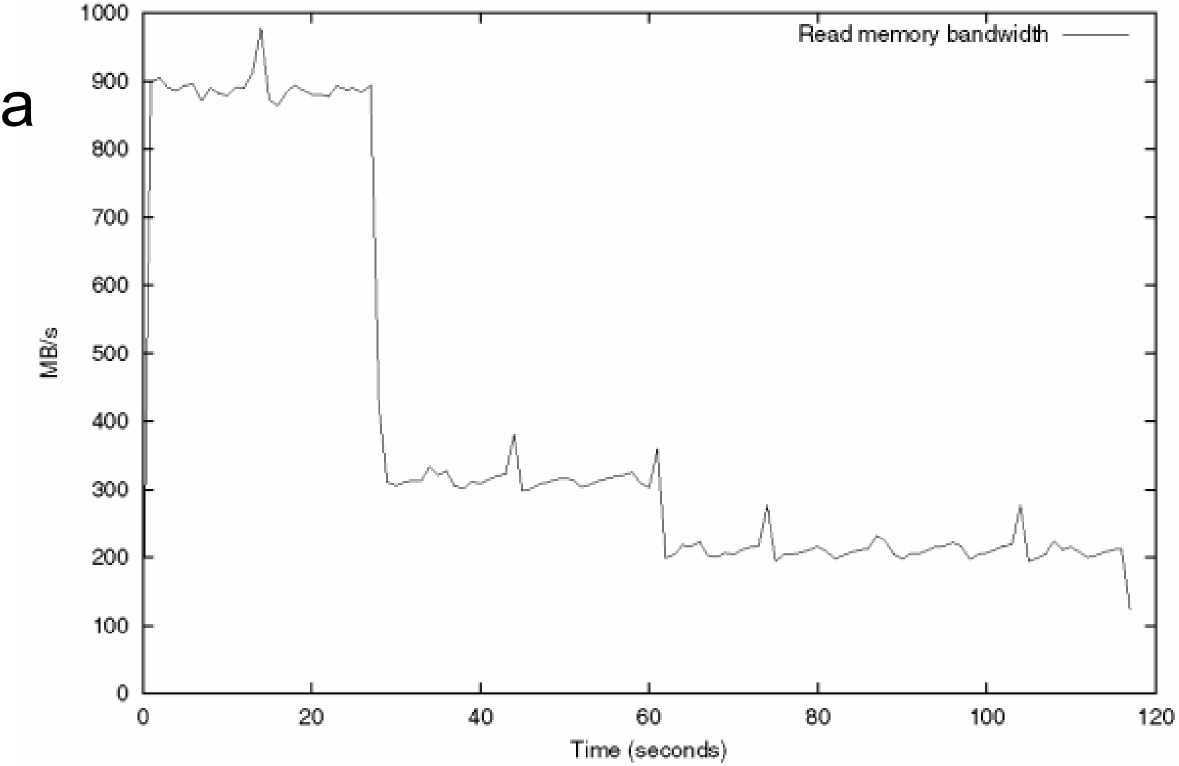
System-wide bandwidth data (-x flag)



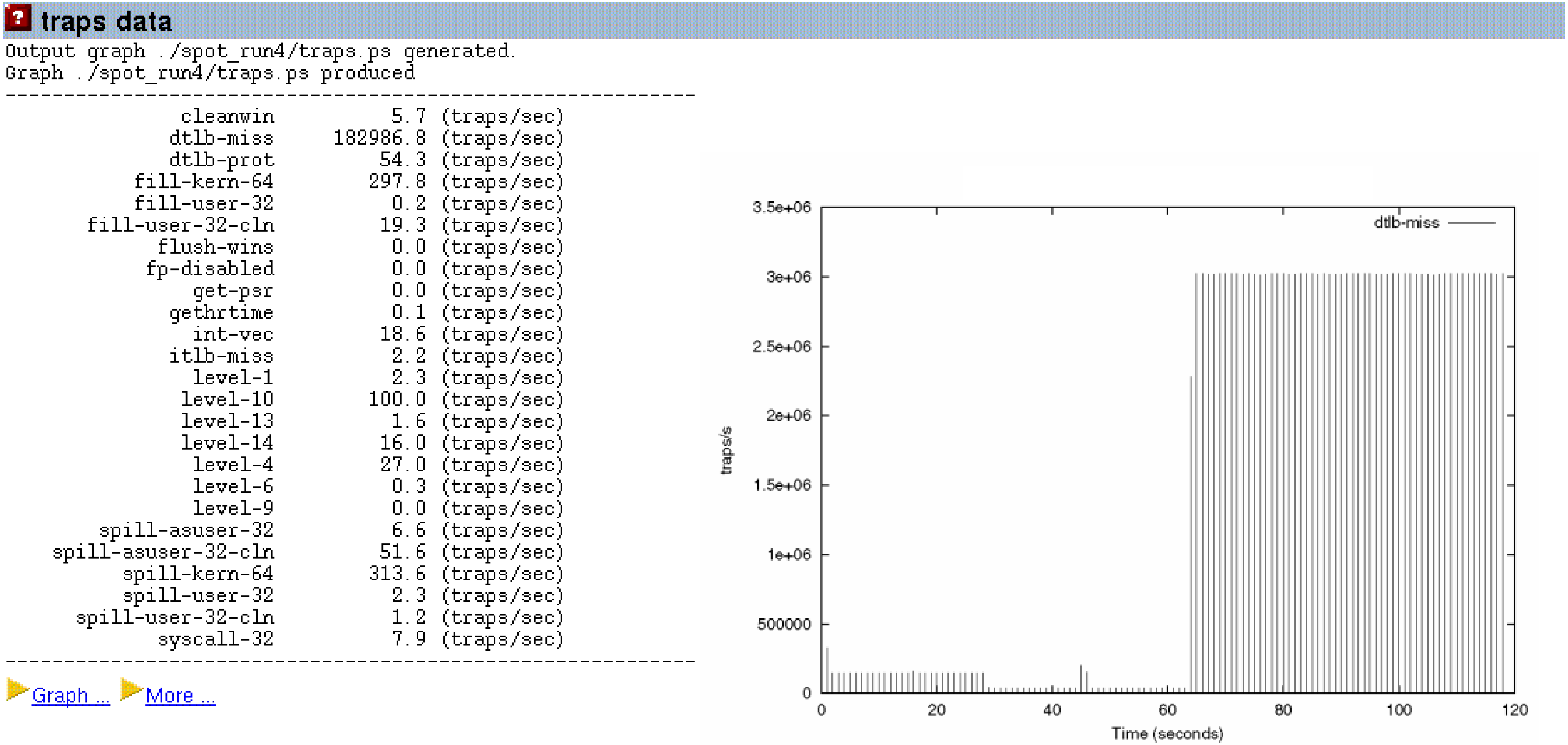
Bandwidth data

```
Graph ./spot_run4/bandwidth.ps produced
Output graph ./spot_run4/bandwidth.ps generated.
-----------------------------------------------------------
 Read  memory bandwidth: 399.613289596688 MB/sec (total bytes = 49025913856)
 Write memory bandwidth: 72.3323692908654 MB/sec (total bytes = 8873980416)
 Total memory bandwidth: 471.945658887553 MB/sec (total bytes = 57899894272)
 Elapsed time   : 117 secs
-----------------------------------------------------------
```

System-wide bandwidth data collected with -x flag and root permissions.



System-wide trap data (-x flag)



traps data

```
Output graph ./spot_run4/traps.ps generated.
Graph ./spot_run4/traps.ps produced
-----------------------------------------------------------
               cleanwin         5.7 (traps/sec)
              dtlb-miss    182986.8 (traps/sec)
              dtlb-prot        54.3 (traps/sec)
           fill-kern-64       297.8 (traps/sec)
           fill-user-32         0.2 (traps/sec)
       fill-user-32-cln        19.3 (traps/sec)
             flush-wins         0.0 (traps/sec)
            fp-disabled         0.0 (traps/sec)
                get-psr         0.0 (traps/sec)
              gethrtime         0.1 (traps/sec)
                int-vec        18.6 (traps/sec)
              itlb-miss         2.2 (traps/sec)
                level-1         2.3 (traps/sec)
               level-10       100.0 (traps/sec)
               level-13         1.6 (traps/sec)
               level-14        16.0 (traps/sec)
                level-4        27.0 (traps/sec)
                level-6         0.3 (traps/sec)
                level-9         0.0 (traps/sec)
         spill-asuser-32        6.6 (traps/sec)
     spill-asuser-32-cln       51.6 (traps/sec)
           spill-kern-64      313.6 (traps/sec)
           spill-user-32        2.3 (traps/sec)
       spill-user-32-cln        1.2 (traps/sec)
              syscall-32        7.9 (traps/sec)
-----------------------------------------------------------
```

Graph ... More ...

System-wide trap information collected with -x flag and root permissions

Agenda

- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- System utilisation
- Other resources

Multithreading code

- Profile application
  - Identify hot code
- Estimate performance gain
  - Amdahl's law
  - Performance gain depends on time spent in region to be parallelised
- Parallelise code

Expected gains from parallelisation

Maximum performance gain from parallelisation is determined by the time spent in code that can be parallelised.

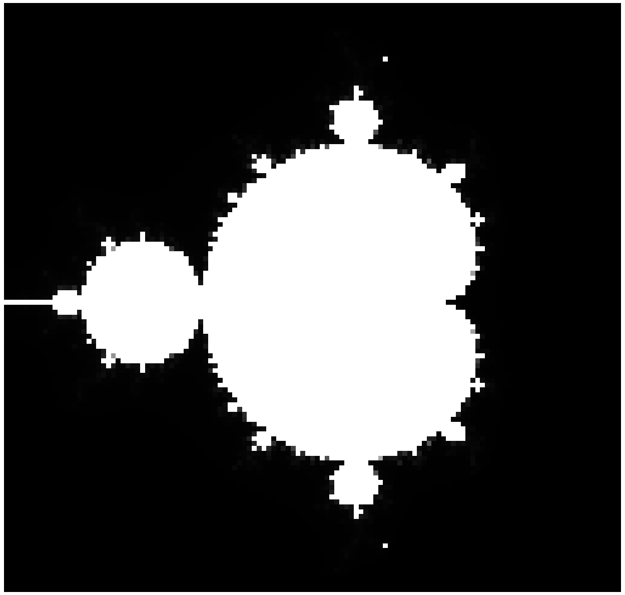
Amdahl's Law.

