Memory Subsystem Profiling with the Sun Studio Performance Analyzer

CScADS, July 20, 2009

Marty Itzkowitz, Analyzer Project Lead
Sun Microsystems Inc.
marty.itzkowitz@sun.com

Outline

- Memory performance of applications
  - > The Sun Studio Performance Analyzer
- Measuring memory subsystem performance
  - > Four techniques, each building on the previous ones
    - – First, clock-profiling
    - – Next, HW counter profiling of instructions
    - – Dive deeper into dataspace profiling
    - – Dive still deeper into machine profiling
      - – What the machine (as opposed to the application) sees
  - > Later techniques needed if earlier ones don't fix the problems
- Possible future directions

No Comment

The Message

- Memory performance is crucial to application performance
  - > And getting more so with time
- Memory performance is hard to understand
  - > Memory subsystems are very complex
    - – All components matter
  - > HW techniques to hide latency can hide causes
- Memory performance tuning is an art
  - > We're trying to make it a science
- The Performance Analyzer is a powerful tool:
  - > To capture memory performance data
  - > To explore its causes

Memory Performance of Applications

- Operations take place in registers
  - All data must be loaded and stored; latency matters
- A load is a load is a load, but
  - Hit in L1 cache takes 1 clock
  - Miss in L1, hit in L2 cache takes ~10-20 clocks
  - Miss in L1, L2, hit in L3 cache takes ~50 clocks
  - Fetch from memory takes ~200-500 clocks (or more)
  - Page-in from disk takes milliseconds
    - Costs are typical; each system is different
- What matters is total stalls in the pipeline
  - If latency is covered, there's no performance cost

Why Memory Performance is Hard

- SW developers know code, algorithms, data structures
  - What the HW does with them is magic
    - Many, if not most, SW developers can't even read assembler
- HW engineers know instruction, address streams
  - How the SW generates them is magic
- HW performance optimizations further confuse the issue
- Difficulty lies in bridging the gap
  - Get data to show HW perspective to SW developers
- The rest of this talk will show how we do so

Memory Performance Problems

- Some causes of memory performance problems:
  - > Initial cache miss, capacity misses
    - – Layout and padding; lack of prefetch
  - > Conflict cache misses within a thread
    - – Striding through arrays
  - > Coherence misses across thread
    - – Sharing: unavoidable misses
    - – False sharing: avoidable miss, not a real conflict
      - – Threads refer to different fields in same cache line
      - – Different processes use same VA for different PA's
  - > Cache and Page coloring
    - – Mappings from addresses to cache lines

The Sun Studio Performance Analyzer

- Integrated set of tools for performance measurement
  - > Data collection
  - > Data examination
- Many types of data:
  - > Clock-profiling, HW counter profiling, ...
  - > Special support for OpenMP, MPI, Java
- Common command-line and GUI interface for all
- Available on SPARC and X86, Solaris and Linux
  - > It's FREE!
- You've seen it before....

Clock Profiling

- Periodic statistical sampling of callstacks
  - `collect -p <interval> target`
    - Note: many other tools do clock-profiling, too
- Shows expensive functions, instructions
  - Is it the heart of the computation, or is it stalled?
  - If it's stalled,
    - Is it stalled waiting for a previous operation?
    - Is it stalled waiting for a load?
    - Is it stalled trying to do a store?
  - Can only guess with clock profiling

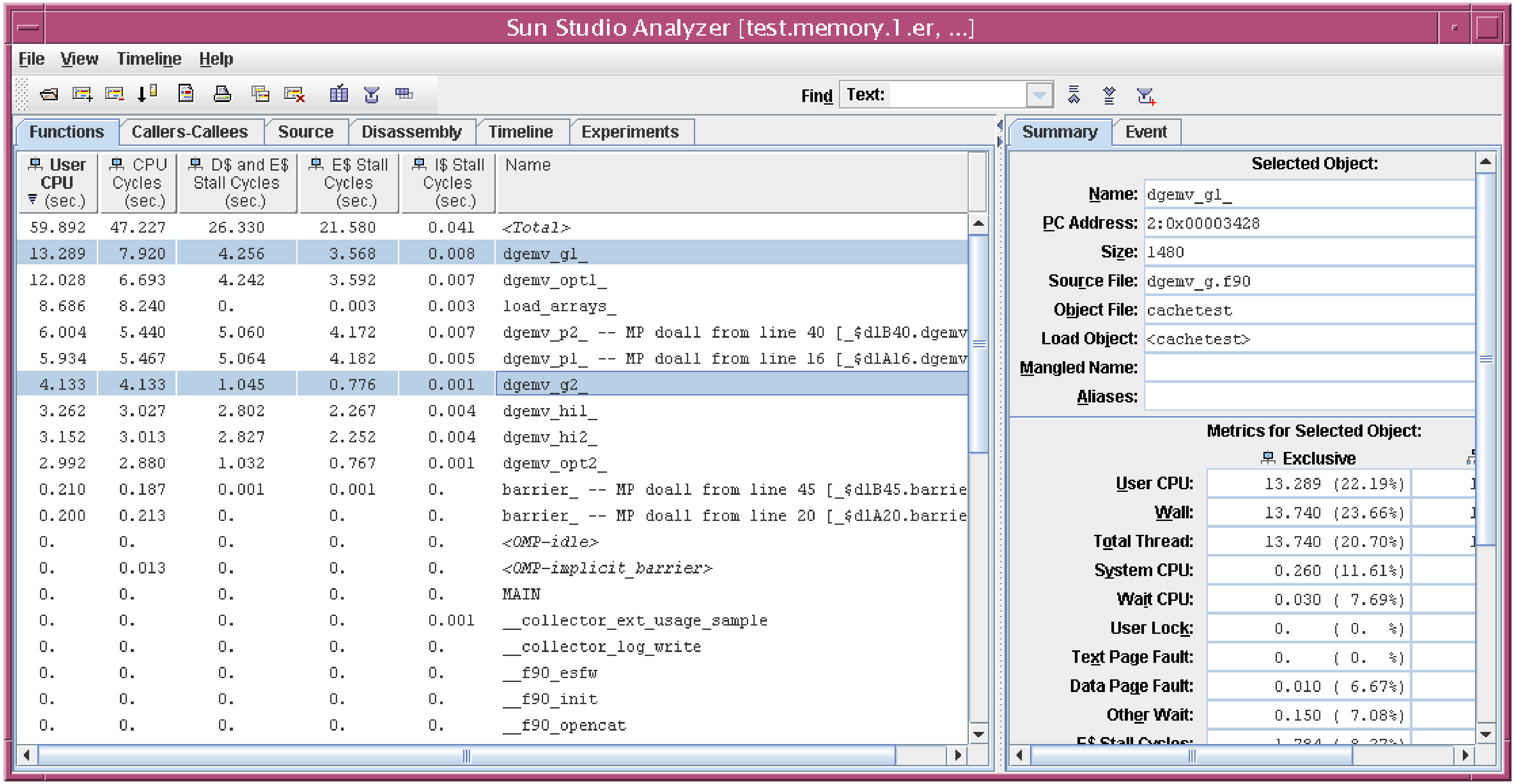
Measuring Memory Costs

- Need better data to understand more
  - See: Zagha, *et.al.*, SC `96
- Use HW counters to trigger sampling
  - **`collect -h <cntr1>,<val1>,<cntr2>,<val2>,...`**
    - As many counters as chip supports
    - **`collect`** with no arguments prints list for that machine
- Collect counter name, overflow value, callstack
  - Cache misses/references, TLB misses, instructions, ...
  - Cycles, L1-Cache stalls, L2-Cache stalls, ...
    - Measured in cycles; convertible to time
- Shows memory costs based on the counters

