

Scheduler Activations

Effective Kernel Support for the User-Level Management of Parallelism

- Anderson, Thomas
- Bershad, Brian
- Lazowska, Edward
- Levy, Henry

Two approaches to creating threads

- User-level threads ("fibers")
- Kernel threads

Kernel threads (1:1)

- Lightweight processes sharing an address space
- Thread operations executed in kernel context
 - fork(), join(), signal(), wait()
 - Requires kernel boundary crossing
 - 10x slower than user threads
- (their measurement with Topaz)

User threads (N:1)

- Fast scheduling (no kernel calls)
- Fast thread operations
- Naive version can't use multiple CPUs
- I/O, page faults block the whole process

User-level threads ON kernel threads (M:N)

- Many user-level threads per kernel thread
- True concurrency with fast userlevel scheduling

- Without kernel-user interaction (scheduler activations):
 - Time-slicing interferes with assumptions
 - Kernel may preempt a thread while user code is holding a spinlock
 - Possibly leading to poor performance
 - User code may run out of kernel threads: some user threads may be starved

User-level management of parallelism

- "Scheduler activations" instead of kernel threads
- Allow use of multiple CPUs without kernel penalty
- Processors allocated by the kernel
- Processors may be added/removed at any time
- Kernel notifies userlevel on changes
- User threads on allocated processors
- User code can request additional (or fewer) processors
- Any user-level scheduling / concurrency model works

Scheduler activations

- Context for execution of a thread
- Several purposes:
 - Executing user code
 - Notifying user system of:
 - preemption
 - processor allocation
 - blocking
 - Stores processor context