Approaches to multithreaded coding

- Automatic parallelisation
  - > Easy to use
  - > Relies on compiler extracting parallelism
- OpenMP
  - > Easy to use
  - > Relies on developer to add directives to source code
- POSIX Threads
  - > Very flexible
  - > Potentially hard to use

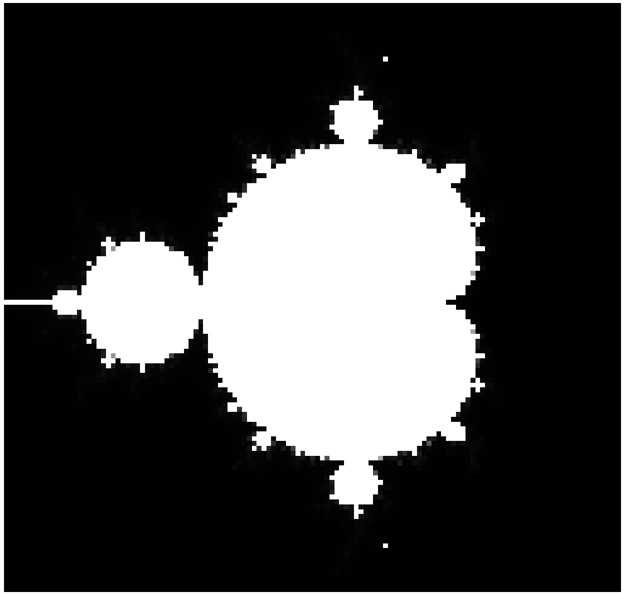
Mandlebrot set example

```
void calculate()
{
  int x,y;
  double xv,yv;
  for (x=0; x<SIZE; x++)
  {
    for (y=0; y<SIZE; y++)
    {
      xv = ((double)(x-SIZE/2))
           /(double)(SIZE/4);
      yv = ((double)(y-SIZE/2))
           /(double)(SIZE/4);
      data[x][y]=inset(xv,yv);
    }
  }
}
```



Determining if a point is in the set

```
int inset(double ix, double iy)
{
   int iterations=0;
   double x=ix, y=iy, x2=x*x, y2=y*y;
   while ((x2+y2<4) && (iterations<1000))
   {
     y = 2 * x * y + iy;
     x = x2 - y2 + ix;
     x2 = x * x;
     y2 = y * y;
     iterations++;
   }
   return iterations;
}
```



Running the application

Include debug information

High optimisation

```
$ cc -g -fast -o m1 m1.c
$ timex m1
real          11.78
user          11.70
sys            0.05
```

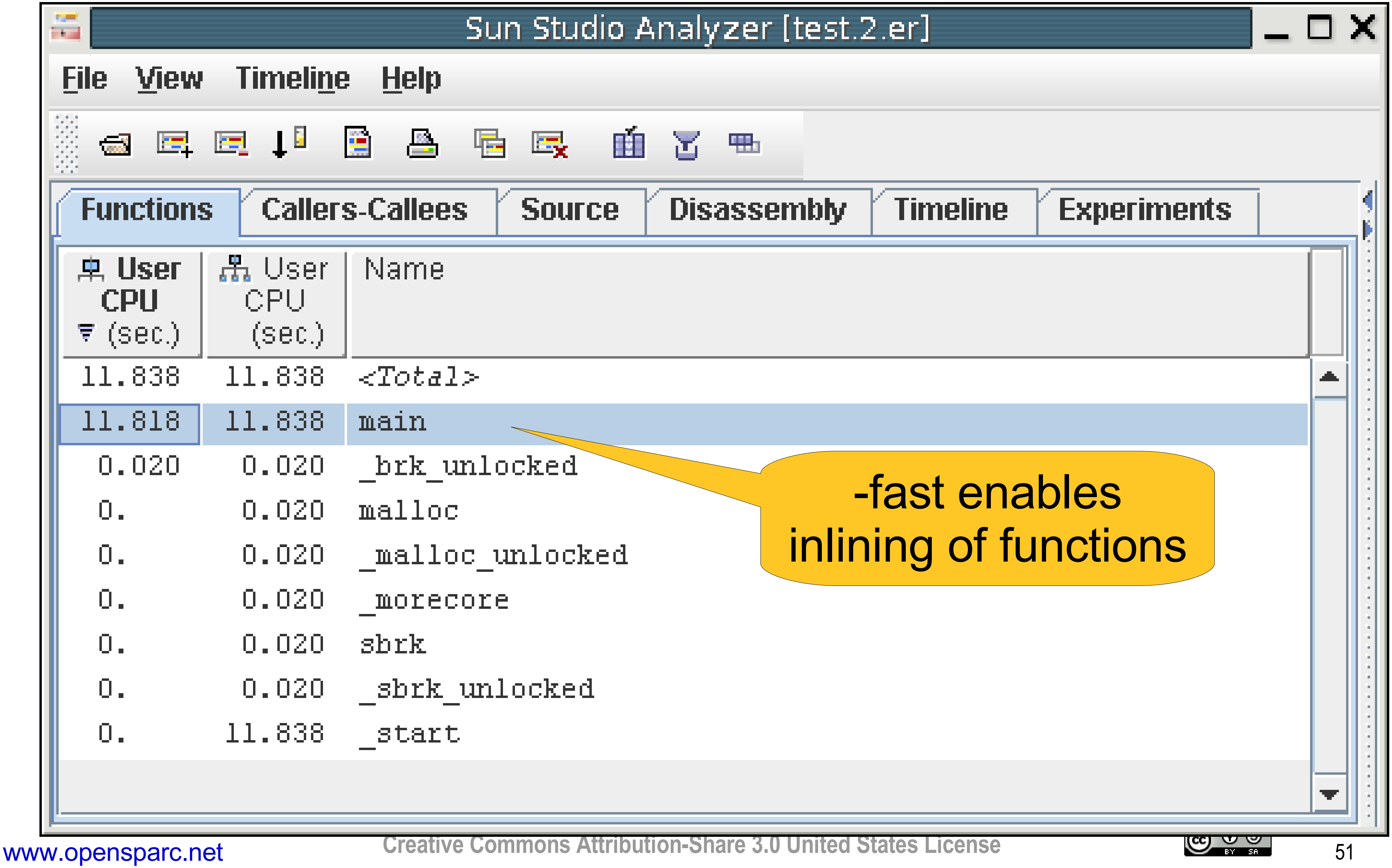
Runtime ~12s

Profiling the application

Include debug information

```
$ cc -g -fast -o m1 m1.c
$ collect m1
$ analyzer test.1.er
```

Application profile



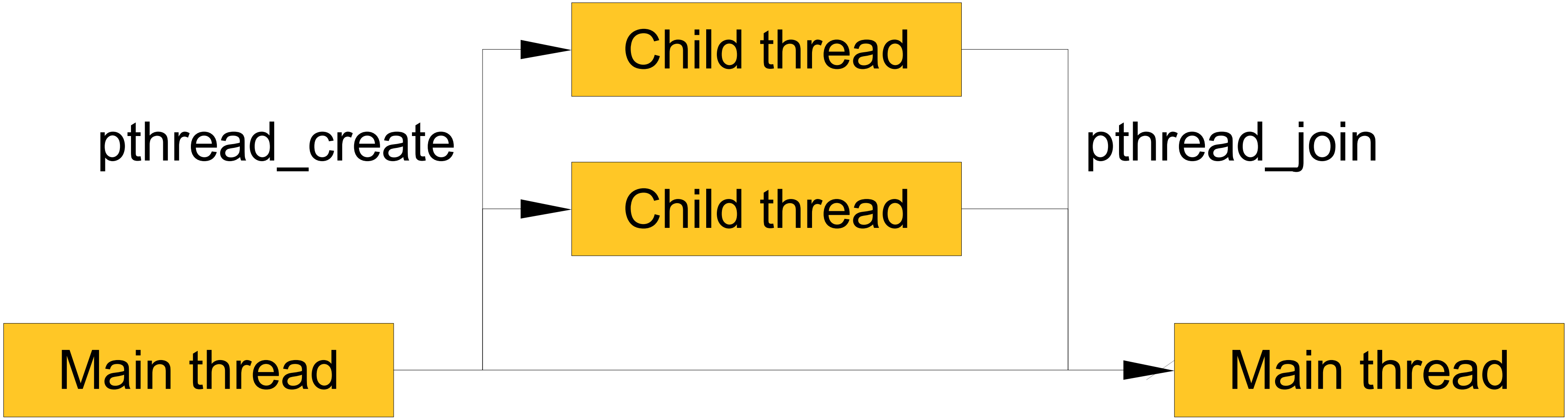
User CPU (sec.)	User CPU (sec.)	Name
11.838	11.838	<Total>
11.818	11.838	main
0.020	0.020	_brk_unlocked
0.	0.020	malloc
0.	0.020	_malloc_unlocked
0.	0.020	_morecore
0.	0.020	sbrk
0.	0.020	_sbrk_unlocked
0.	11.838	_start

-fast enables inlining of functions

POSIX threads (pthreads)

- Advantages:
  - High degree of control of parallelisation.
  - Flexibility in how threads cooperate.
- Disadvantages
  - Typically significant code changes
  - Increases program complexity
  - Can be hard to debug

Pthread model



Setting up pthreads

```
void main()
{
   pthread_t threads[2];
   int id[2];
   data = setup();
   for (int i=0; i<2; i++) {
     id[i]=i;
     pthread_create(&threads[i],0,
           calculate,(void*)&id[i]);
   }
   for (int i=0; i<2; i++) {
     pthread_join(threads[i],0);
   }
   validate();
}
```