Memory Performance Example

- Test code: 8 copies of vector-matrix multiply
  - 8 functions named: **`dgemv_`*`<opt-flag><order>`***
    - Same computation, different performance
  - Two loop orders
    - Row, column and column,row
    - ***`<order>`*** = **`1`**, **`2`**
  - Four optimization levels
    - Compile with **`-g`**, **`-O`**, **`-fast`**, and **`-fast -autopar`**
    - ***`<opt-flag>`*** = **`_g`**, **`_opt`**, **`_hi`**, and **`_p`**

Detailed Memory Performance



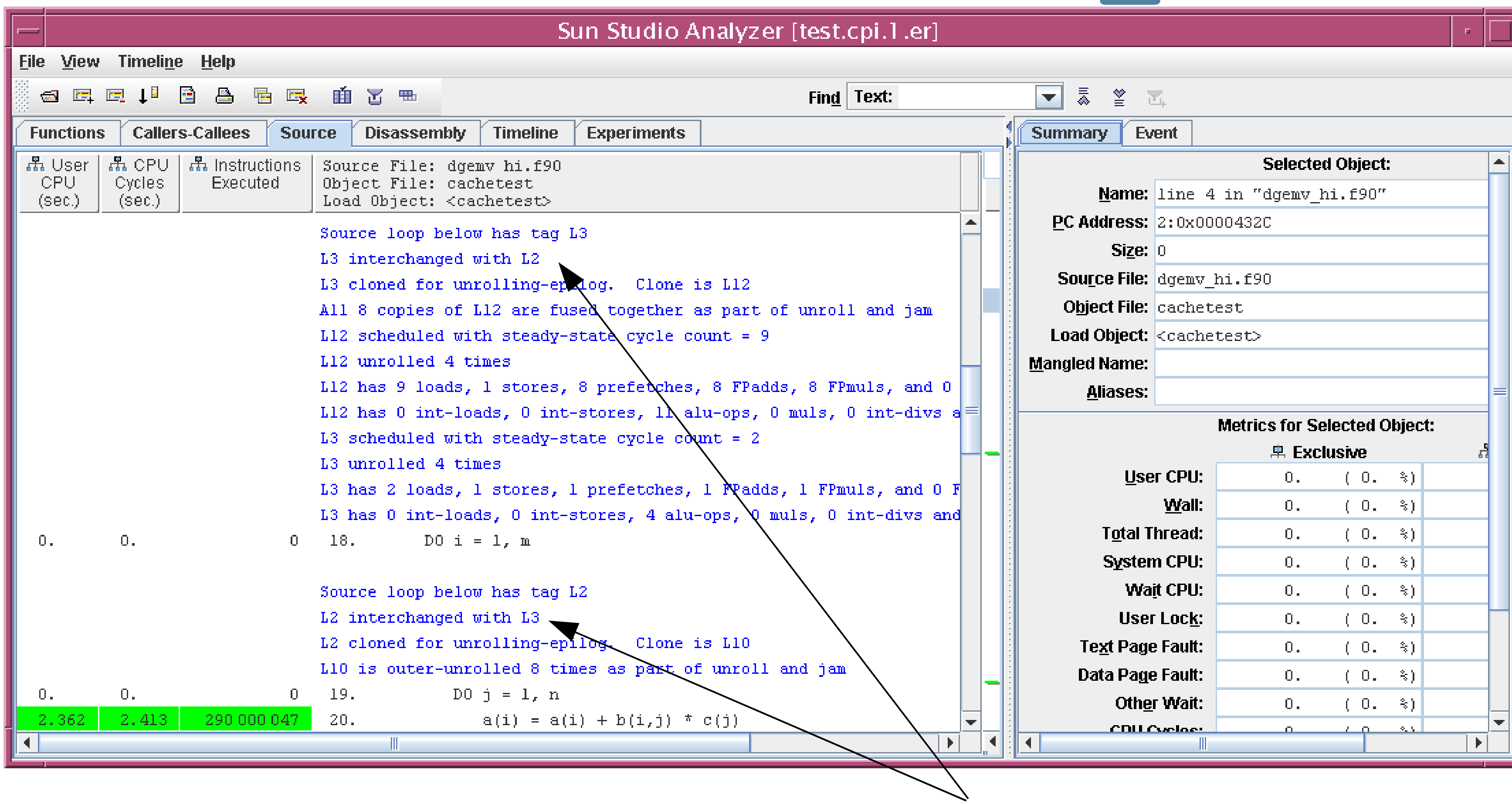
Separate out costs of the various caches

Two experiments, combined in Analyzer

Memory Performance Problems

- Data shows where in program problems occur
  - High cache misses, TLB misses
    - Does not show why
- Cause is striding through memory
  - Clue from differences between loop order versions
  - In this example, the compiler can diagnose
    - Studio compilers generate commentary to say what they did
      - See next slide
- In general, diagnosing these problems is hard
  - This one is easy – other cases are more difficult

Annotated Source of `dgemv_hi1`



Loop interchange – compiler knows best order of loops

Compiler commentary from **`-fast`** compilation

Dive Deeper

- We understand program instructions, not data
- Want better performance data
  - The data *addresses* that trigger the problems
  - The data objects that trigger the problems
    - *i.e.*, Source references
- Hard to get data reference address:
  - HW counters skid past triggering instruction
    - Interrupt PC != Trigger PC
    - Current registers may not reflect state at time of event
- Solution: Dataspace profiling
  - Built on top of HW counter profiling

