Parallelising serial applications

Darryl Gove
Compiler Performance Engineering

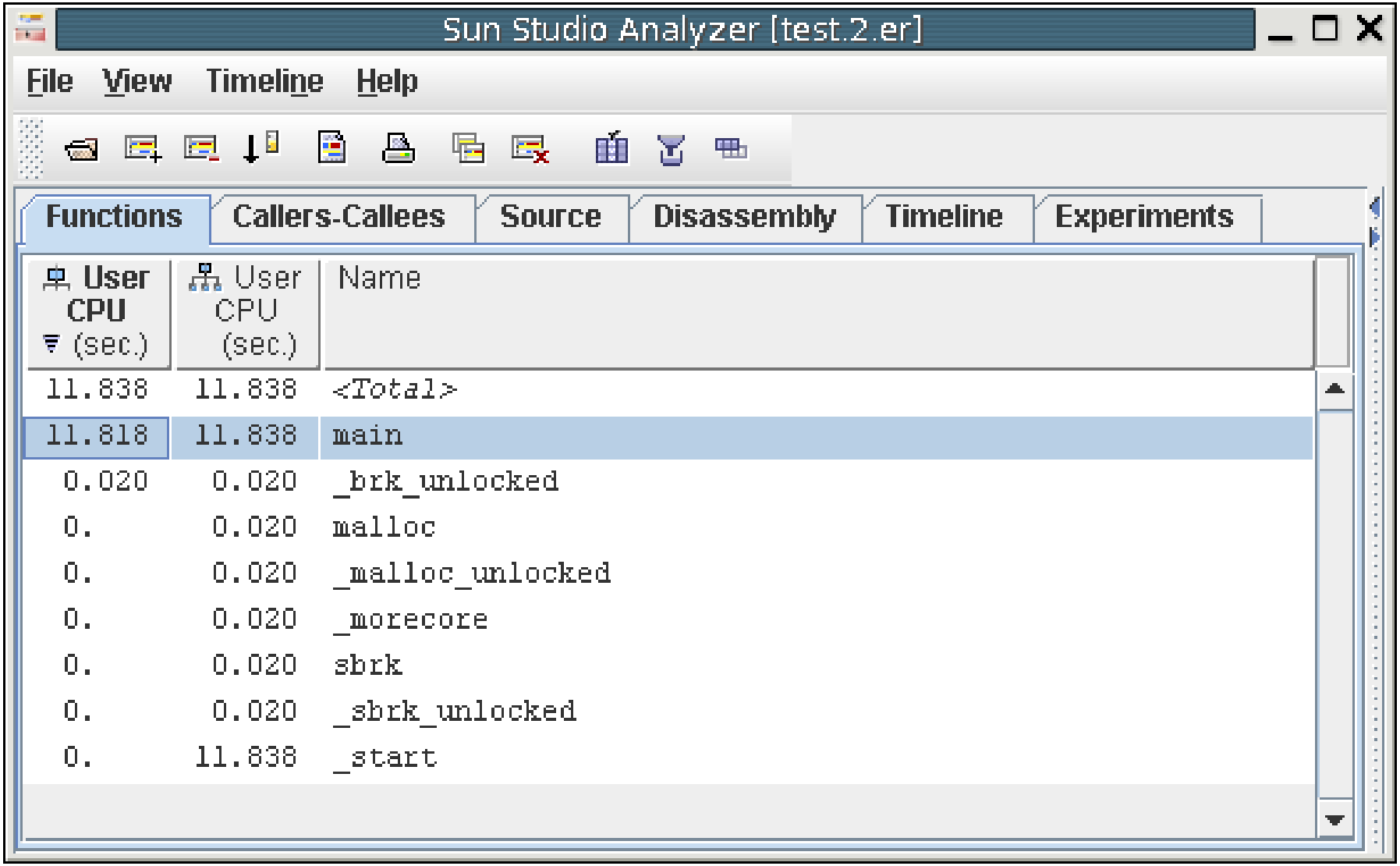
Topics

- Process
- Tools
- Expectations

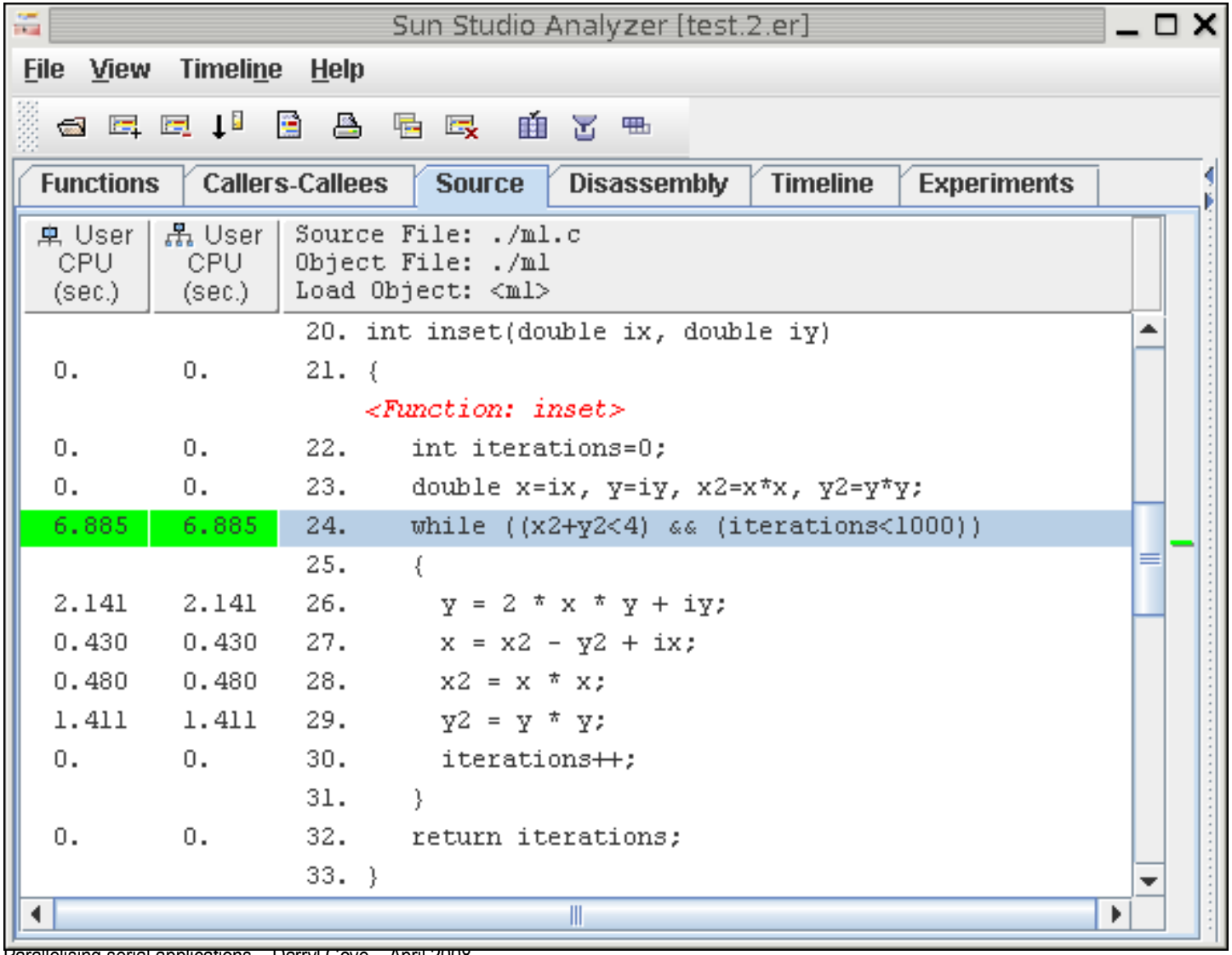
Profile

- Compile with debug info
  - -g [C/Fortran]
  - -g0 [C++]
  - Enables mapping of disassembly to source
- Profile with Performance Analyzer
  - collect <app> <params>
  - collect -P <pid>

Application profile



Source level profile



Parallelisation opportunities

- Large tasks
- No dependencies
- Avoid synchronisation
- Examples:
  - Independent transactions
  - Different iterations of loops
  - Different tasks

Expected gains from parallelisation

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

Amdahl's Law.