Create threads

Wait for threads to finish

Parallelising using pthreads

```
void *calculate(void * arg)
{
   int x,y;
   double xv,yv;
   int id = *(int*)arg;
   int start = (int)(1.0*id/2*SIZE);
   int end =  (int)(1.0*(id+1)/2*SIZE);
   for (x=start; x<end; x++){
     for (y=0; y<SIZE; y++){
 ...
    }
   }
}
```

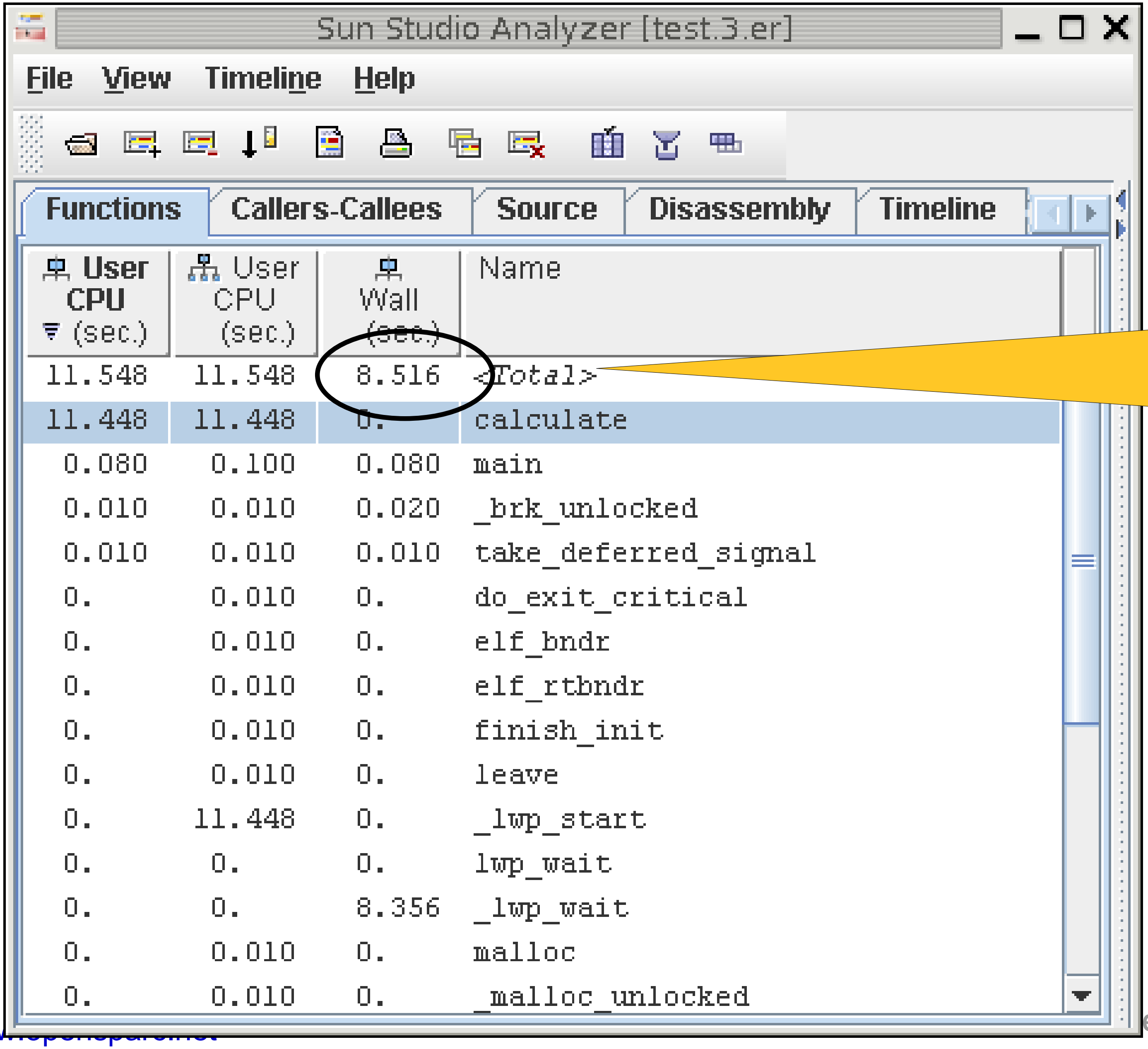
Divide work between threads

Running the application

compiler flags to support pthreads

```
$ cc -g -fast -o m2 m2.c -mt -lpthread
$ collect m2
$ analyzer test.2.er
```

Parallelising using pthreads



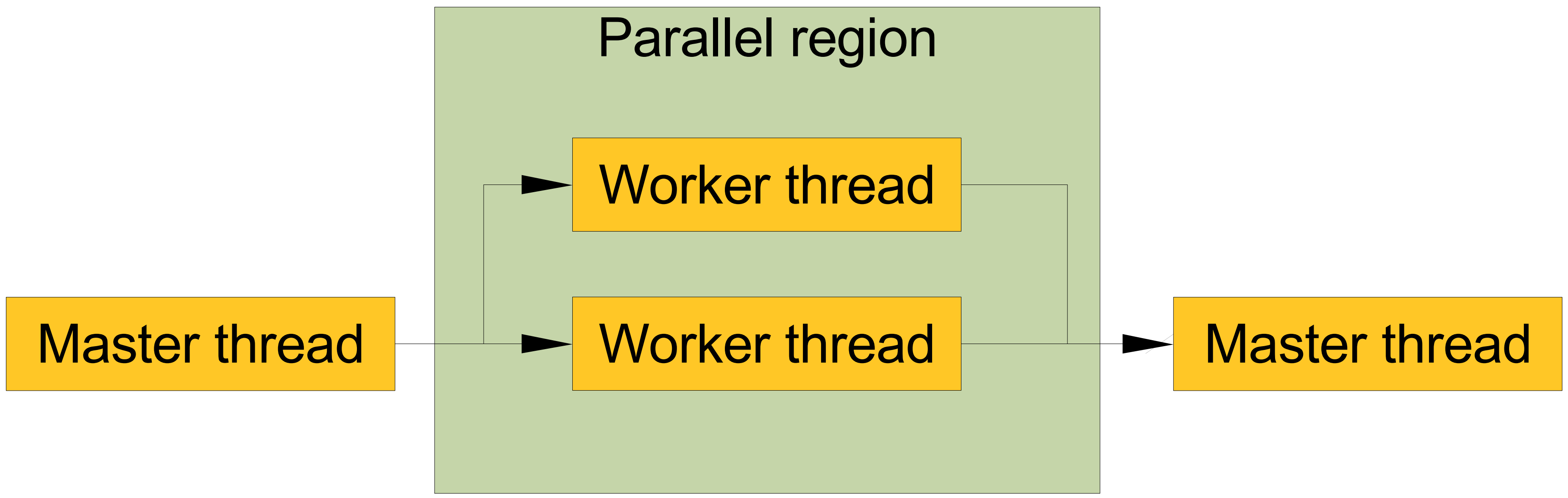
User CPU (sec.)	User CPU (sec.)	Wall (sec.)	Name
11.548	11.548	8.516	*<Total>*
11.448	11.448	0.	calculate
0.080	0.100	0.080	main
0.010	0.010	0.020	_brk_unlocked
0.010	0.010	0.010	take_deferred_signal
0.	0.010	0.	do_exit_critical
0.	0.010	0.	elf_bndr
0.	0.010	0.	elf_rtbndr
0.	0.010	0.	finish_init
0.	0.010	0.	leave
0.	11.448	0.	_lwp_start
0.	0.	0.	lwp_wait
0.	0.	8.356	_lwp_wait
0.	0.010	0.	malloc
0.	0.010	0.	_malloc_unlocked

Wall time of 8.5 seconds. 1.35x faster than original 12s

OpenMP

- Advantages:
  - Easy to use
  - Minimal source changes
  - Incremental parallelisation
- Disadvantages
  - OpenMP 2.5 only supports parallelism through
    - Parallel loops
    - Parallel sections

OpenMP model



OpenMP

```
void calculate()
{
   int x,y;
   double xv,yv;
#pragma omp parallel for private(y,xv,yv)
   for (x=0; x<SIZE; x++){
     for (y=0; y<SIZE; y++){
       xv = ((double)(x-SIZE/2))
            /(double)(SIZE/4);
       yv = ((double)(y-SIZE/2))
            /(double)(SIZE/4);
       data[x][y]=inset(xv,yv);
     }
   }
}
```

Single source line change for parallelization

Using OpenMP

Recognise OpenMP directives

Emit parallelization warnings

```
$ cc -fast -xopenmp -xvpara -o m3 m3.c
$ setenv OMP_NUM_THREADS 2
$ timex m3
real          8.37
user         11.40
sys           0.06
```

Select number of threads to use

Same performance as Pthreads

Sun auto-scoping extension

```
void calculate()
{
   int x,y;
   double xv,yv;
#pragma omp parallel for default(__auto)
   for (x=0; x<SIZE; x++){
     for (y=0; y<SIZE; y++){
...
 $ cc -g -fast -xopenmp -xvpara -o m3 m3.c
 $ er_src -src calculate m3
 ...
    Source OpenMP region below has tag R1
    Variables autoscoped as SHARED in R1: data
    Variables autoscoped as PRIVATE in R1: xv, yv, y
 ...
```

Compiler determines variable scoping

Automatic parallelisation

- Advantages
  - Trivial to use
- Disadvantages
  - Limited ability to parallelise

