
SystemC and Tunathon

Jerome Verbeke & Dominic Paulraj

Market Development Engineering &
Compiler Analysis and Tuning Group
Sun Microsystems, Inc.
jerome.verbeke@sun.com



Outline

- Tunathon
- Baseline performance
- Performance Analysis
- Performance Tuning
- Future work



Tunathon, what it is:

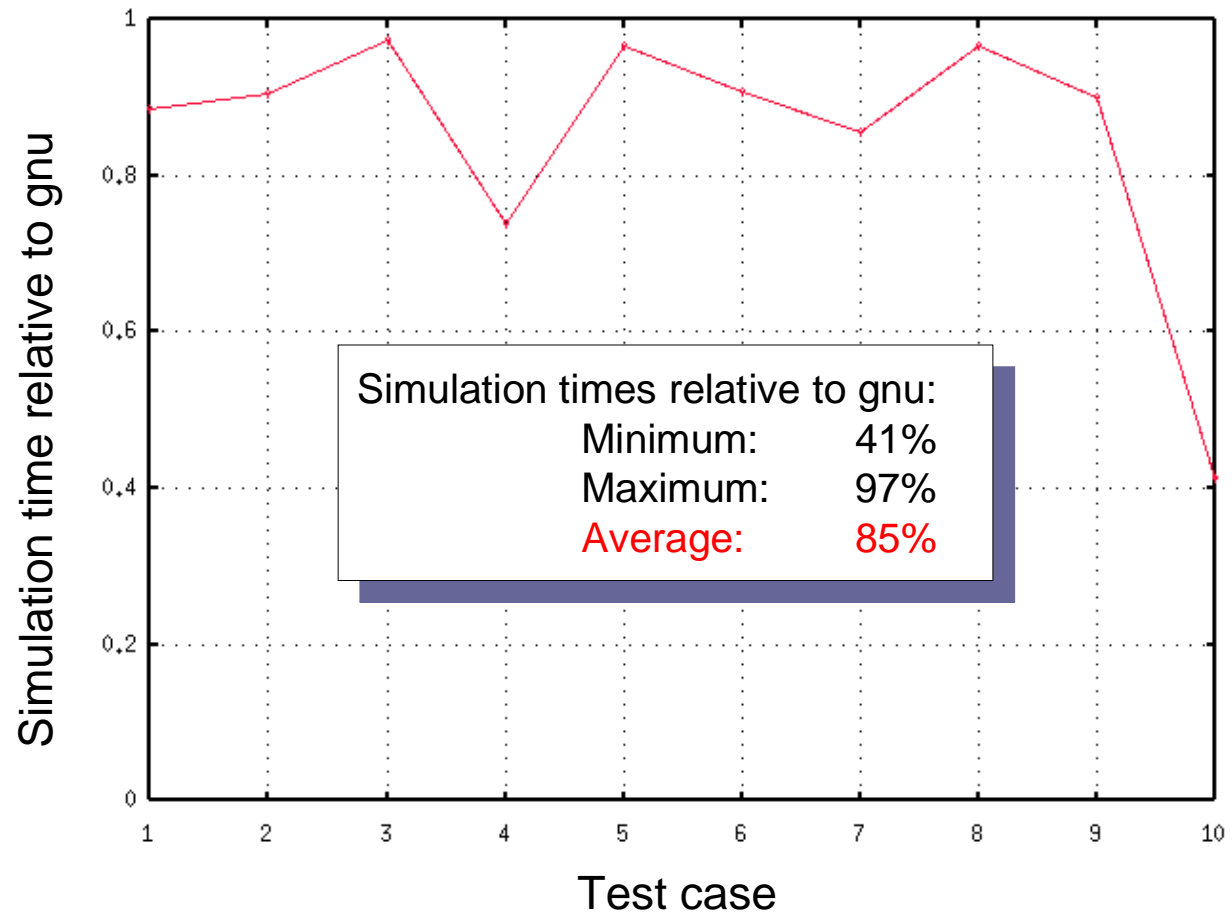
The "Tunathon" is a period of intense effort by Sun that brings several divisions together to make applications run faster on Sun platforms. The first Tunathon was in early 2001, and showed that large gains were possible as the result of the cross-divisional teamwork. Tunathon II will extend the success of Tunathon I by making sure that the gains can be used by ISVs and will work for a broad spectrum of workloads and platforms.