Upcall points

Upcalls from kernel

- Add processor
- Preempted processor
- Blocked in kernel
- Unblocked in kernel

System calls from app, when threads <> activations

- Add more processors
- This processor is idle

Example: IO blocking

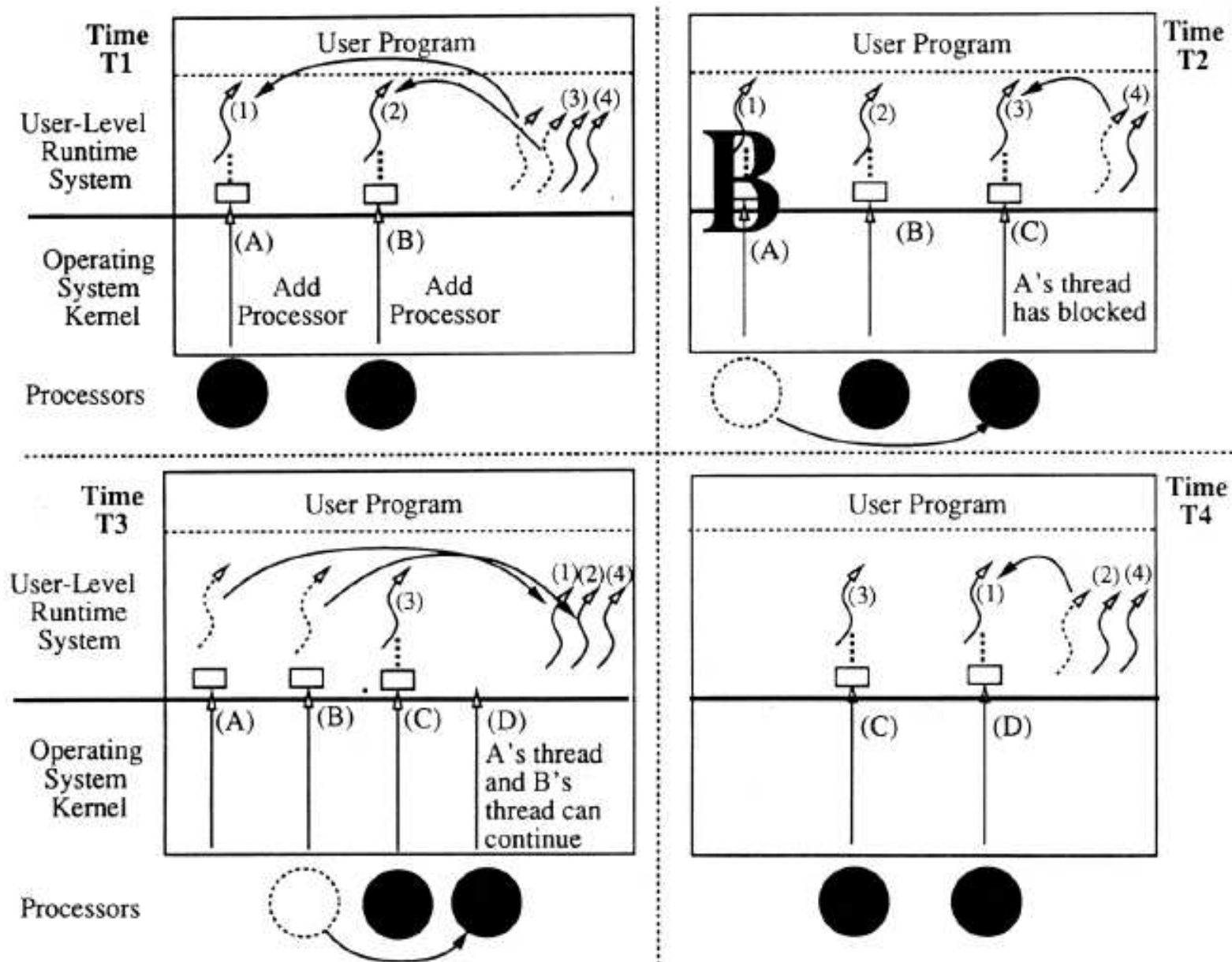


Fig. 1. Example: I/O request/completion.

Processor preemption

- 1. Scheduler activation interrupted, processor reassigned
- 2. Original process interrupted again, on another scheduler activation, with notification
- 3. User scheduler can decide which threads to run
- 4. Notification delayed until processor is available

Recovering from a preempted critical section

- Prevention would permit applications to escape preemption
- SA handler immediately runs any threads previously in critical sections
- Failing to do this can result in deadlock if the critical section locks the ready list

- For performance, this system creates a *copy* of the critical section at compile time
- Copy ends with code to yield to the code handling the interrupt
- A thread starts with the original code, is pre-empted, and resumes in the copy

Page faults

- As with I/O and other blocking system calls:
 - User threads: whole process blocks
 - Kernel threads: one thread blocks
- Scheduler activations: User code notified
 - *unless* the notification-handling code would page fault in the same place
 - which would need notification
 - which would page fault
 - which would need notification
 - which would page fault
 - etc.
 - creating an unbounded number of activations
- Kernel first checks whether the notification code would page fault and delays until the region is loaded

Performance

Table IV. Thread Operation Latencies ($\mu\text{sec.}$)

| Operation | FastThreads on Topaz Threads | FastThreads on Scheduler Activations | Topaz threads | Ulrix processes |
|-------------|---------------------------------|---|---------------|-----------------|
| Null Fork | 34 | 37 | 948 | 11300 |
| Signal-Wait | 37 | 42 | 441 | 1840 |

- Upcalls 5x slower than in Topaz in Signal-Wait
- Blamed on poor Modula-2 performance