Using autopar

Enable auto-parallelisation

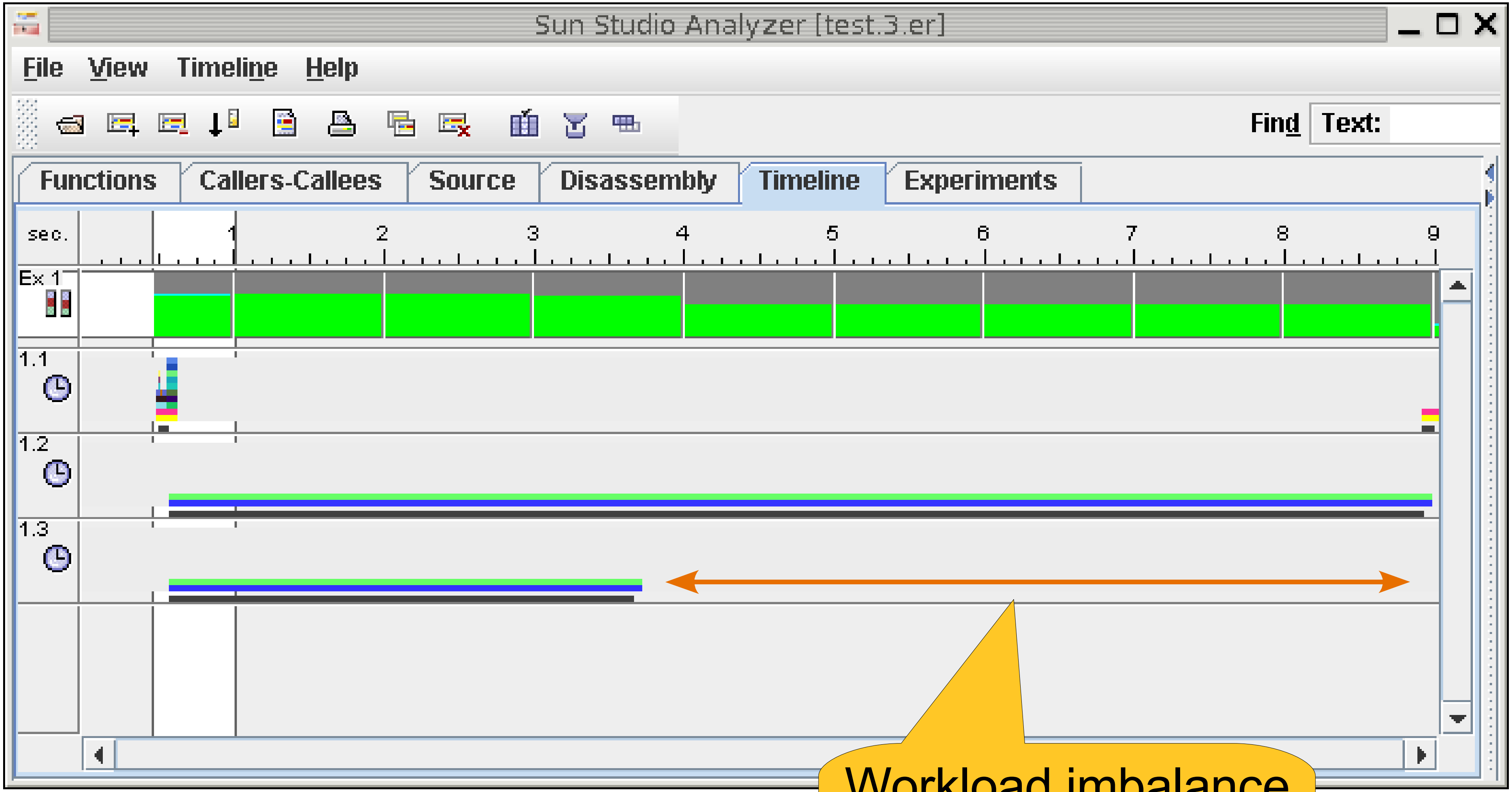
```
$ cc -fast -xautopar -xloopinfo -o m4 m1.c
...
"m1.c", line 40: PARALLELIZED, in
  (inlined loop)
...
```

Emit auto-parallelisation information

```
$ setenv OMP_NUM_THREADS 2
$ timex m4

real         6.00
user        11.69
sys          0.07
```

Wall time of 6 seconds 1.95x faster than original

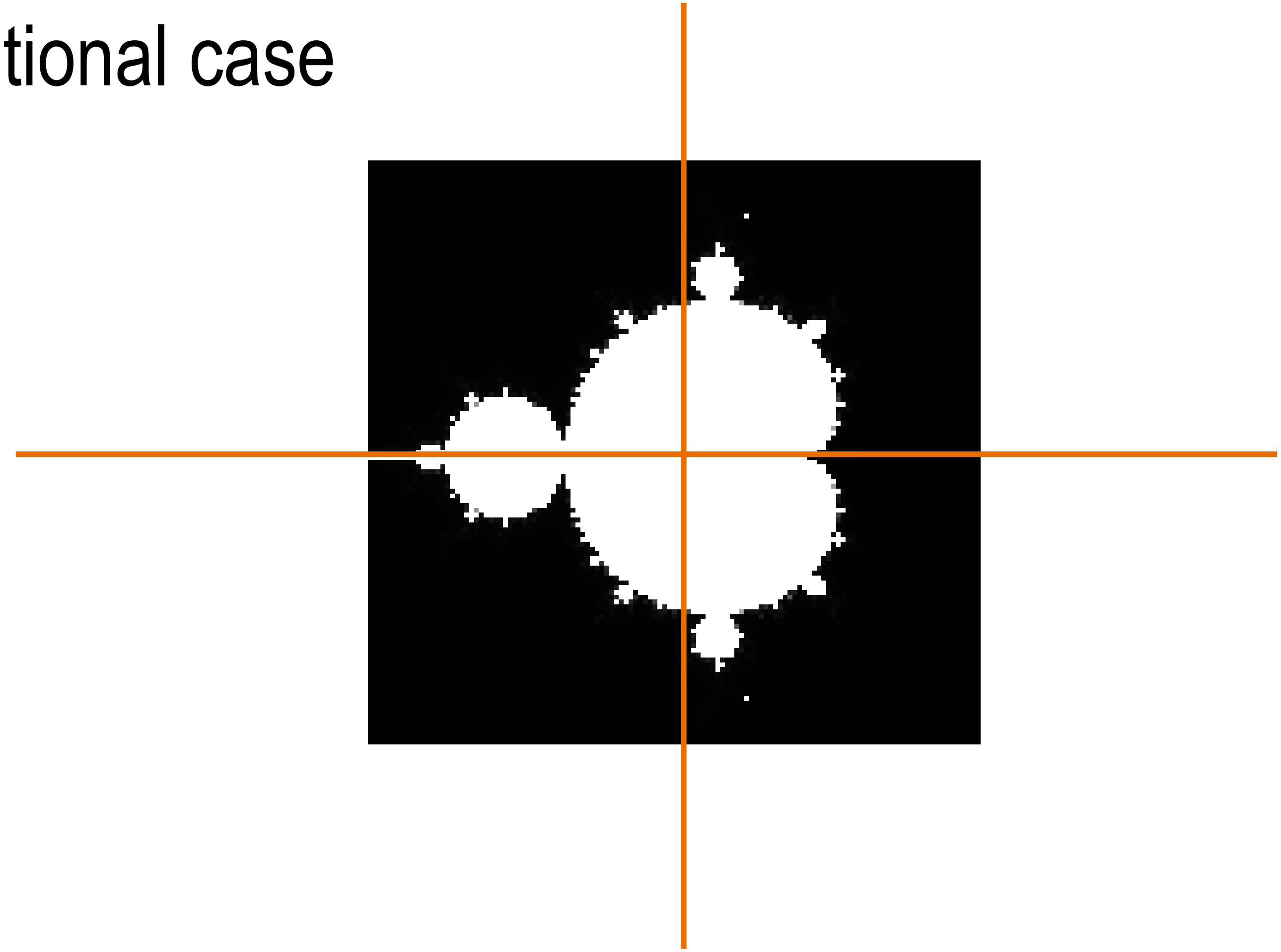
Why is OpenMP/pthread slower?



Workload imbalance between the two threads

Load imbalance

- Horizontal symmetry, not vertical
- Exceptional case



Dynamic scheduling OpenMP

```
void calculate()
{
   int x,y;
   double xv,yv;
#pragma omp parallel for \
        private (y,xv,yv)\
        schedule (dynamic)
   for (x=0; x<SIZE; x++) {
     for (y=0; y<SIZE; y++) {
       xv = ((double)(x-SIZE/2))
            /(double)(SIZE/4);
       yv = ((double)(y-SIZE/2))
            /(double)(SIZE/4);
       data[x][y]=inset(xv,yv);
     }  }  }
```

Use dynamic scheduling

Using OpenMP

```
$ cc -fast -xopenmp -xvpara -o m3 m3.c
$ setenv OMP_NUM_THREADS 2
$ timex m3
real          5.84
user         11.38
sys           0.06
```

Computation evenly divided between 2 threads

Notes on OpenMP and Autopar

What stops autopar?