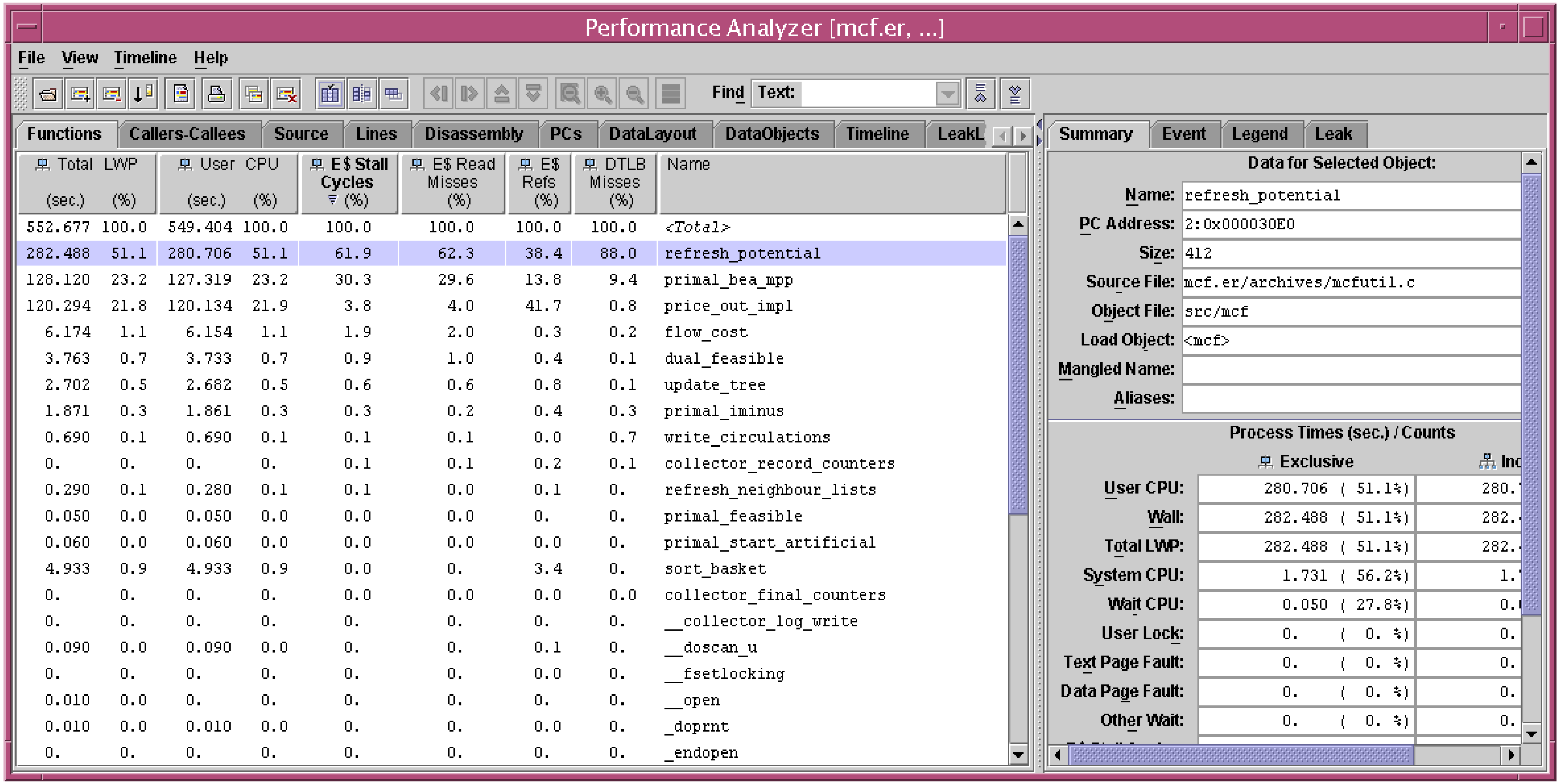
Dataspace Profiling Technology

- Extend HW counter to capture more data
  - `collect -h +<cntr1>,<val1>,+<cntr2>,<val2>,...`
    - **+** sign in front of counter name
- Causes backtracking at HW profile event delivery
  - Capture trigger PC (might fail)
  - Capture virtual and physical data addresses (might fail)
    - Track register changes that might affect address
  - Post-process to see if branch-target crossed
    - Typically, 95% of backtracking succeeds
- SPARC-only functionality, alas
  - Backtracking not possible on x86/x64
    - But instruction sampling can extend it to x86/x64

Dataspace Profiling Example

- **`mcf`** from SPEC cpu2000 benchmark suite
  - > Single depot vehicle scheduler; network simplex
    - – Single-threaded application
- Collect two experiments
  - > **`-p on -h +ecstall,lo,+ecrm,on`**
  - > **`-p off -h +ecref,on,+dtlbm,on`**
- Combine in Analyzer
- See Itzkowitz, *et.al.*, SC|03 for details

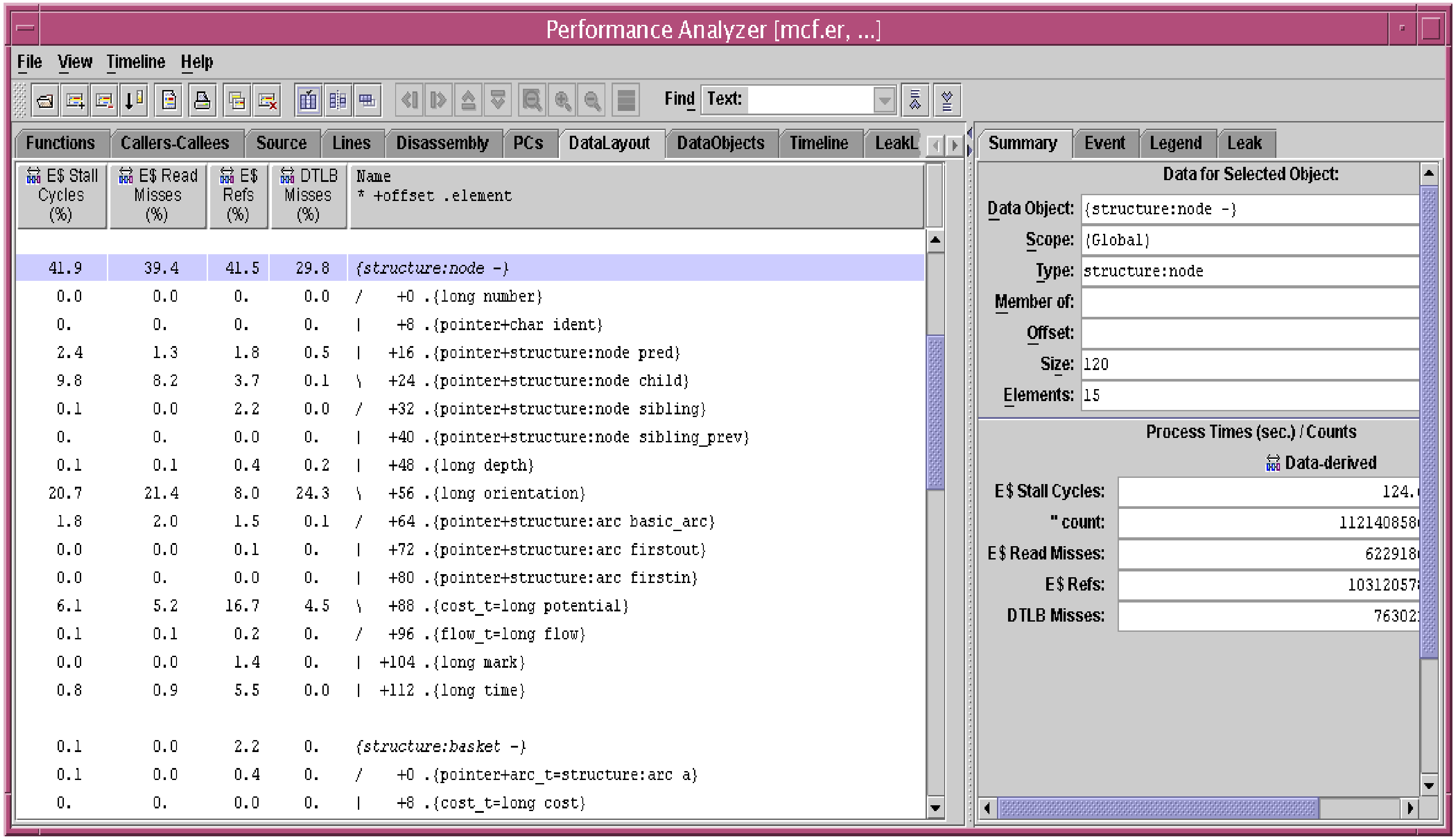
Dataspace Profiling: Function List



Total LWP (sec.)	Total LWP (%)	User CPU (sec.)	User CPU (%)	E$ Stall Cycles (%)	E$ Read Misses (%)	E$ Refs (%)	DTLB Misses (%)	Name
552.677	100.0	549.404	100.0	100.0	100.0	100.0	100.0	<Total>
282.488	51.1	280.706	51.1	61.9	62.3	38.4	88.0	refresh_potential
128.120	23.2	127.319	23.2	30.3	29.6	13.8	9.4	primal_bea_mpp
120.294	21.8	120.134	21.9	3.8	4.0	41.7	0.8	price_out_impl
6.174	1.1	6.154	1.1	1.9	2.0	0.3	0.2	flow_cost
3.763	0.7	3.733	0.7	0.9	1.0	0.4	0.1	dual_feasible
2.702	0.5	2.682	0.5	0.6	0.6	0.8	0.1	update_tree
1.871	0.3	1.861	0.3	0.3	0.2	0.4	0.3	primal_iminus
0.690	0.1	0.690	0.1	0.1	0.1	0.0	0.7	write_circulations
0.	0.	0.	0.	0.1	0.1	0.2	0.1	collector_record_counters
0.290	0.1	0.280	0.1	0.1	0.0	0.1	0.	refresh_neighbour_lists
0.050	0.0	0.050	0.0	0.0	0.0	0.	0.	primal_feasible
0.060	0.0	0.060	0.0	0.0	0.0	0.0	0.	primal_start_artificial
4.933	0.9	4.933	0.9	0.0	0.	3.4	0.	sort_basket
0.	0.	0.	0.	0.0	0.0	0.0	0.0	collector_final_counters
0.	0.	0.	0.	0.	0.	0.	0.	__collector_log_write
0.090	0.0	0.090	0.0	0.	0.	0.1	0.	__doscan_u
0.	0.	0.	0.	0.	0.	0.0	0.	__fsetlocking
0.010	0.0	0.	0.	0.	0.	0.	0.	__open
0.010	0.0	0.010	0.0	0.	0.	0.0	0.	_doprnt
0.	0.	0.	0.	0.	0.	0.	0.	_endopen