POSIX Threads

- POSIX Threads (Pthreads)
  - Very flexible
  - Requires new code
  - May mean restructuring

Setting up Pthreads

```
void main()
{
    pthread_t threads[2];
    int id[2];
    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);
    }
}
```

Target routine

Parameter passing

Notes: Pthreads

- Solaris 9
  - `-mt -lpthread`
- Solaris 10
  - `-mt`
  - libc includes thread library

OpenMP

- OpenMP
  - Minimal source modification
  - Incremental parallelisation
  - Serial and parallel versions from same source
  - Some skill needed
  - Limited parallelisation options
    - 2.5 standard
      - Parallel for
      - Parallel sections
    - 3.0 standard
      - Tasks

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);
      }
    }
}
```

Notes: OpenMP

- Compile with
  - `-xopenmp`
- Warnings with
  - `-xloopinfo -xvpara`
- Set number of threads threads with
  - `OMP_NUM_THREADS`

Autoparallelisation

- Autoparallelisation
  - Easy to use
  - Limited range
  - Improve with:
    - Source changes
    - Compiler flags

Using autopar

```
$ cc -fast -xautopar -xreduction \
> -xloopinfo -xvpara m1.c
...
"m1.c", line 40: PARALLELIZED,
  interchanged (inlined loop)
...
```

Emit auto-parallelisation information

Notes: autoparallelisation

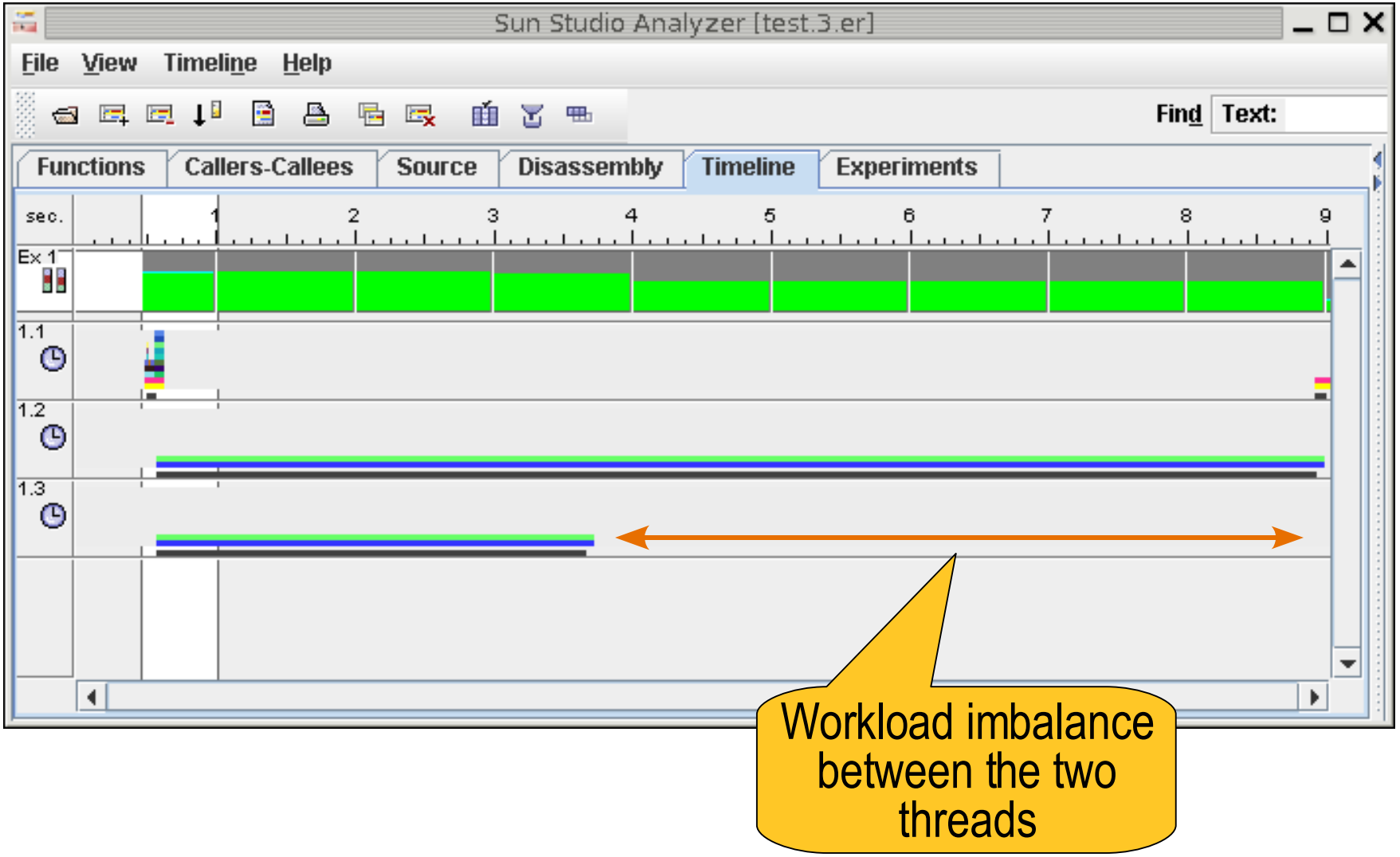
- Compiler options
  - `-xautopar -xreduction`
- Warnings:
  - `-xloopinfo -xvpara`
- Set number of threads threads with
  - `OMP_NUM_THREADS`

Reductions

- Multiple threads cooperating to produce a single value.
- Order of calculation will be different to serial case