Performance

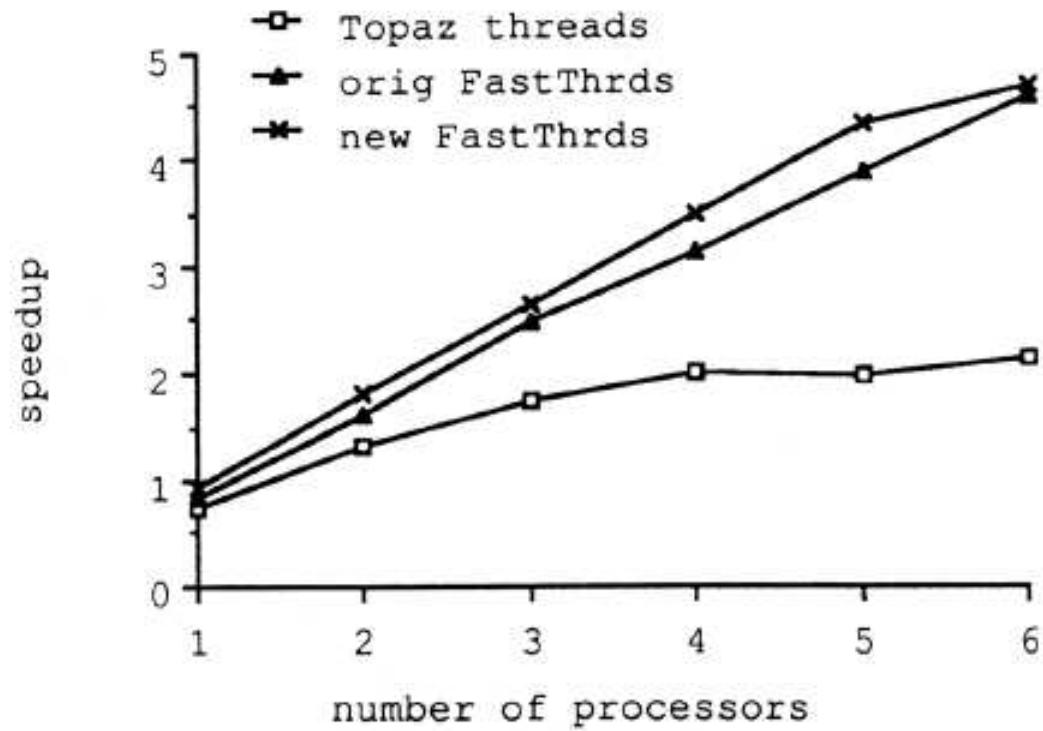


Fig. 2. Speedup of N-Body application versus number of processors, 100% of memory available.

Performance

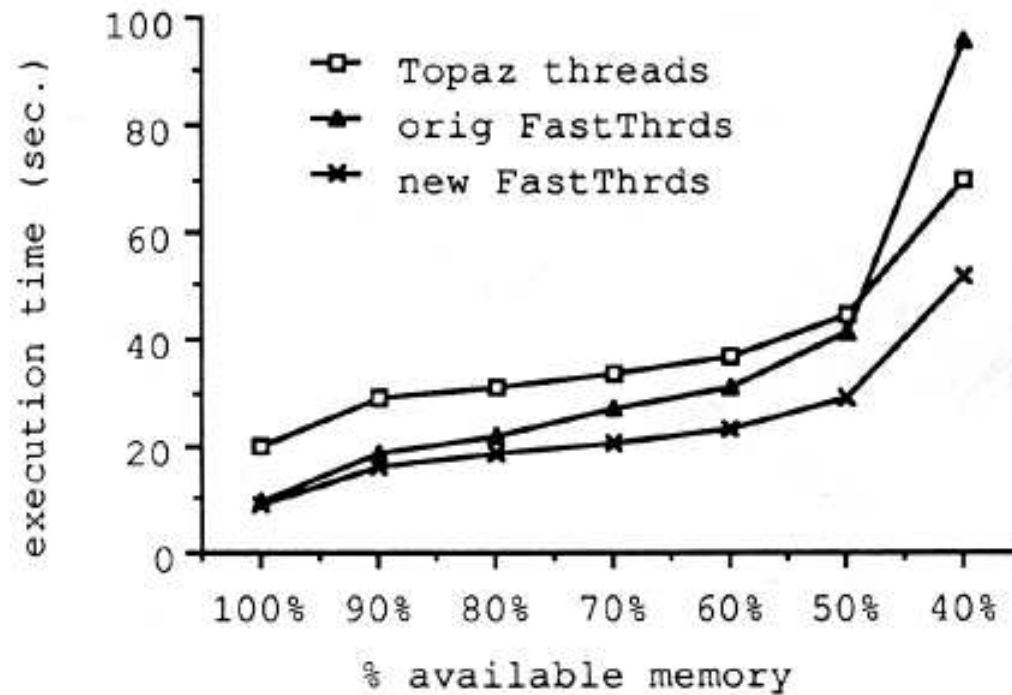


Fig. 3. Execution time of N-Body application versus amount of available memory, 6 processors.

Table V. Speedup of N-Body application, Multiprogramming Level = 2, 6 Processors, 100% of Memory Available

| Topaz Threads | Original FastThreads | New FastThreads |
|---------------|----------------------|-----------------|
| 1.29 | 1.26 | 2.45 |

Currently

- MxN model:
 - WinNT (fibers)
 - HP-UX
 - Tru64
 - NetBSD (scheduler activations)
- Solaris abandoned M:N model for 1:1 in v9
- Linux kernel developers hostile to SA
 - Ingo Molnar claims: Kernel threads (NPTL) are much faster and less complex