Which functions have memory performance issues

Dataspace Profiling: Data Layout



Show costs against Data Structure Layout, not code

Dataspace Example Conclusions

- Hot references all are to **`node`** and **`arc`** fields
- Structures not well-aligned for cache
  > Need to pad to cache-line boundary
  > Reorganize structures to put hot fields on same line
    – Note: reorganizing might move hot fields, but not improve perf.
- High TLB misses imply need for large heap pages
- These changes led to ~21% improvement
  > But not following SPEC cpu2000 rules
    – That does not matter for real codes, of course

Dive Still Deeper

- We understand instructions and data, but not machine
  - So far, problems have been in a single thread
- Use same data to explore interactions among threads
  - Sample questions to answer:
    - Which cache lines are hot?
      - Is usage uniform across lines, or is there one very hot line?
    - Which threads refer to those lines?
    - Which addresses are being referred to by which threads?

Advanced Diagnostic Strategy

- Iterative analysis of the data:
  - > Slice and dice data into sets of “objects”
    - – Cache lines, pages, TLB entries, CPUs, ...
    - – Threads, Processes, Time intervals
  - > Find the hot objects of one set
  - > Filter to include data only for those hot objects
  - > Look at other types of objects to see why
    - – It is non-trivial to know which ones to look at
  - > Repeat as needed

Advanced Diagnostic Techniques

- Collect Dataspace profiling data
  > `collect -h +<cntr1>,<val1>,+<cntr2>,<val2>,...`
- Collect over all threads and processes
- Slice into "Index Object" or "Memory Object" sets
  > Each set has formula for computing an index from records
  > Analyzer has a Tab for each object set
  > Each Tab shows metrics for the objects in each set
    - *e.g.*, Threads, L2-cache lines, ...

Memory and Index Objects

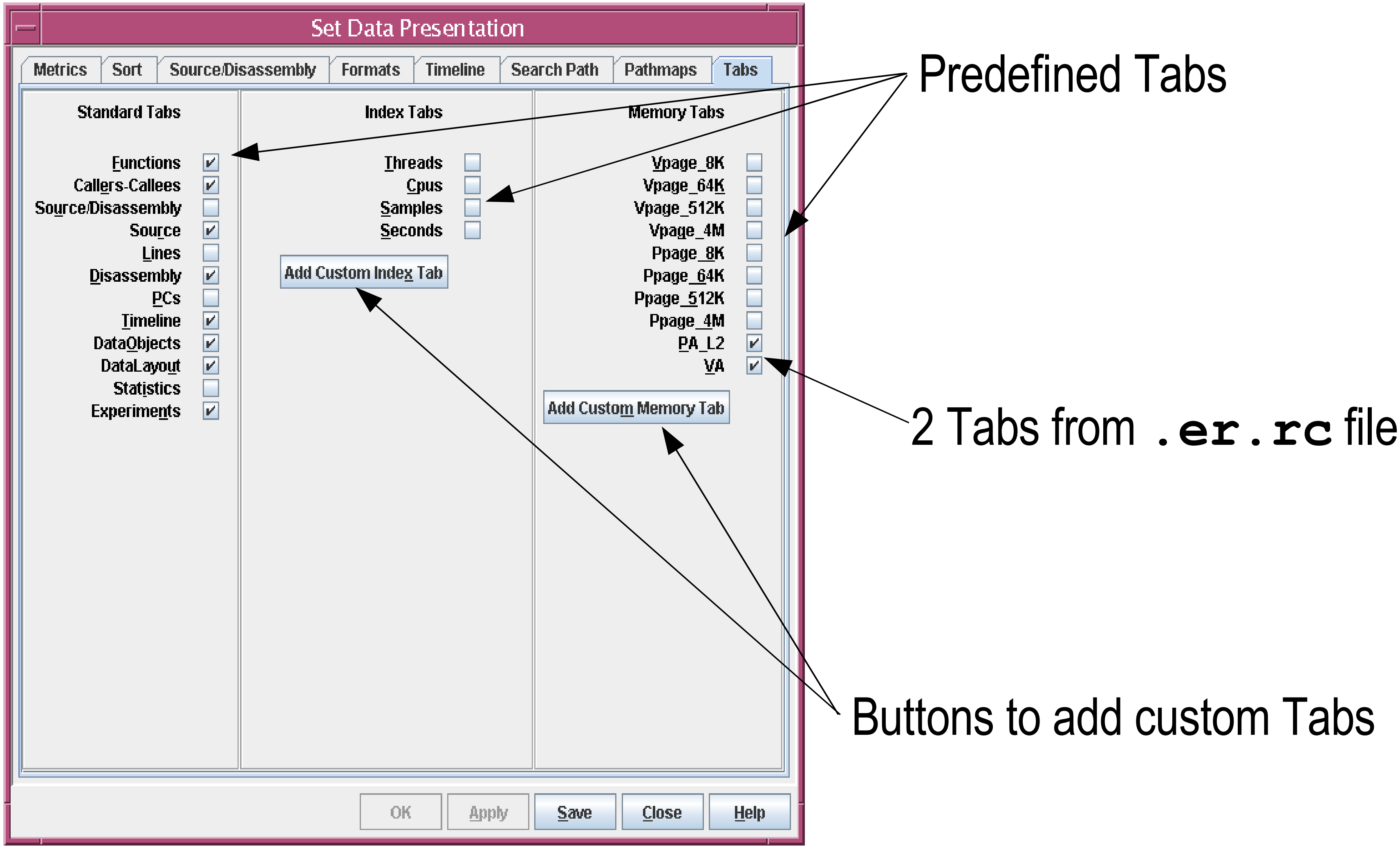
- Index Objects: formula does not use VADDR or PADDR
  - Formula fields present in all records
  - Can be used for all data
  - Some are predefined
- Memory Objects: formula uses VADDR or PADDR
  - Address fields present only dataspace records
  - Definitions depend on the specific physical machine
    - Cache structure, page size, TLB organization
    - Not yet captured automatically, but could be
- Define in `.er.rc` file, based on specific machine

Sample Object Definitions

- Each definition uses one (or more) fields
  - Thread
    - **`indxobj_define Threads THRID`**
  - Virtual address
    - **`mobj_define VA VADDR`**
  - L2 cache line
    - **`mobj_define PA_L2 (PADDR&0x7ffc0)>>6`**
- Can be a lot more complicated
  - *e.g.*, Niagara-2 level-2 data cache line set

```
mobj_define UST2_L2DCacheSet \
(((((PADDR>>15)^PADDR)>>9)&0x1f0) | \
((((PADDR>>7)^PADDR)>>9)&0xc) | \
((PADDR>>9)&3))
```