```
for (i=0; i<N; i++)
{
  total += a[i];
}
```

Profiling a Multi-threaded application



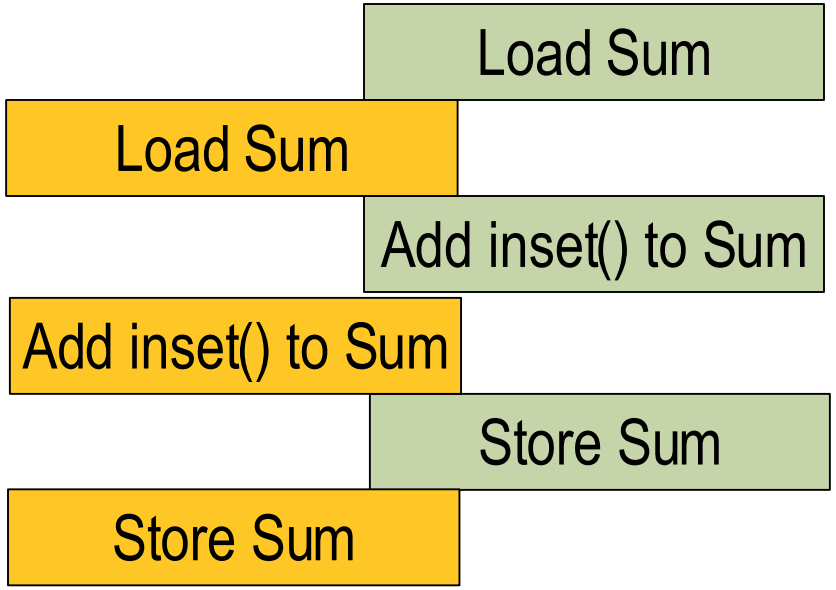
Data races