- Function calls (could modify or read data)
  - Inlining with `-xipo` or `-xinline=<func>`
- Pointer aliasing
  - Use `restrict` keyword or `-xrestrict`
  - Use `-xalias_level=<level>`

Reductions

- Multiple threads cooperating to produce a single value.

```
double sum(double *a, int n)
{
  double t=0;
  for (int i=0; i<n; i++)
  { t+=a[i]; }
  return t;
}
% cc -xautopar -xloopinfo -c -O red.c
line 4: not parallelized, unsafe dependence(t)
```

Reductions

- Order of calculation will be different to serial case

```
double sum(double *a, int n)
{
  double t=0;
  for (int i=0; i<n; i++)
  { t+=a[i]; }
  return t;
}
% cc -xautopar -xreduction -xloopinfo -c -O red.c
line 4: PARALLELIZED, reduction, and serial version
 generated
```

Reductions

- OpenMP

```
double sum(double *a, int n)
{
  double t=0;
#pragma omp parallel for reduction(+:t)
  for (int i=0; i<n; i++)
  { t+=a[i]; }
  return t;
}
```

OpenMP 2.5 – parallel sections

```
#pragma omp parallel sections
{
   #pragma omp section
   {
      /*Region 1*/
   }
   #pragma omp section
   {
      /*Region 2*/
   }
}
```

OpenMP 3.0 - tasks

- Spread tasks over multiple threads

```
node * p = head;
while (p)
{
  #pragma omp task
  {
    process(p);
  }
  p = p->next;
}
```

Sharing data between threads

Sharing data between threads

- Multiple threads updating same data (variable, array etc.)
- Data needs to be `volatile`
  - To avoid being held in register
- Only one thread should update at a time

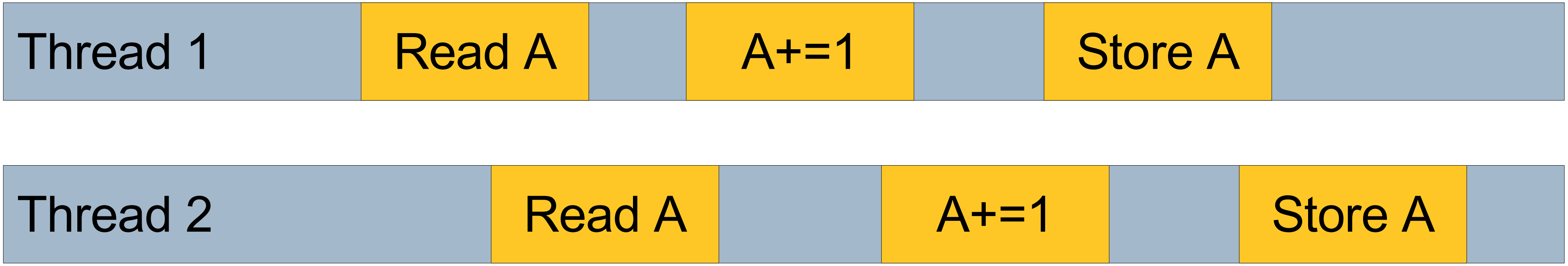
Sharing between Pthreads

```
volatile int sum=0;
...
void *calculate(void * arg) {
   int x,y;
   double xv,yv;
   int id = *(int*)arg;
   int start = (int)(1.0*id/2*SIZE);
   int end =  (int)(1.0*(id+1)/2*SIZE);
   for (x=start; x<end; x++) {
     for (y=0; y<SIZE; y++) {
       ...
       sum+=inset(xv,yv);
     }
   }
}
```

Variable `sum` is `volatile` but shared between multiple threads

Data races

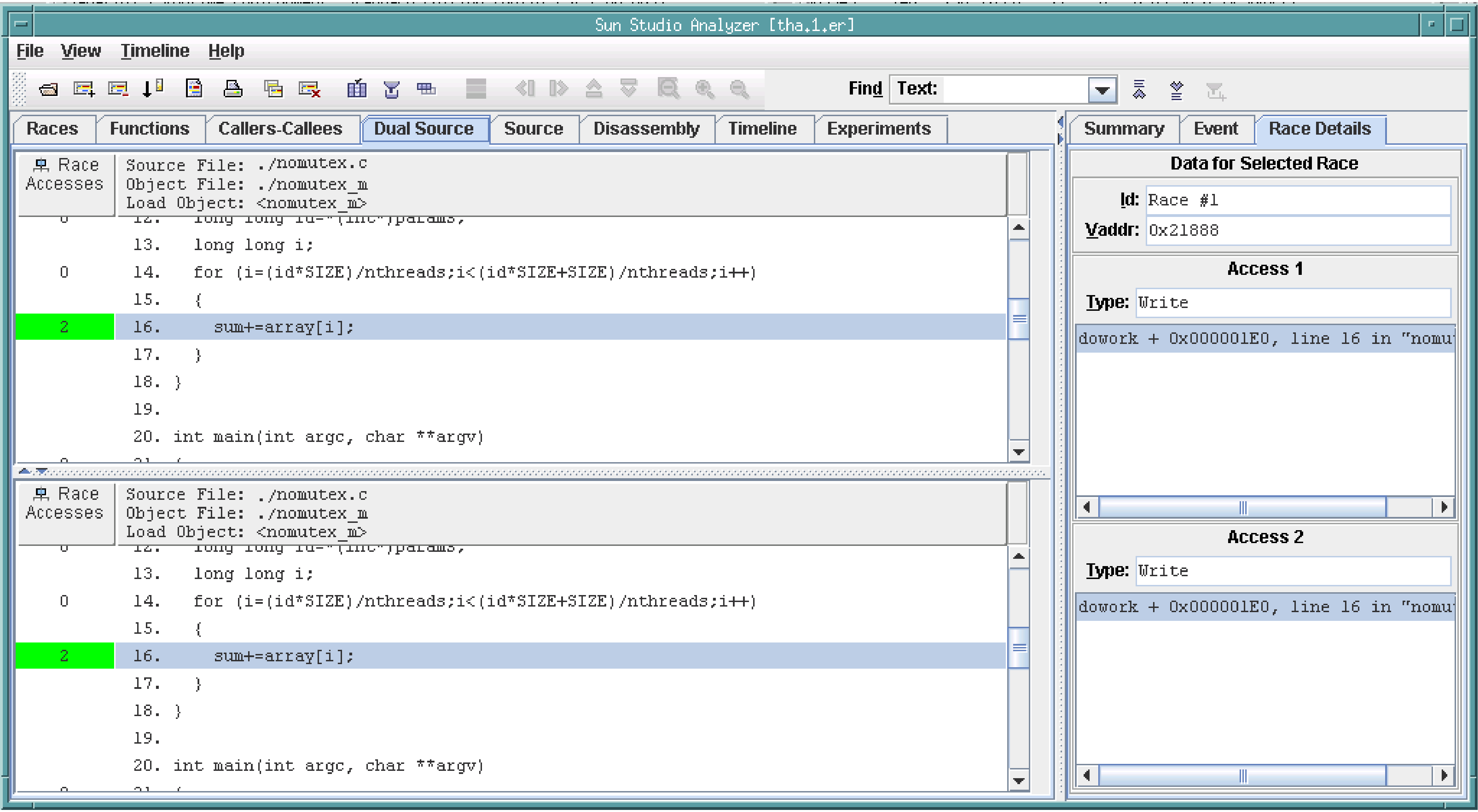
- When:
  - several threads access same variable
  - without synchronisation
  - one or more accesses are writes
- Results in hard to debug non-deterministic behaviour



Thread 1 | Read A | A+=1 | Store A

Thread 2 | Read A | A+=1 | Store A

Thread analyzer



```
Race       Source File: ./nomutex.c
Accesses   Object File: ./nomutex_m
           Load Object: <nomutex_m>
    0        12.   long long id=*(int*)params;
             13.   long long i;
    0        14.   for (i=(id*SIZE)/nthreads;i<(id*SIZE+SIZE)/nthreads;i++)
             15.   {
    2        16.     sum+=array[i];
             17.   }
             18. }
             19.
             20. int main(int argc, char **argv)
    0        21. {
```

```
Race       Source File: ./nomutex.c
Accesses   Object File: ./nomutex_m
           Load Object: <nomutex_m>
    0        12.   long long id=*(int*)params;
             13.   long long i;
    0        14.   for (i=(id*SIZE)/nthreads;i<(id*SIZE+SIZE)/nthreads;i++)
             15.   {
    2        16.     sum+=array[i];
             17.   }
             18. }
             19.
             20. int main(int argc, char **argv)
    0        21. {
```

Data for Selected Race

Id: Race #1

Vaddr: 0x21888

Access 1

Type: Write

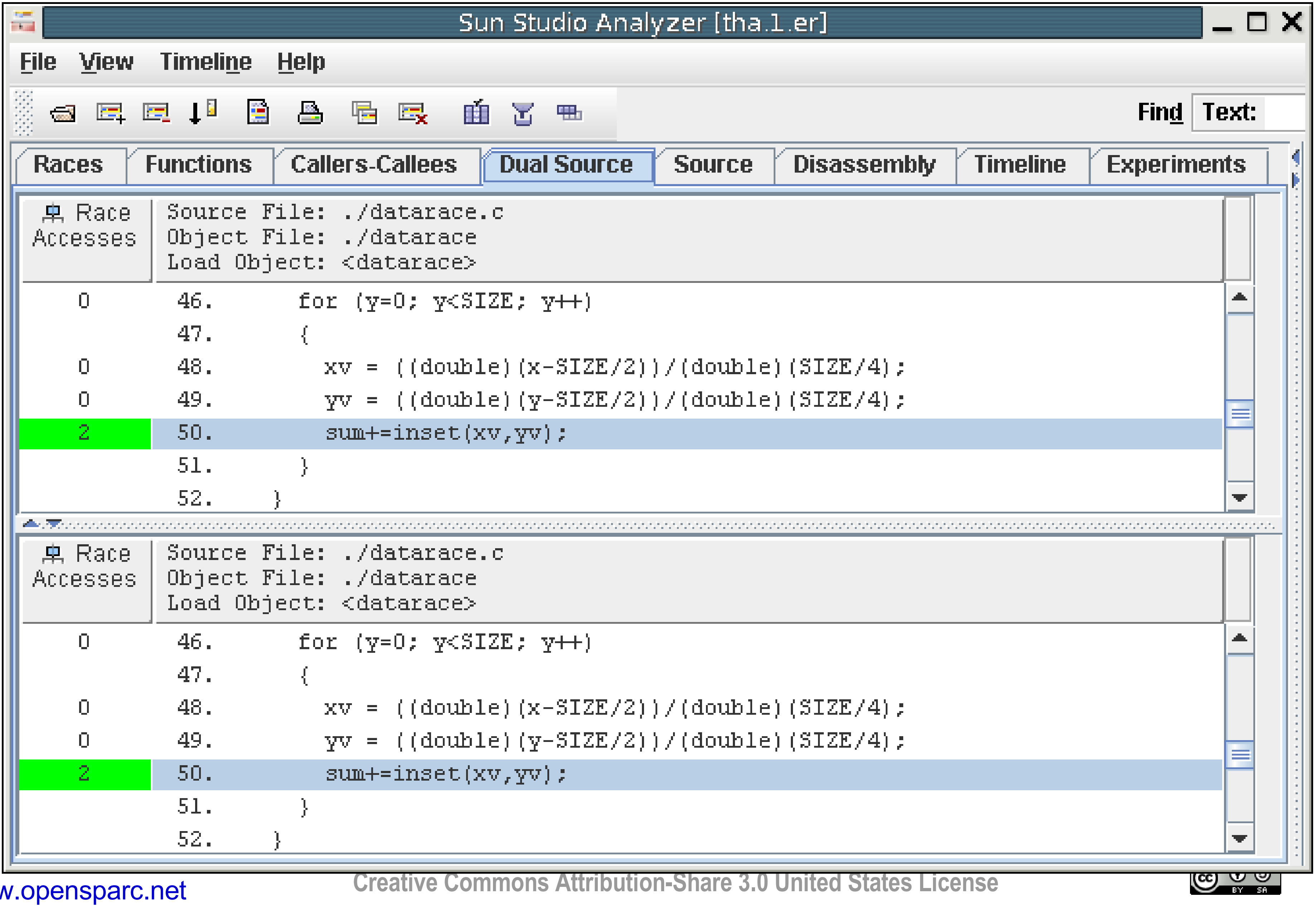
```
dowork + 0x000001E0, line 16 in "nomu
```