Displaying Objects in Analyzer Tabs

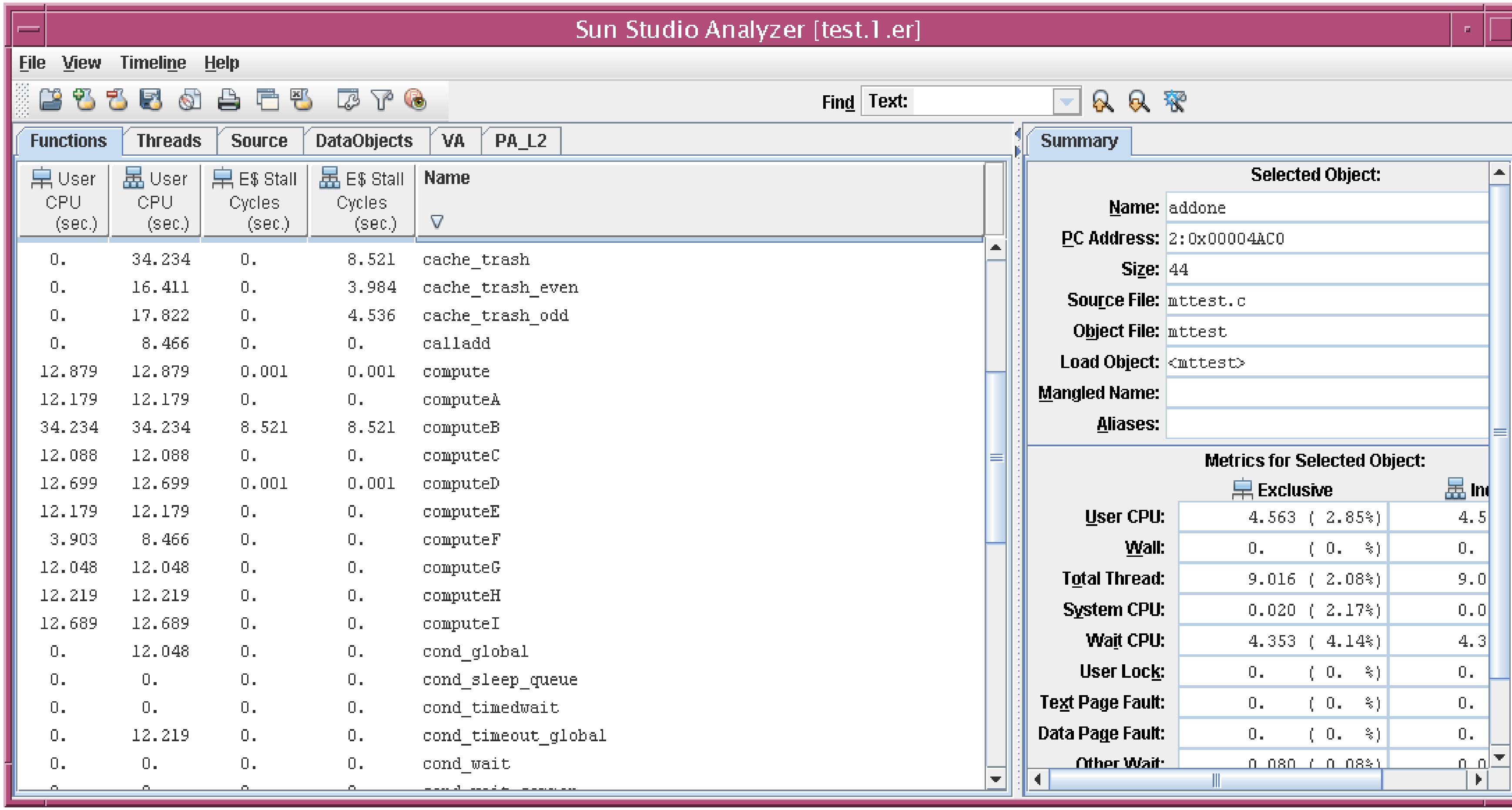


Example: `mttest`

- Analyzer test code
  - > Organized as series of tasks
    - – Each task queues 4 blocks, spawns 4 threads
    - – Threads synchronize differently for each task
    - – Each thread calls one of the **`compute*`** functions for its block
- We will explore why **`computeB`** is different
  - > Takes almost 3X as much time as the others
- Collect experiment:
  - > **`collect -p on -h +ecstall,on`**

Demo

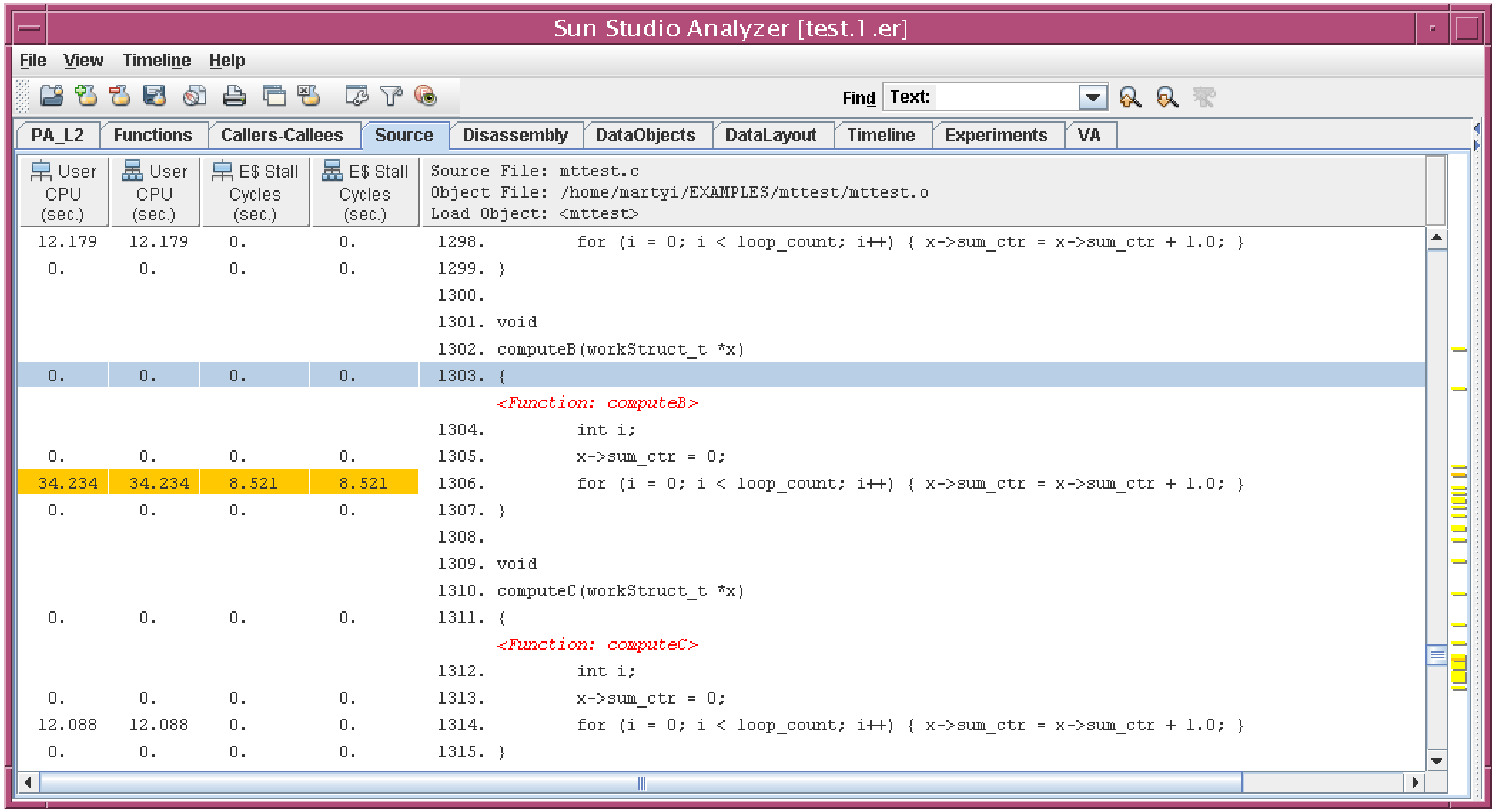
Function List



User CPU (sec.)	User CPU (sec.)	E$ Stall Cycles (sec.)	E$ Stall Cycles (sec.)	Name
0.	34.234	0.	8.521	cache_trash
0.	16.411	0.	3.984	cache_trash_even
0.	17.822	0.	4.536	cache_trash_odd
0.	8.466	0.	0.	calladd
12.879	12.879	0.001	0.001	compute
12.179	12.179	0.	0.	computeA
34.234	34.234	8.521	8.521	computeB
12.088	12.088	0.	0.	computeC
12.699	12.699	0.001	0.001	computeD
12.179	12.179	0.	0.	computeE
3.903	8.466	0.	0.	computeF
12.048	12.048	0.	0.	computeG
12.219	12.219	0.	0.	computeH
12.689	12.689	0.	0.	computeI
0.	12.048	0.	0.	cond_global
0.	0.	0.	0.	cond_sleep_queue
0.	0.	0.	0.	cond_timedwait
0.	12.219	0.	0.	cond_timeout_global
0.	0.	0.	0.	cond_wait