```
$ cc -fast -mt -o race race.c -lpthread

$ datarace
sum = 385158800

$ datarace
sum = 385071679
```

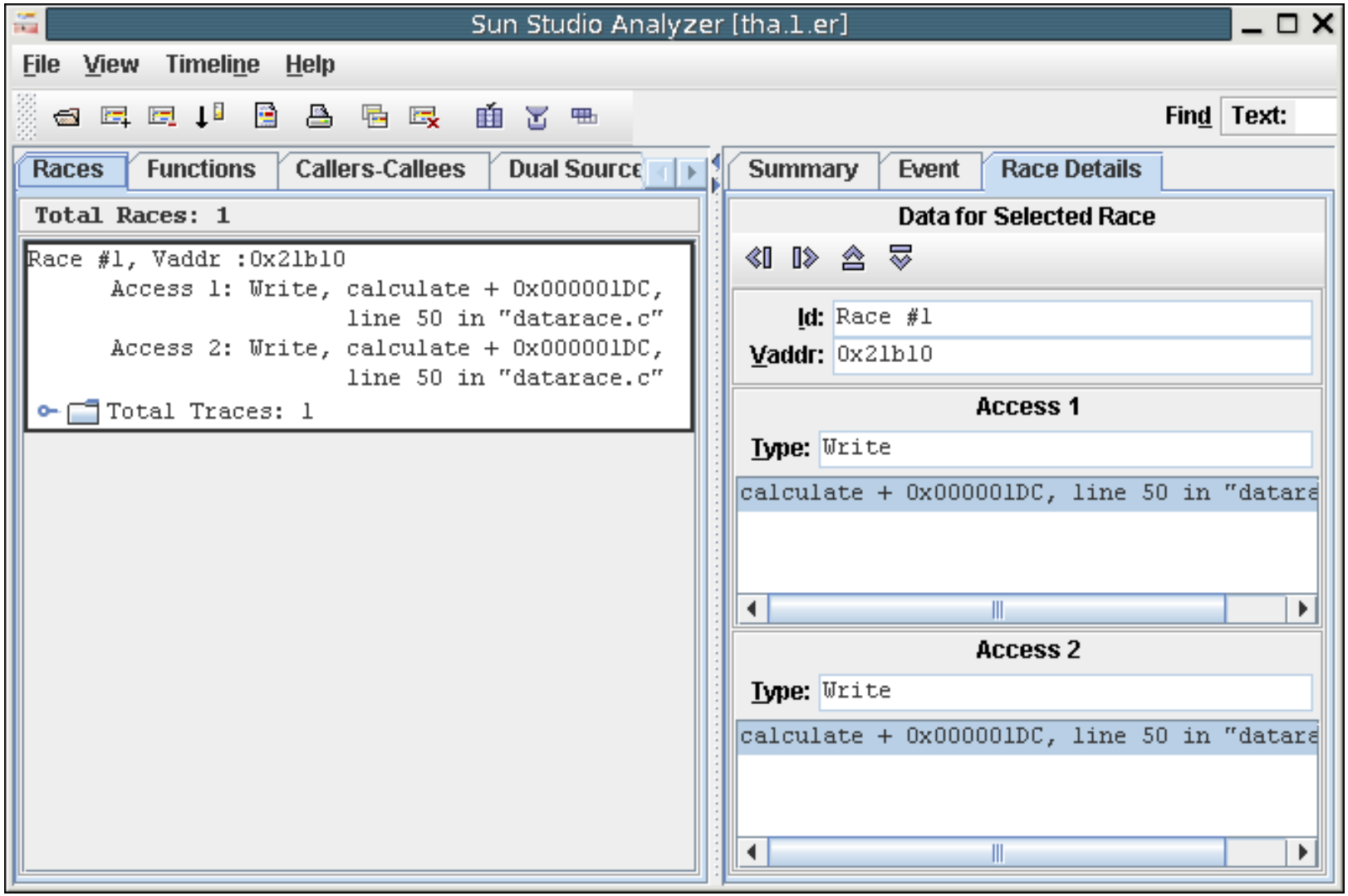
Data races

- Multiple threads reading/writing the same data without exclusive access
- Results in unpredictable behaviour

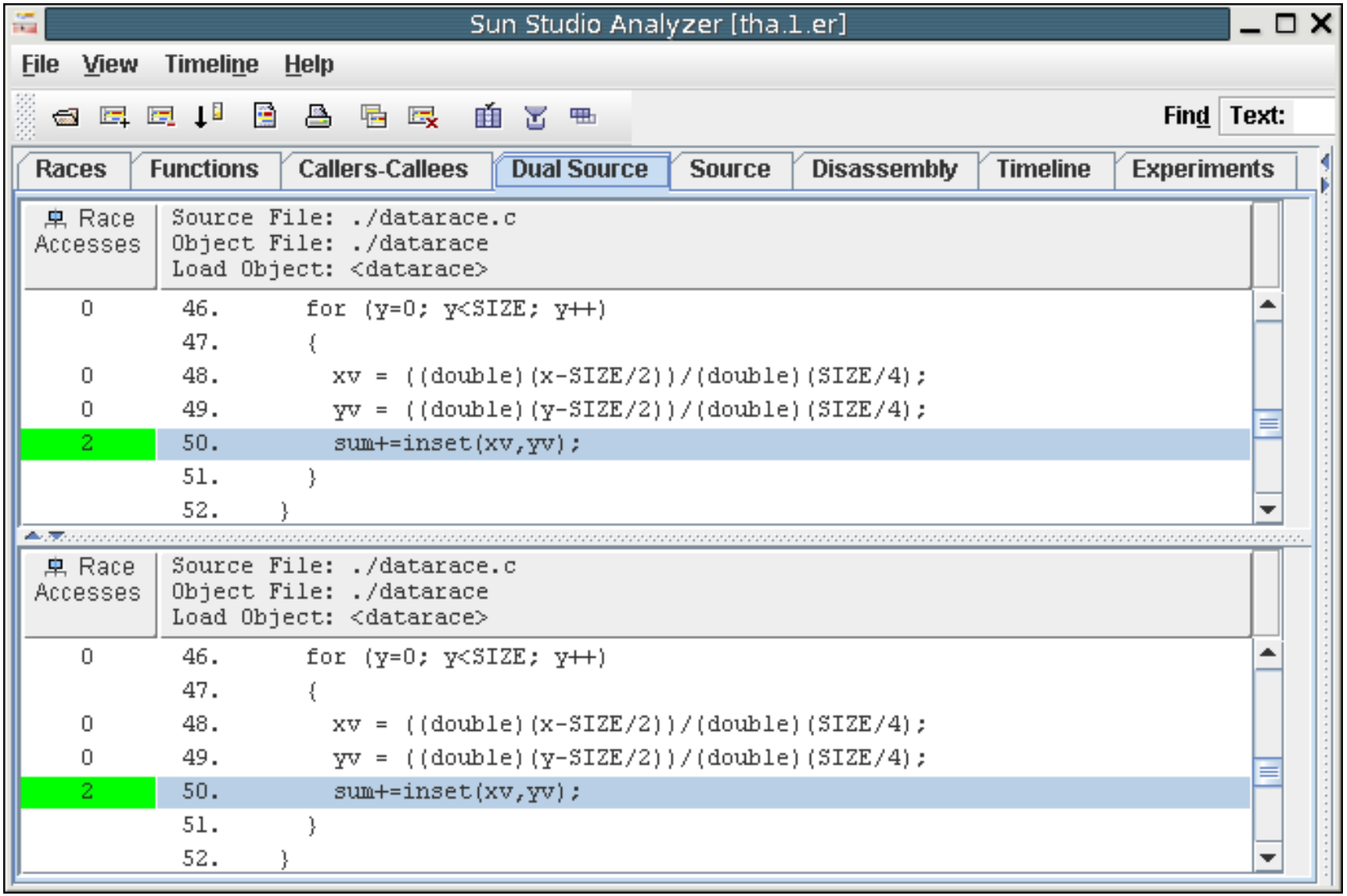
```
$ cc -fast -mt -o race race.c -lpthread