Access 2

Type: Write

```
dowork + 0x000001E0, line 16 in "nomu
```

View source code



```
 Race      Source File: ./datarace.c
Accesses   Object File: ./datarace
           Load Object: <datarace>
   0        46.       for (y=0; y<SIZE; y++)
            47.       {
   0        48.         xv = ((double)(x-SIZE/2))/(double)(SIZE/4);
   0        49.         yv = ((double)(y-SIZE/2))/(double)(SIZE/4);
   2        50.         sum+=inset(xv,yv);
            51.       }
            52.     }
```

```
 Race      Source File: ./datarace.c
Accesses   Object File: ./datarace
           Load Object: <datarace>
   0        46.       for (y=0; y<SIZE; y++)
            47.       {
   0        48.         xv = ((double)(x-SIZE/2))/(double)(SIZE/4);
   0        49.         yv = ((double)(y-SIZE/2))/(double)(SIZE/4);
   2        50.         sum+=inset(xv,yv);
            51.       }
            52.     }
```

Notes: Data races

Generate instrumented executable

```
$ cc -g -xinstrument=datarace -mt \
  -lpthread -o race race.c
$ collect -r on datarace
$ analyzer tha.1.er
```

Fixing data accesses

- Single thread access:
  - Mutex locks
  - Critical regions (OpenMP)
- Lock-less:
  - Atomic operations (**`man atomic_ops`**)
- Data sharing:
  - Thread local storage
  - OpenMP reduction directive

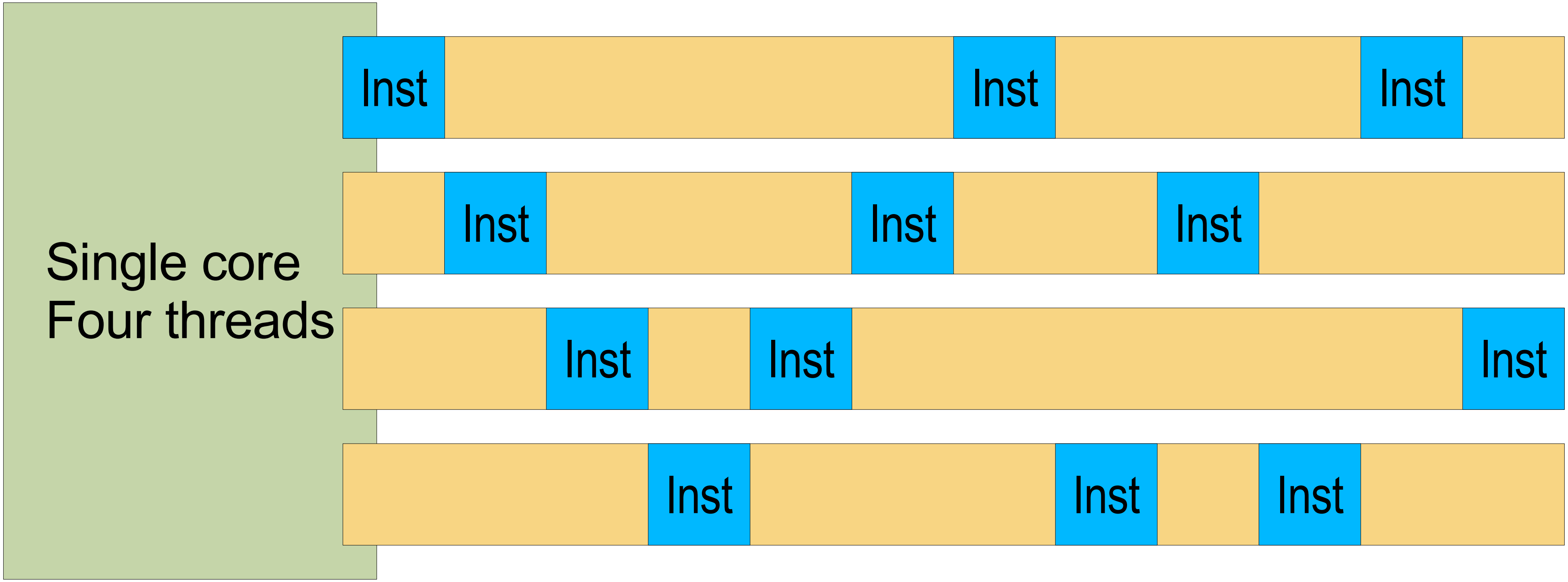
Agenda

- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- System utilisation
- Other resources

System utilisation

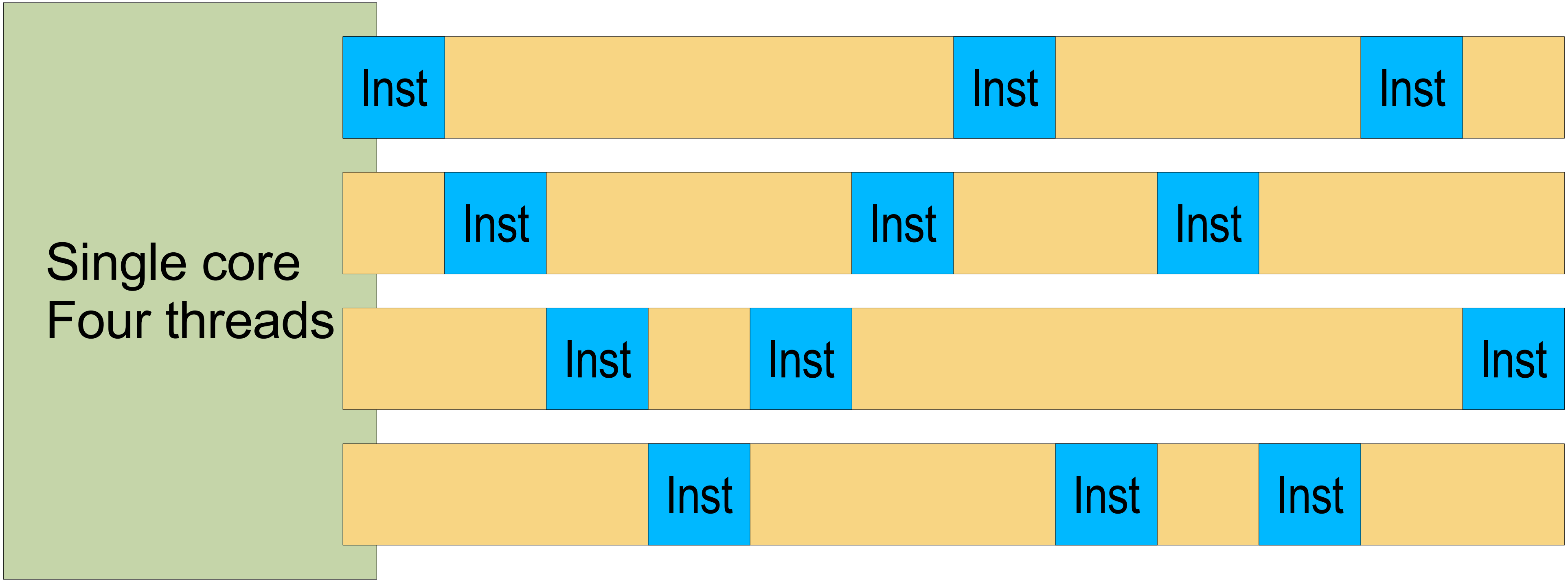
- Tools:
  - System aggregate: **`vmstat`**
  - Processes: **`prstat`**
  - Processors: **`mpstat`**
- Indicate that processors are busy
- But not the utilisation of the cores

Core utilisation



- Multiple threads share core
- Instruction count indicates core utilisation
- Utilisation = Instructions issued
  / Max. Instruction Issue rate