Alphabetical (name) sort – note **`ComputeB`** vs. others

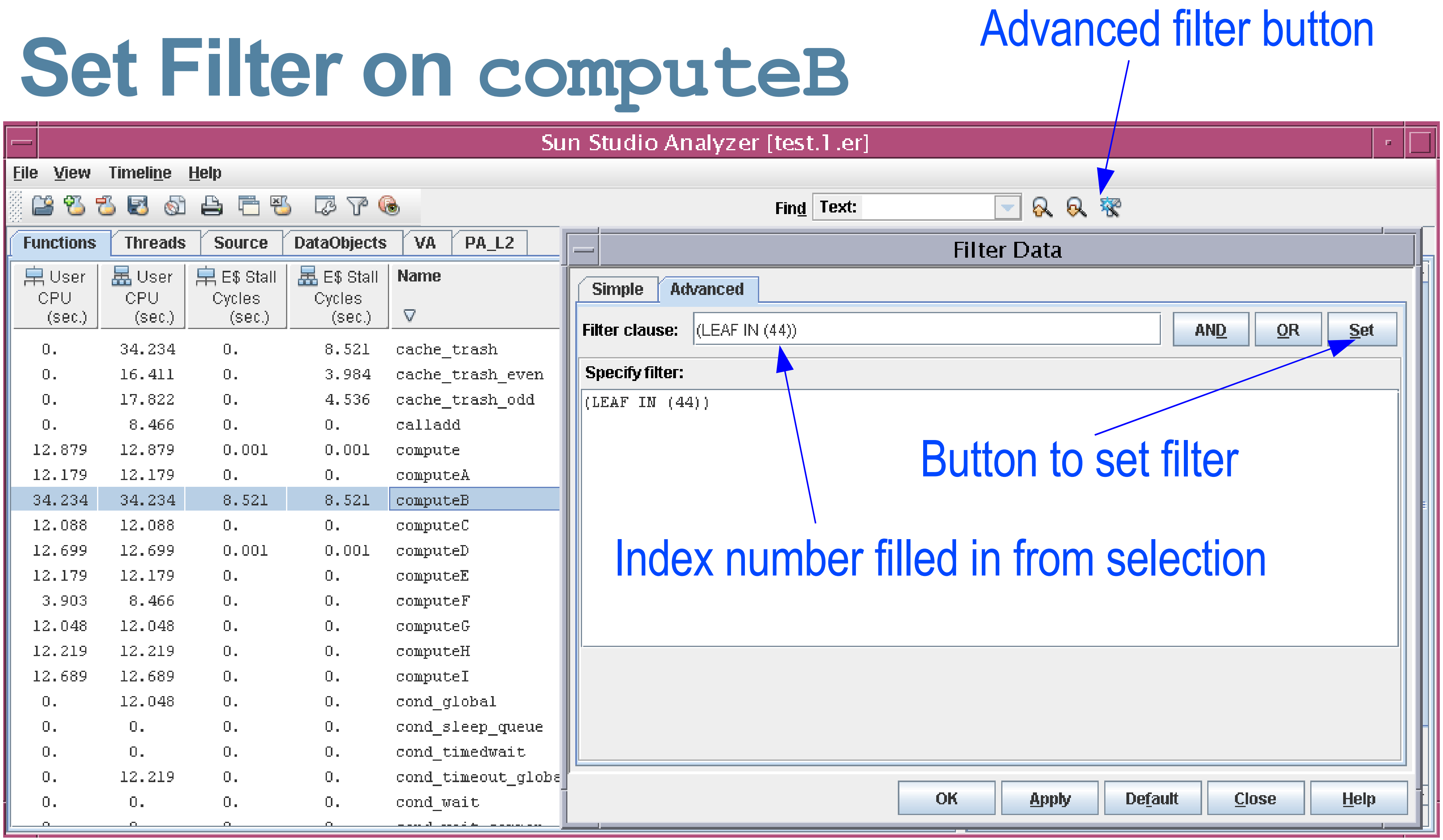
Source for `compute*`



Lines 1298,1306,1314 are *identical*

But they perform *differently*

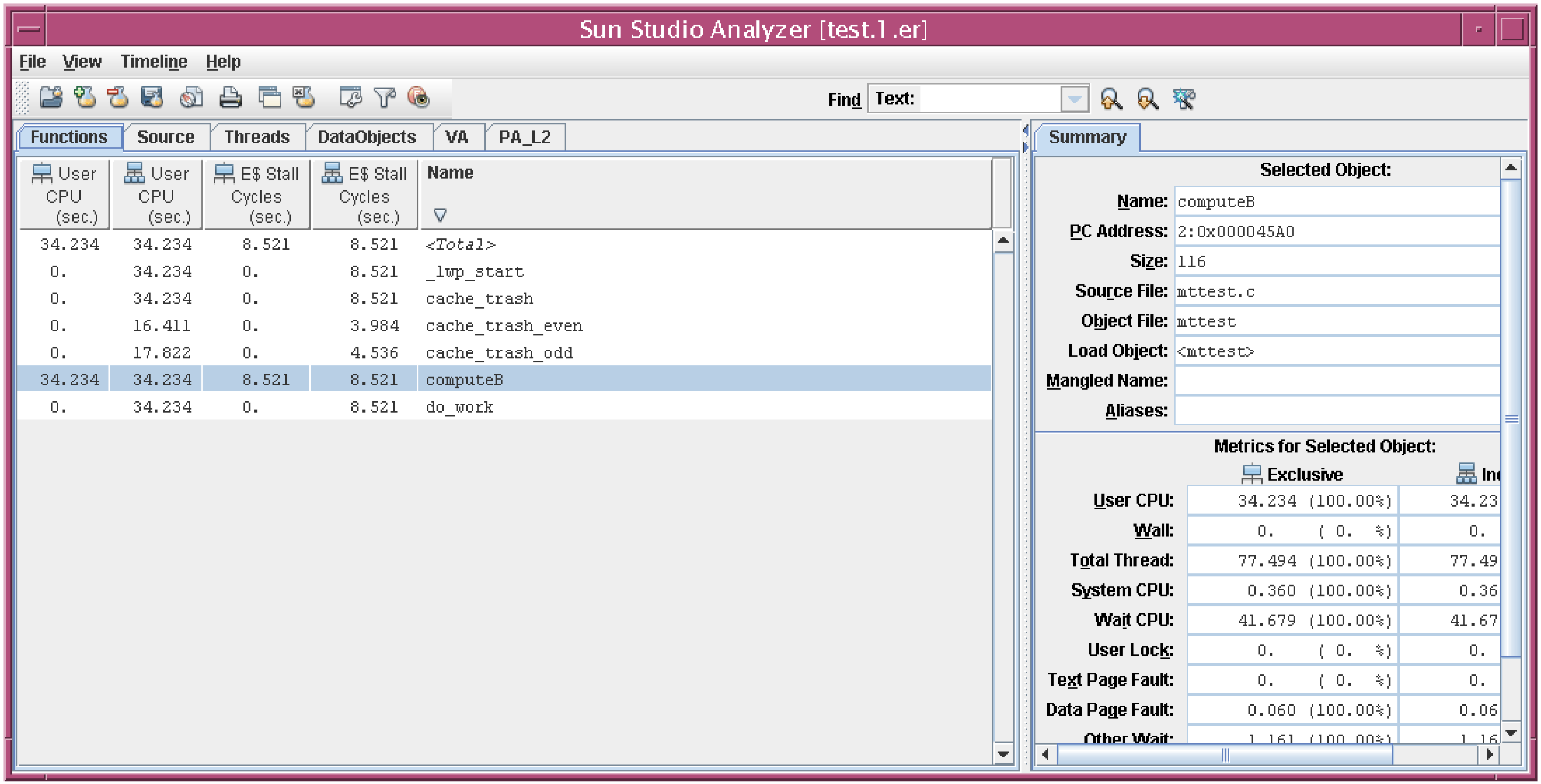
Set Filter on `computeB`