The Tunathon-II started mid-January 2002. Early results will be shown during this presentation.



Baseline Performance

Comparison of gnu 2.95 and Forte Developer 6 update 2 C++ compilers at default -xO3 opt level



Baseline Performance: Test Case X

- Description:

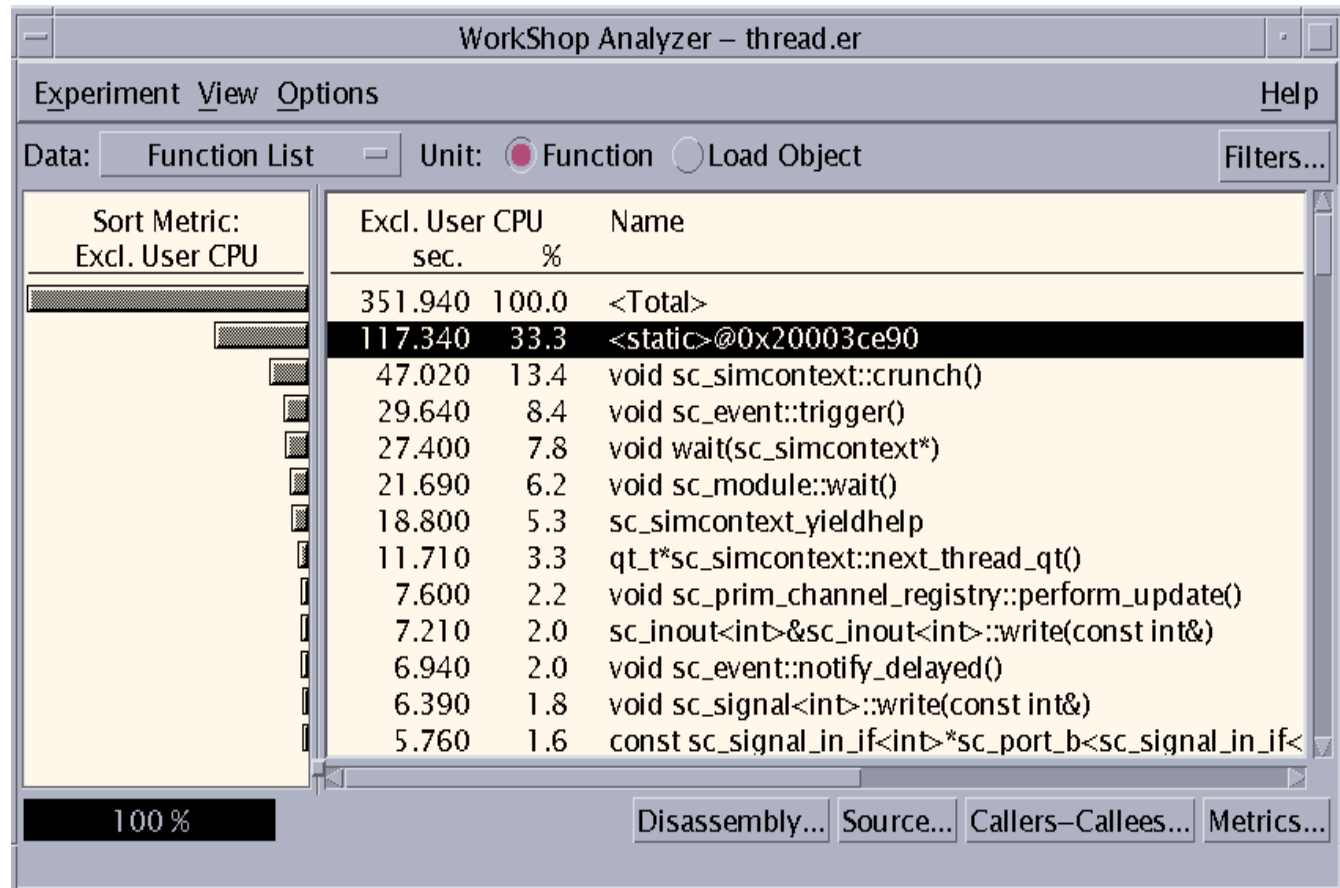
Test case X contains a 1000-SC_THREAD long chain connected serially using signals. A token is sent at one end. Each process copies the token from its input to its output. This triggers the next process and so on until the token comes out at the end of the chain. The trip time is measured for 100000 tokens. This test case is intended to measure basic kernel and thread speed.