Stall and instruction budgets



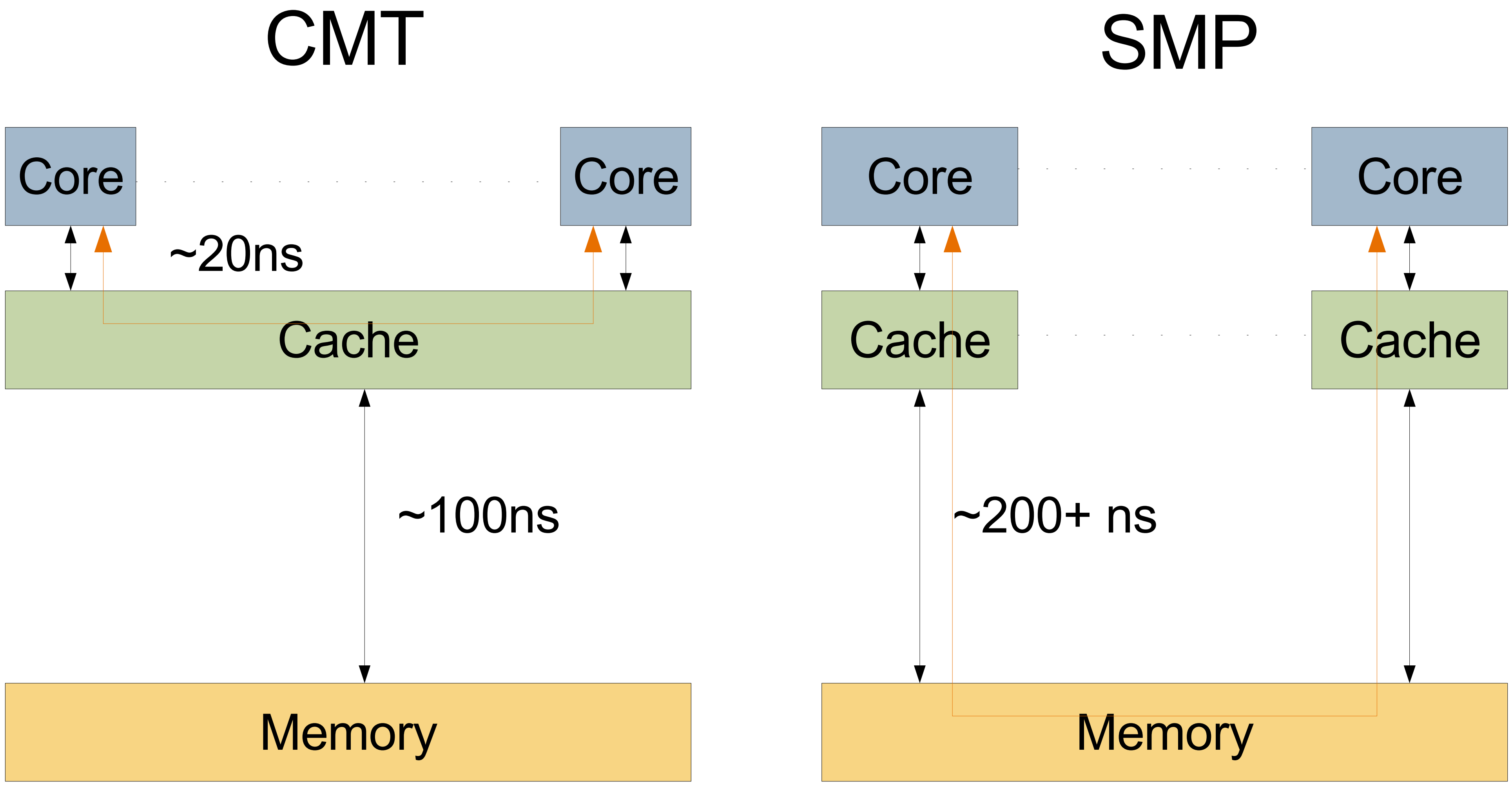
Single core
Four threads

- One instruction issued from one of four threads
- Three threads do not issue – can be stalled or ready
- Stall budget of 3x instruction issue
- Instruction issue rate not impacted until stall budget exceeded

Processor utilisation

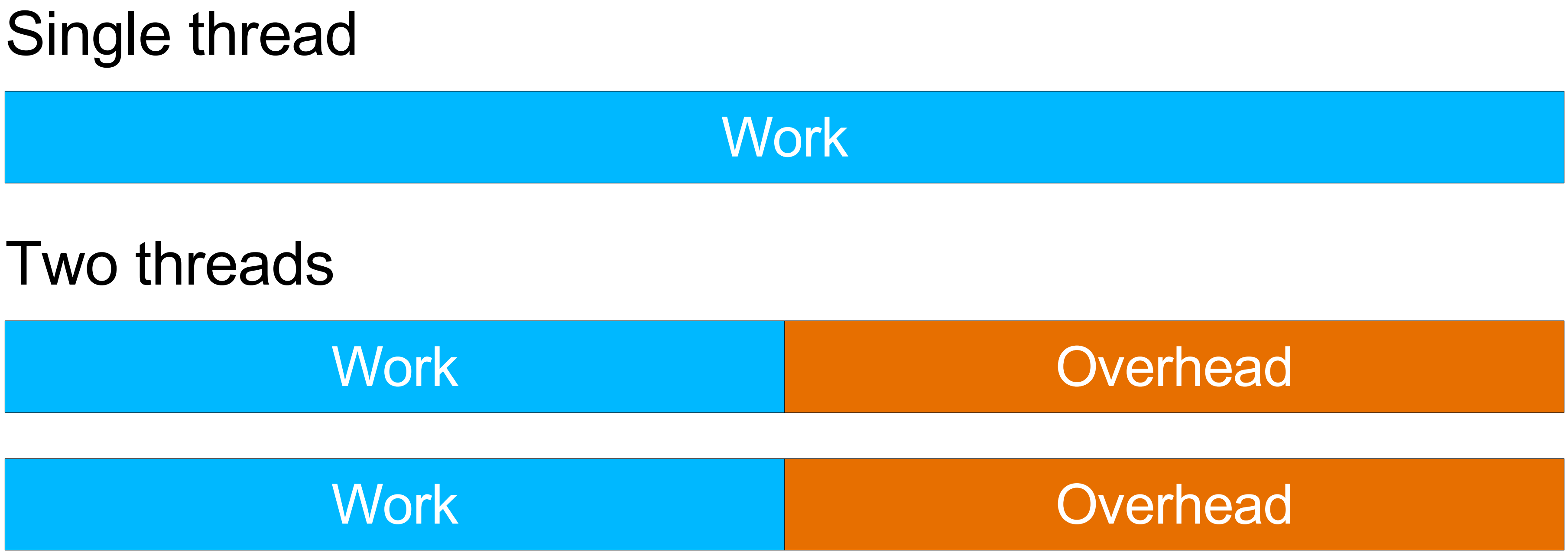
- Utilisation = percentage of peak instruction issue rate
- Raw data: **`cpustat`**
- Formatted: **`corestat`**
- Per process:
  - **`cputrack`**
  - **`ripc`** (from spot)

CMT vs SMP


CMT
SMP
Core
Core
~20ns
Cache
~100ns
Memory
Core
Core
Cache
Cache
~200+ ns
Memory

Limits of parallelisation

- Limit to parallelisation is:
  - > Costs of synchronisation
  - > Is greater than gain from more threads



Single thread

Two threads

Microparallelism

- Many cores
- Low communication latency between cores
- => Synchronisation cost is low
- => Profitable to parallelise smaller regions

Microparallelism outline

- Get next quanta of work
- Check that it is safe to start work
  - No other thread touching writing to read data
  - Avoids data races
  - Ensures correct ordering
- Perform work
- Update status to indicate work completed

Optimising for CMT

- Traditional approach
  - First reduce latency
  - Then parallelise
- CMT approach
  - First parallelise
  - Check instruction issue rate
    - If at max, need to reduce instruction count
    - If not, improve instruction latency

Compiling for CMT

- Generic compiler flags ok:
  - `-xtarget=generic`
- Usual optimisations are good:
  - Profile feedback: `-xprofile=[collect:|use:]`
  - Crossfile optimisation: `-xipo`
  - At least `-O`
  - Perhaps `-fast`

Summary

- Always profile your application
- Always use optimisation
- Use multiple threads
  - > Autopar
  - > OpenMP
  - > PThreads
- On CMT synchronisation costs are lower
- Check instruction count
  - > corestat
  - > BIT

Agenda

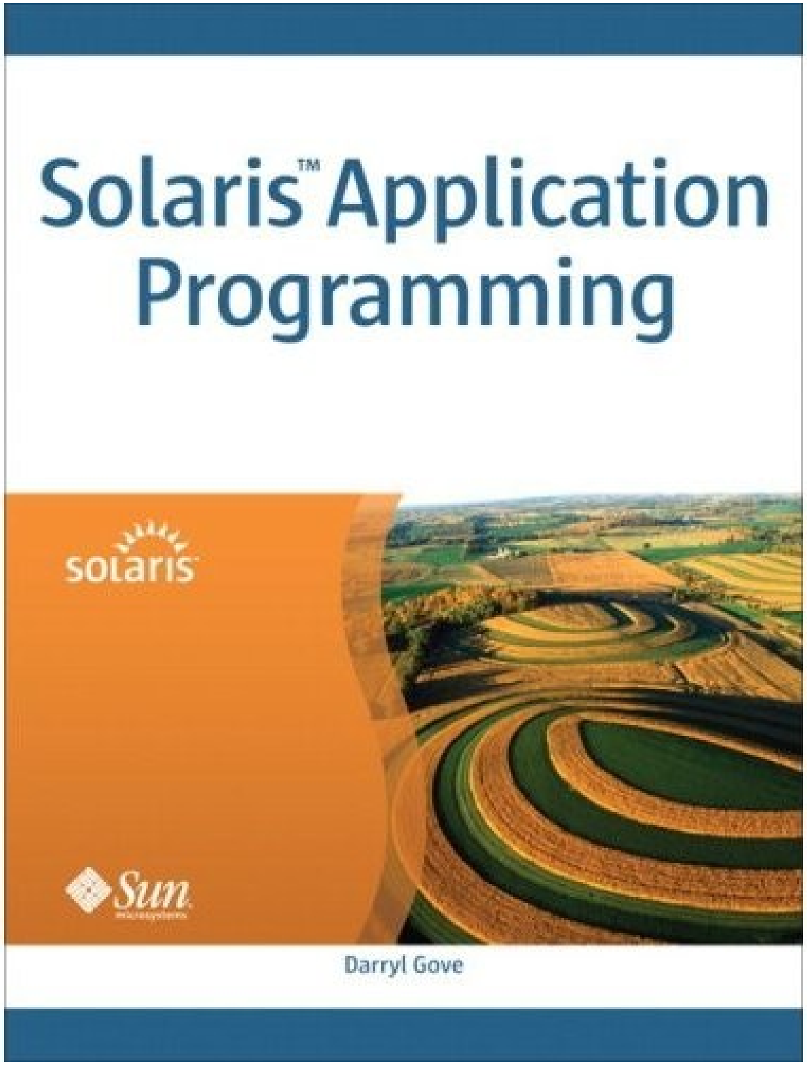
- Compiler and tools options
- Compiling applications
- Profiling applications
- Writing parallel applications
- Calculating system utilisation
- Other resources