Filter to show only those events with `computeB` as leaf

Sorry syntax is so ugly

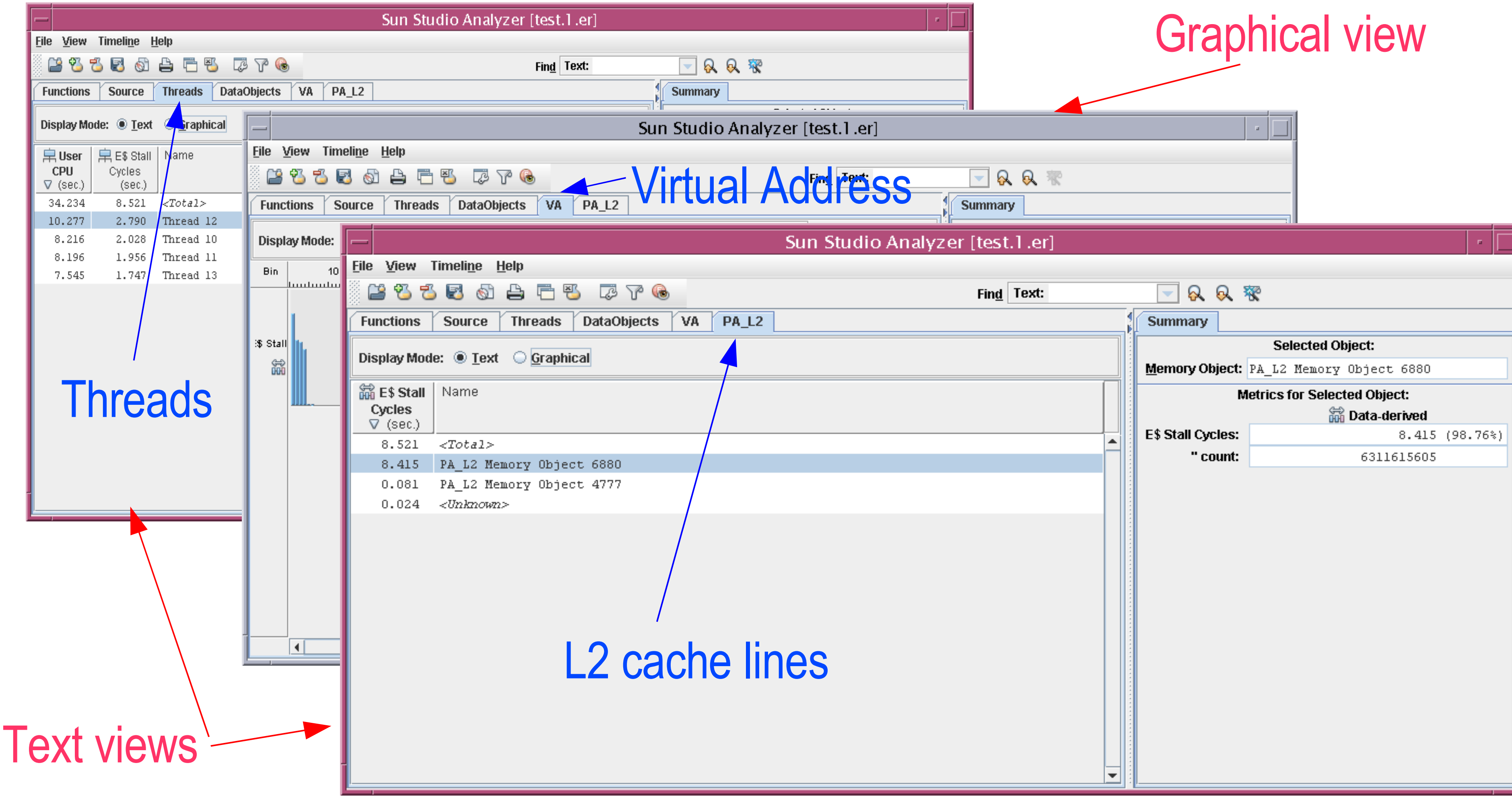
Function List Filtered on `computeB`



Function list only shows callers of **`computeB`**

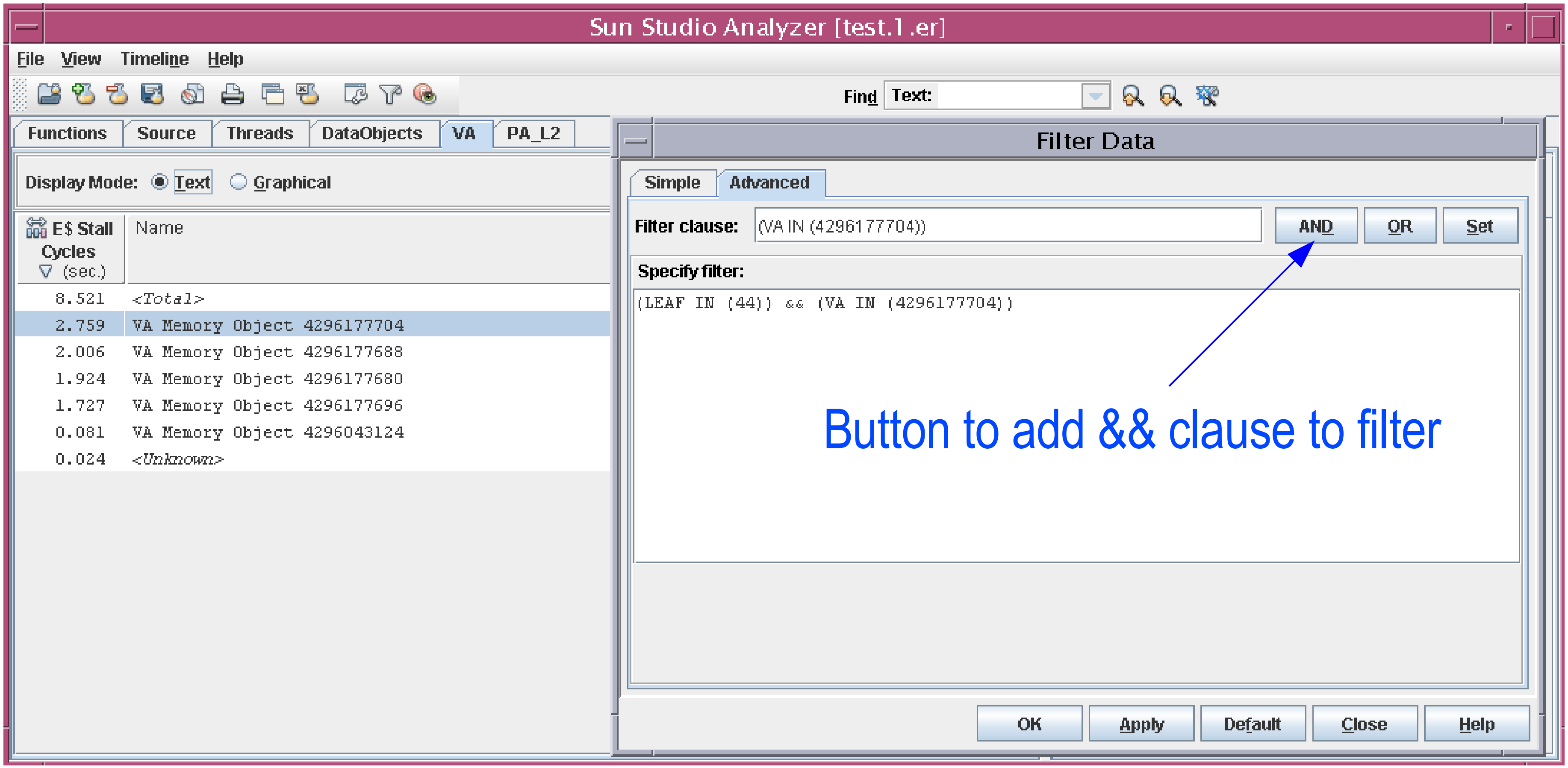
(In this example **`computeB`** is a leaf function)