- Baseline simulation time: 177.12 sec.



Performance Analysis for Test Case X

- Unknown function `<static>@0x20003ce90` is taking >30% of simulation time



Performance Analysis: QuickThread library

- Assembler file `sparc.s` from the QuickThread library does not contain the necessary `.type` and `.size` tags for proper identification

```
/* #include <machine/trap.h> */
```

```
.text  
.align 4  
.global _qt_blocki  
.type _qt_blocki,2  
.global _qt_block  
.type _qt_block,2  
.global _qt_abort  
.type _qt_abort,2  
.global _qt_start  
.type _qt_start,2  
.global _qt_vstart  
.type _qt_vstart,2
```

(...)

```
/* Restore callee-save regs. The kwsa  
// is on this stack, so offset all  
// loads by sizeof(kwsa), 64 bytes. */  
ldd [%sp+ 0+64], %l0  
ldd [%sp+ 8+64], %l2  
ldd [%sp+16+64], %l4  
ldd [%sp+24+64], %l6  
ldd [%sp+32+64], %i0  
ldd [%sp+40+64], %i2  
ldd [%sp+48+64], %i4  
ldd [%sp+56+64], %i6  
ld [%sp+64+64], %o7 /* Restore return pc. */
```

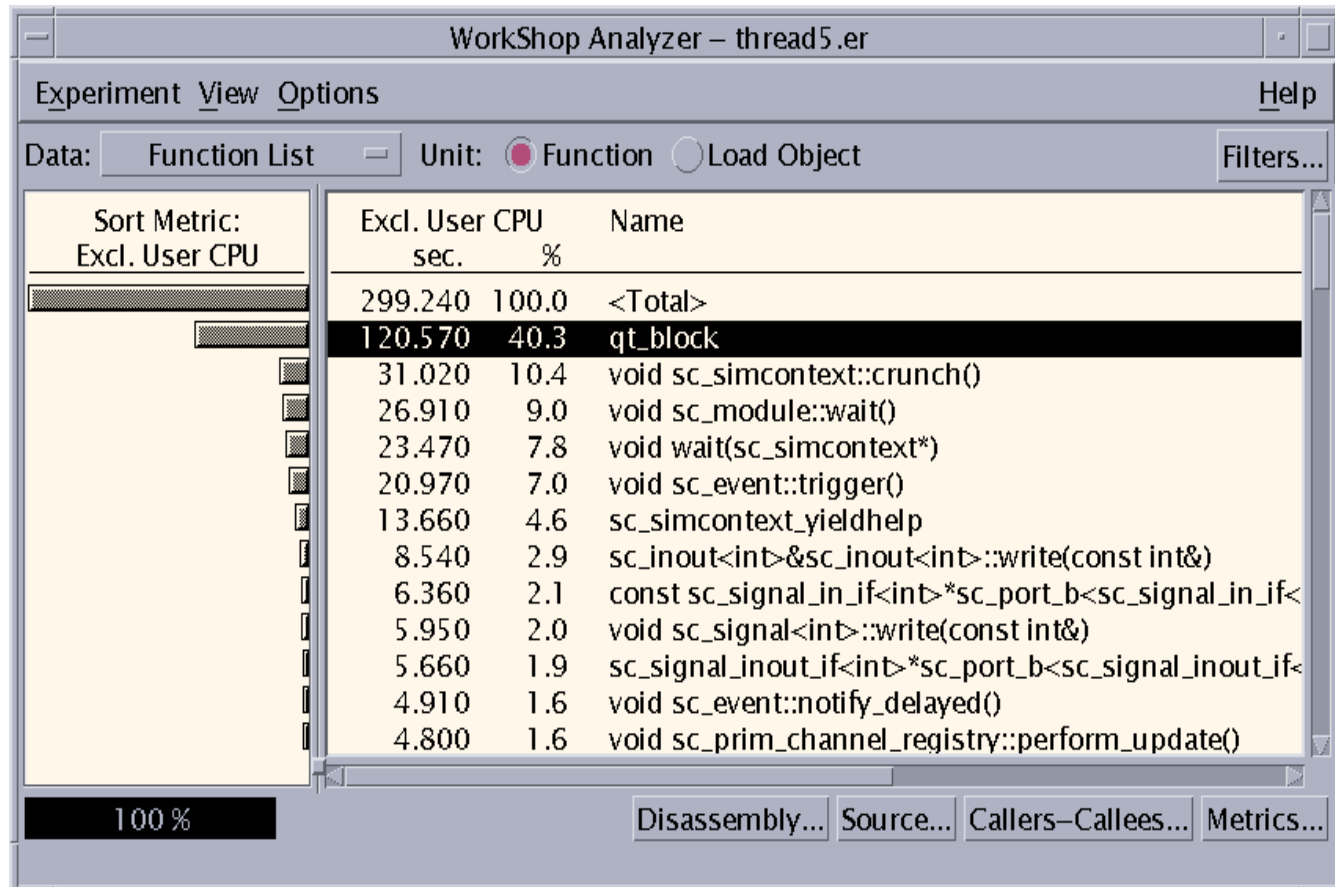
```
retl /* Return to address in %o7. */  
add %sp, 72, %sp /* Deallocate kwsa, ret pc area. */  
.size _qt_block,(-_qt_block)
```

(...)



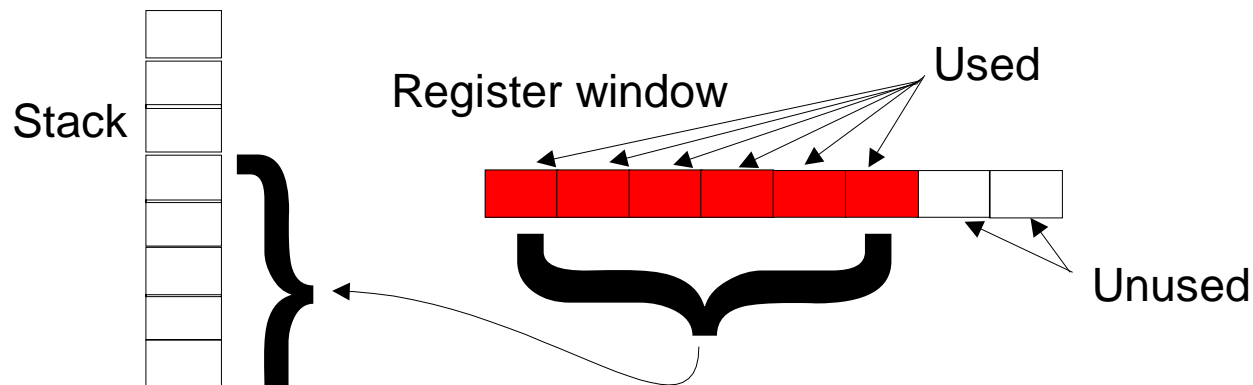
Performance Analysis

- Function `qt_block` from QuickThread lib shows in the analyzer window to be taking 40% of simulation time