Web-based resources

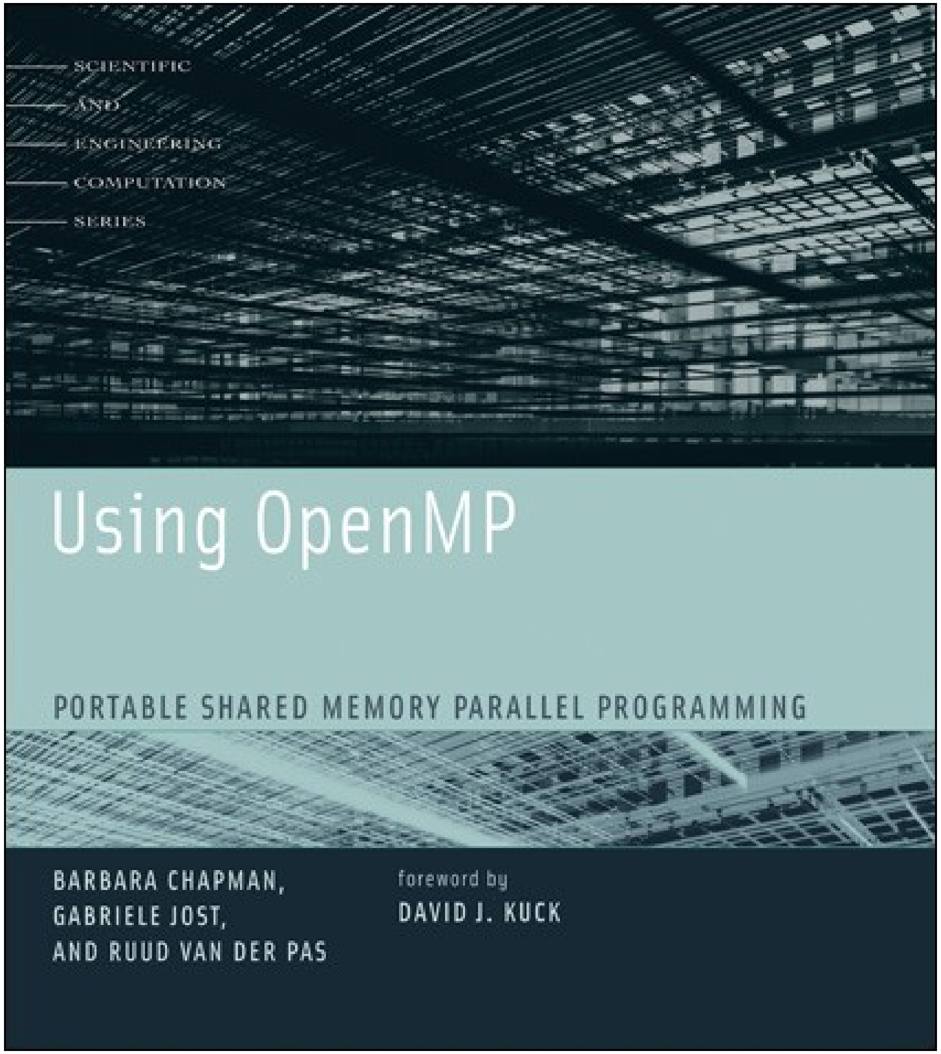
- Developer portal:
  http://developers.sun.com/
- Documentation:
  http://developers.sun.com/sunstudio/documentation/index.jsp
- Forums:
  http://developers.sun.com/sunstudio/community/forums/index.jsp
- Blogs:
  http://developers.sun.com/sunstudio/community/blogs/index.jsp

Solaris Application Programming

"*Solaris Application Programming* ... gives you the background information, tips, and techniques for developing, optimizing, and debugging applications on Solaris."



Using OpenMP



SCIENTIFIC AND ENGINEERING COMPUTATION SERIES

Using OpenMP

PORTABLE SHARED MEMORY PARALLEL PROGRAMMING

BARBARA CHAPMAN,
GABRIELE JOST,
AND RUUD VAN DER PAS

foreword by
DAVID J. KUCK

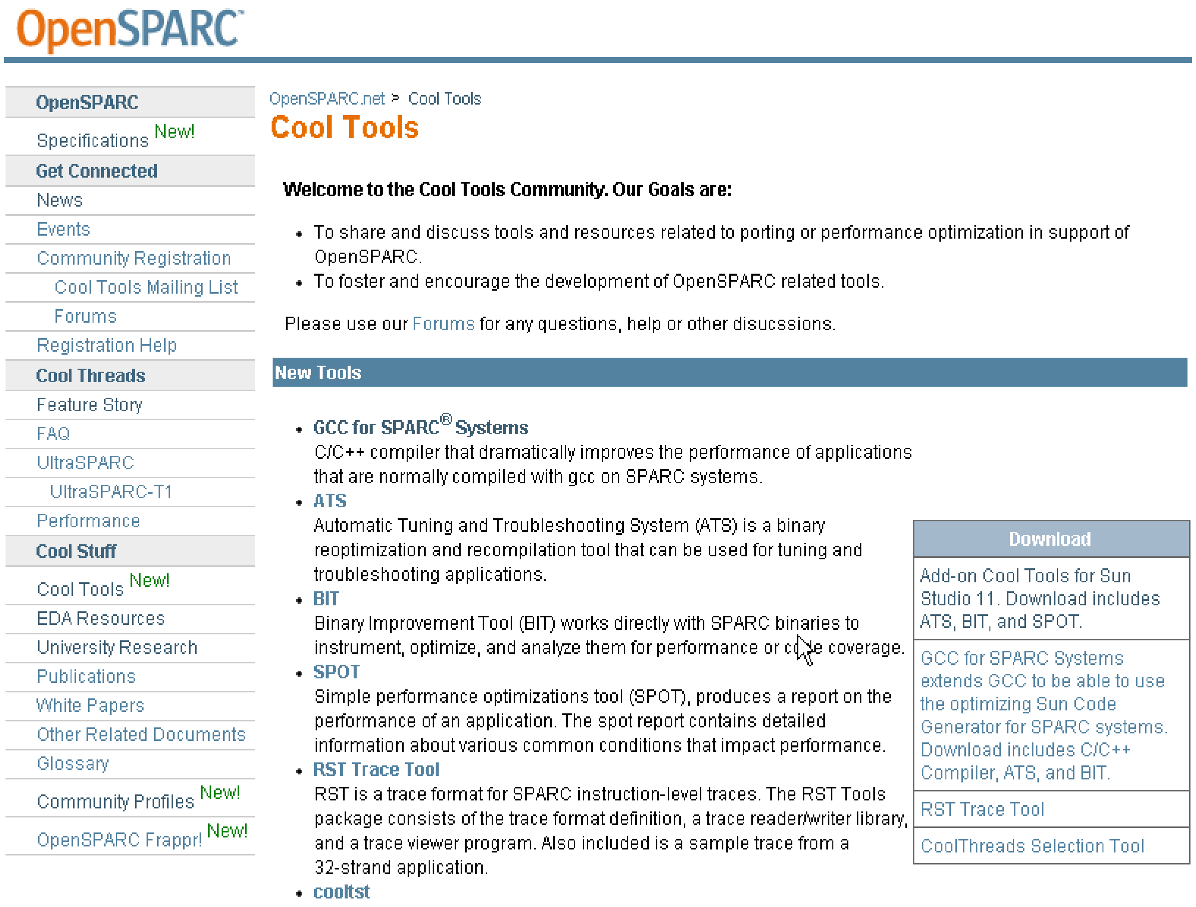
"*Using OpenMP* offers a comprehensive introduction to parallel programming concepts and a detailed overview of OpenMP."

OpenSPARC.net: Find Cool Tools

- Your resource for developer tools – FREE !
  - > **GCC**
    SPARC systems highly optimized
  - > **SPOT**
    Simple Performance Optimization Tool
  - > **RST Trace**
  - > **ATS**
    Automatic Tuning System

And –
Share *your* tools with the community at this site

64 bit 32 threads OpenSPARC



OpenSPARC

OpenSPARC
Specifications New!
Get Connected
News
Events
Community Registration
Cool Tools Mailing List
Forums
Registration Help
Cool Threads
Feature Story
FAQ
UltraSPARC
UltraSPARC-T1
Performance
Cool Stuff
Cool Tools New!
EDA Resources
University Research
Publications
White Papers
Other Related Documents
Glossary
Community Profiles New!
OpenSPARC Frappr! New!

OpenSPARC.net > Cool Tools

Cool Tools

Welcome to the Cool Tools Community. Our Goals are:

- To share and discuss tools and resources related to porting or performance optimization in support of OpenSPARC.
- To foster and encourage the development of OpenSPARC related tools.

Please use our Forums for any questions, help or other disucssions.

New Tools

- **GCC for SPARC® Systems**
  C/C++ compiler that dramatically improves the performance of applications that are normally compiled with gcc on SPARC systems.
- **ATS**
  Automatic Tuning and Troubleshooting System (ATS) is a binary reoptimization and recompilation tool that can be used for tuning and troubleshooting applications.
- **BIT**
  Binary Improvement Tool (BIT) works directly with SPARC binaries to instrument, optimize, and analyze them for performance or code coverage.
- **SPOT**
  Simple performance optimizations tool (SPOT), produces a report on the performance of an application. The spot report contains detailed information about various common conditions that impact performance.
- **RST Trace Tool**
  RST is a trace format for SPARC instruction-level traces. The RST Tools package consists of the trace format definition, a trace reader/writer library, and a trace viewer program. Also included is a sample trace from a 32-strand application.
- **cooltst**

Download

Add-on Cool Tools for Sun Studio 11. Download includes ATS, BIT, and SPOT.

GCC for SPARC Systems extends GCC to be able to use the optimizing Sun Code Generator for SPARC systems. Download includes C/C++ Compiler, ATS, and BIT.

RST Trace Tool

CoolThreads Selection Tool