Threads, VA, and PA_L2 Tabs



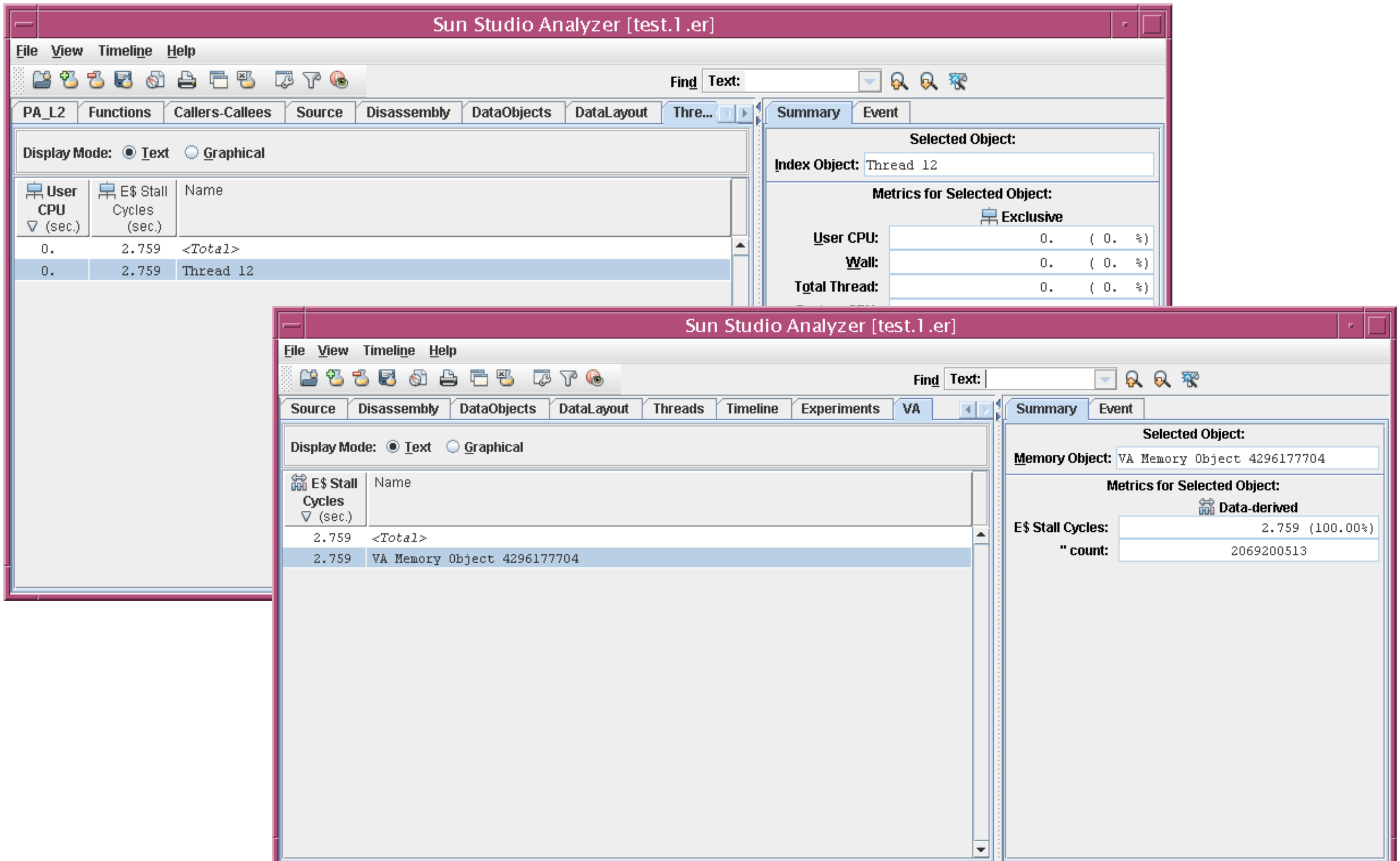
Four threads, four virtual addresses, one cache line

Add VA Filter for One Address



Will show only events in **`computeB`** referring to that one address

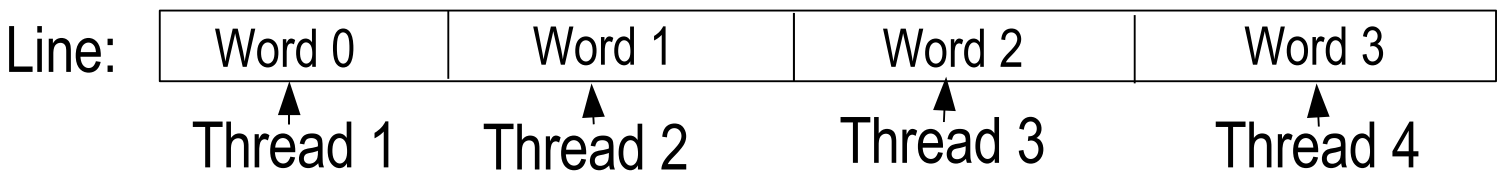
Look at VA and Threads again



One thread per address; true of all four addresses

Diagnosis

- Most cache misses are on a single cache line
  - Four threads get the misses
  - Four addresses are referenced
  - Each thread references only one virtual address



  - Write from one thread invalidates line for all others
- Classic manifestation of false sharing
  - A notoriously difficult problem to spot
  - In true sharing, multiple threads refer to each address

Potential Future Development

- Enhance data collection
  - > Support x86/x64 with instruction-based sampling
    - *Set up working group at this meeting?*
  - > Integrate configuration capture with data collection
- Improve the GUI and navigation
  - > Improve filtering grammar and syntax
  - > Other usability improvements
- Develop tuning strategy
  - > Systematic procedures for exploring problems

For more information

External Sun Studio Website

http://developers.sun.com/sunstudio/

External Sun Studio Performance Tools Website

http://developers.sun.com/sunstudio/overview/topics/analyzer_index.html

SC'96 paper on HW Counter Profiling

http://portal.acm.org/citation.cfm?id=369028.369059&coll=portal&dl=ACM&CFID=33541981&CFTOKEN=50518735

SC|03 paper on Dataspace Profiling

http://www.sc-conference.org/sc2003/paperpdfs/pap182.pdf

Solaris Application Programming by Darryl Gove

http://www.sun.com/books/catalog/solaris_app_programming.xml

Acknowledgments

- Nicolai Kosche, PAE
  - > Driving force for dataspace profiling enhancements
  - > Developed advanced techniques
  - > Invented term “DProfile” to refer to those techniques
- The Sun Studio Peformance Analyzer team
  - > Made it all work

Thank You

Marty Itzkowitz, Analyzer Project Lead
Sun Microsystems Inc.
marty.itzkowitz@sun.com