Performance Analysis: Register Windows

- `ta 0x03` is an expensive trap, which goes into the kernel to save the register window of a `SC_THREAD`

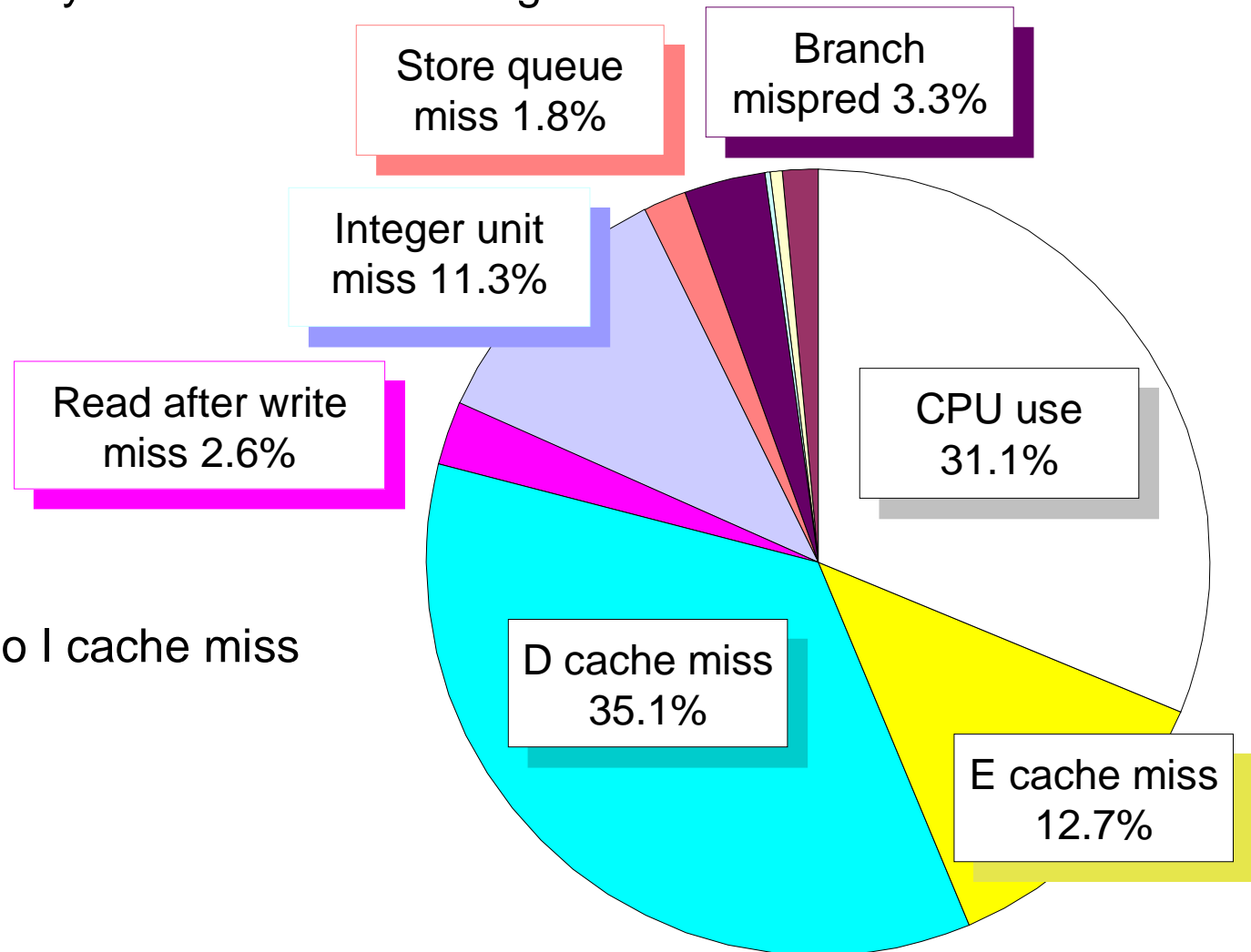


```
_qt_block:  
_qt_blocki:  
    sub %sp, 8, %sp /* Allocate save area for return pc. */  
    st %o7, [%sp+64] /* Save return pc. */  
_qt_abort:  
    ta 0x03 /* Save locals and ins. */  
  
    mov %sp, %o5 /* Remember old sp w/o chng ins/locals. */  
    sub %o3, 64, %sp /* Allocate kwsa, switch stacks. */  
    call %o0, 0 /* Call 'helper' routine. */  
    mov %o5, %o0 /* Pass old thread to qt_after_t() */  
    /* .. along w/ args in %o1 & %o2. */
```