$ datarace
sum = 385158800

$ datarace
sum = 385071679
```



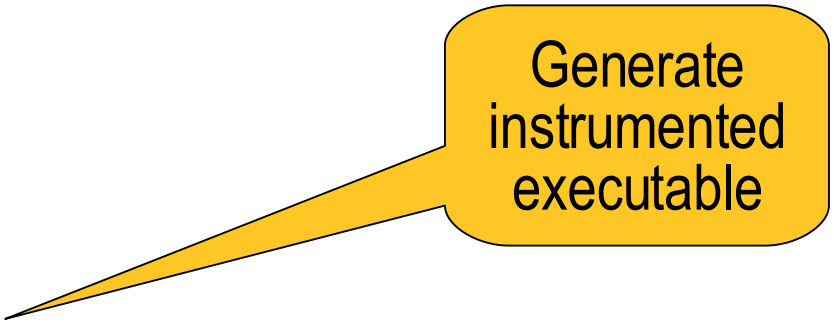
Thread Analyzer



View source code



Notes: Data races




```
$ 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

Summary

- Profile to find hot code
- Use multiple threads
  - Autoparallelisation
  - OpenMP easy to use
  - Pthreads gives more control
- Use the tools in Sun Studio
  - Performance Analyzer for profiling
  - Thread Analyzer for race condition detection

Parallelising serial applications

Darryl Gove
darryl.gove@sun.com
http://blogs.sun.com/d/