(...)

Performance Analysis: Hardware Counters

- Where is the simulation spending its time ?
Analysis of simulation using various hardware counters



- No I cache miss

Performance Tuning: Data Prefetching

- Some Ecache misses can be avoided using prefetch instructions in qt_block
- 34% Ecache miss decrease.
- Overall simulation time decrease: 7%.

```
_qt_block:
_qt_blocki:
    prefetch [%o3], 0
    prefetch [%o3+64], 0
    prefetch [%o3+128], 0
    sub %sp, 8, %sp /* Allocate area
                        for return pc */
    st %o7, [%sp+64] /* Save return pc */
_qt_abort:
    ta 0x03 /* Save locals and ins */
    mov %sp, %o5 /* Remember old sp
                    w/o chng ins/locals*/
    sub %o3, 64, %sp /* Allocate kwsa,
                    switch stacks */
    call %o0, 0 /* Call 'helper' routine */
    mov %o5, %o0 /* Pass old thread to
                    qt_after_t() along w/ args in %o1 & %o2 */
    /* Restore callee-save regs. */
    ldd [%sp+ 0+64], %i0
    ldd [%sp+ 8+64], %i0
(...)
    ldd [%sp+48+64], %i4
    ldd [%sp+56+64], %i6
    ld [%sp+64+64], %o7 /* Restore return pc. */
    retl /* Return to address in %o7. */
    add %sp, 72, %sp /* Deallocate kwsa */
```



Performance Tuning: Bug Fix in Qthreads

- CAUSE:

The routine `qt_block`, which calls a helper routine, doesn't reserve the standard stack frame 96 bytes, but only 64 bytes. In particular, the area starting at `%sp+68` should be reserved for the callee to write the outgoing registers `%o0–%o5`.

- EFFECT:

The reserved area at `%sp+68+` collides with the "callee-save regs" that `qt_block` tries to restore after returning from helper function.

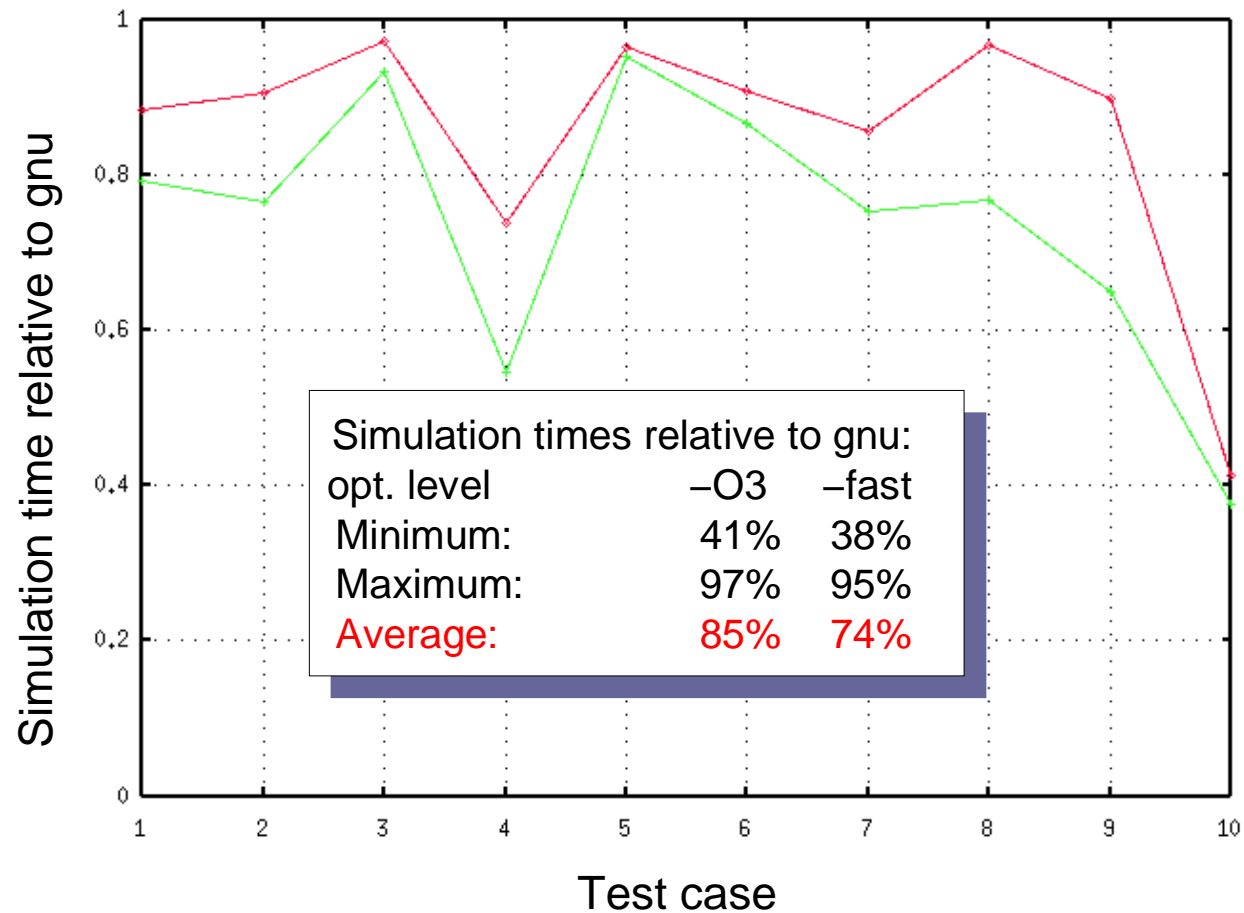
- RESULT:

With fixed code, the libraries can be compiled at `-fast` optimization level. Overall simulation time decrease: 9% for test case X.



Performance Tuning: Bug Fix in Qthreads

- Performance Improvement: **13%** in average over all test cases.



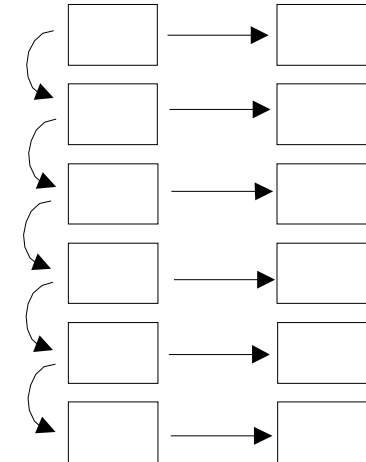
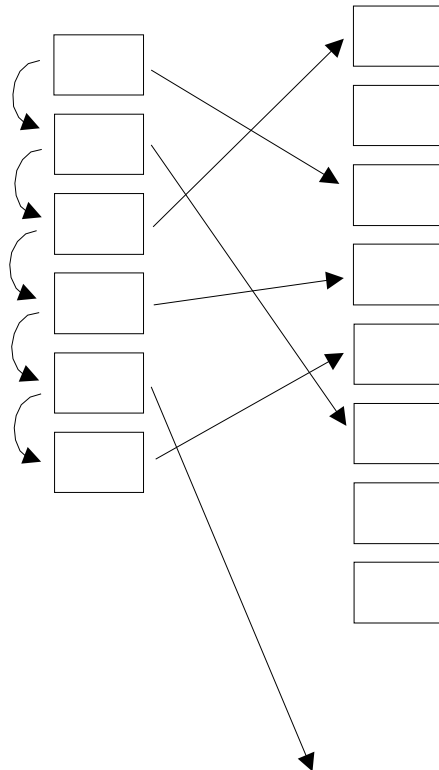
Performance Tuning: So far...

- Data Prefetching: 7%
- QT fix and higher optimization level: 13%
- Total improvement so far: 20%



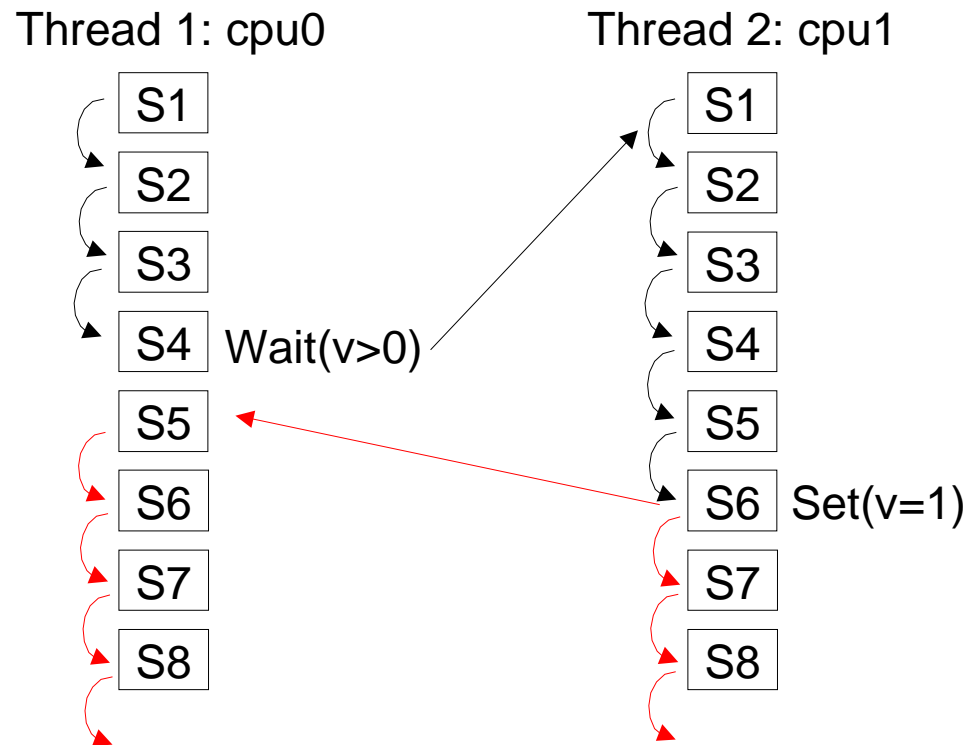
Future Work

- Decrease the level of indirection in the event list to improve memory locality.
 - ♦ Advantage: Improve performances by decreasing Ecache misses.
 - ♦ Drawback: Loss of generality of object oriented programming



Future Work

- Code could be parallelized on a multi-processor machine.
 - ♦ Threads stay on processors.
 - ♦ Use busy waits, no costly context switching.
 - ♦ Several threads could execute concurrently.



Conclusion

- SystemC performance Analysis:
 - ◆ Most of the time spent in thread context switching.
 - ◆ CPU stalls due to cache misses.
- Performance tuning:
 - ◆ Total improvements so far are 20% due to data prefetching and higher compile optimization level.
 - ◆ Forte Developer 6 update 2 compiler gives >20% faster code than gnu compiler.
- Further improvements:
 - ◆ Investigate possibility of increasing performance by improving memory locality, at the expense of generality.
 - ◆ Investigate multithreaded code for multi-processor machines.

