Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism

by Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, Henry M. Levy, ACM Transactions on Computer Systems, 10(1):53--79, February 1992.

Presented By
John Chee

For Jonathon Walpole's Concepts of Operating Systems (CS533) at Portland State University

The problem

- Kernel-level threads are the correct abstraction
  - But the implementation is expensive:
    - Entering and leaving the kernel on thread operations
- User-level threads are less expensive
  - But they are oblivious to kernel-level information:
    - Blocking a CPU on I/O or page faults

User-level threads

- Lightweight counterpart of kernel threads
- Implemented in “FastThreads”
  - “Require no kernel intervention”
  - Maps work from user-level threads to kernel-level threads with minimal information sharing
- FastThreads' thread operations cost one order of magnitude more than procedure calls
- Since they are implemented in a library they can be customized to the varying needs of developers
  - Without need to modify the kernel

User-level threads

- But kernel interfaces prevent efficient implementations.
- The virtual processor model can be insufficient in the presence of “'Real world' OS activity” such as:
  - Multiprogramming
  - I/O
  - Page faults
    - If you develop with kernel threads, the OS handles these situations intelligently. But they are too expensive.

The solution

- Provide the user-level with a “virtual multiprocessor”
- The kernel will communicate information to the application so that we can better handle I/O and page faults.
- The user-level thread system will provide a similar interface to a kernel-level interface and schedule threads intelligently.
  - The scheduler can now be modified without having to modify the kernel.

Scheduler activations (SA)

- The medium through which the kernel communicates to the user-level, more or less, a kernel thread.
  - Holds an execution context for the user-level like a kernel thread
    - Can be suspended just like a thread
  - Notifies the user-level of events

Communicating kernel information to the user-level

- When the kernel would need to make a scheduling decision, upcalls are used instead:
  - Add processor
  - Processor preempted
  - SA blocked
  - SA unblocked
- See Table II

Communicating user-level information to the kernel

- When the user-level system starts, enters a parallel section, or leaves a parallel section
  - Add *n* more processors
  - This processor is idle
- See Table III

Example

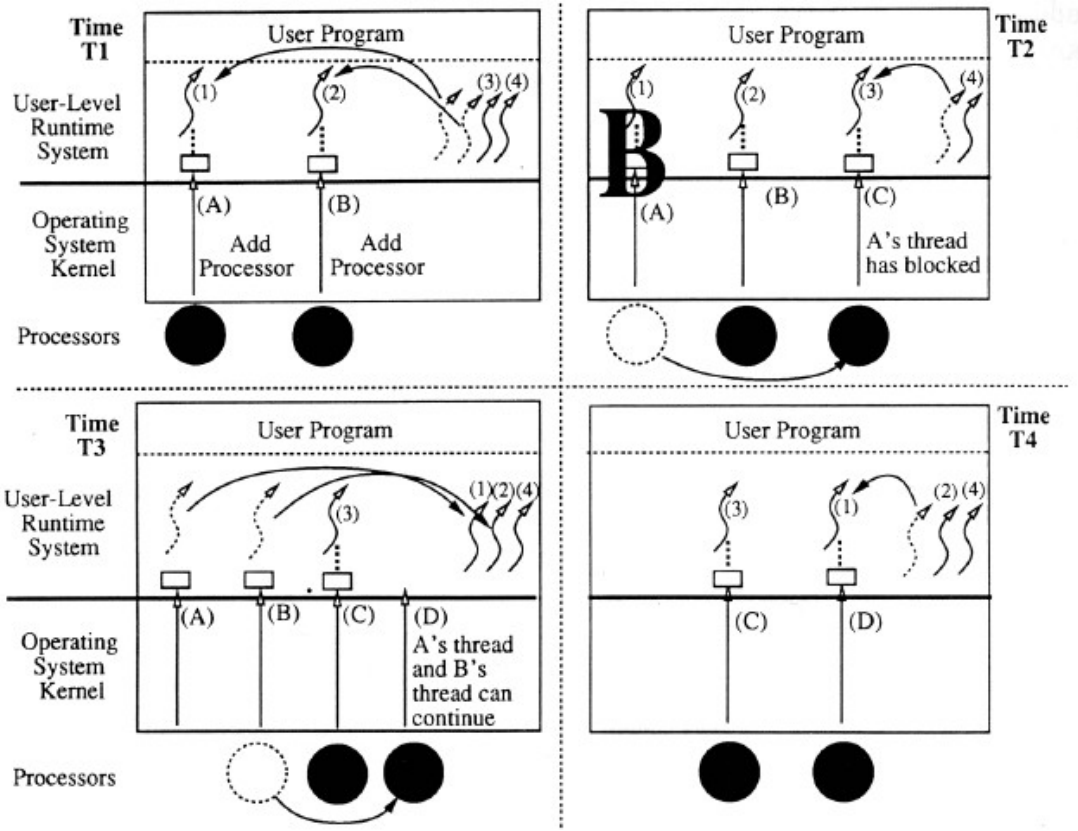

Fig. 1. Example: I/O request/completion.

- (1), (2) each get a processor.
- (1) blocks on I/O then user-level scheduler takes (3) off the ready and runs it.
- (1)'s I/O completes, (2)'s processor gets used to inform user-level and add (1) and (2) to the ready list.
- (1) is taken of the ready list and run.

A note on critical sections

- If a thread is executing in a critical section and it is preempted it continues executing until it's done executing the critical section.
  - This is implemented with minimal overhead by copying critical section code and bracketing it with yields back to preemptor.
- Deadlock is avoided, but long critical sections are still a problem.

Implementation

- Where the kernel performed thread operations, upcalls (kernel to user-level) were used instead.
  - Added processor affinity and maintained object code compatibility
    - 1200 lines of code
- The user-level had to process upcalls and to provide the kernel with information about it's processor needs.
    - Few hundred lines of code

Implementation notes

- A new processor allocation policy was used which evenly allocated processors to all the highest priority applications.
- Scheduler activations are cached and batched rather than being eagerly destroyed or processed one by one.

Performance

- Preserved order of magnitude decrease from processes to kernel threads to user-level threads.
- Perilously parallel application: N-Body Barnes-Hut algorithm
  - Recursively dividing the problem creating trees, calculating the centroids, approximating the force.

Performance

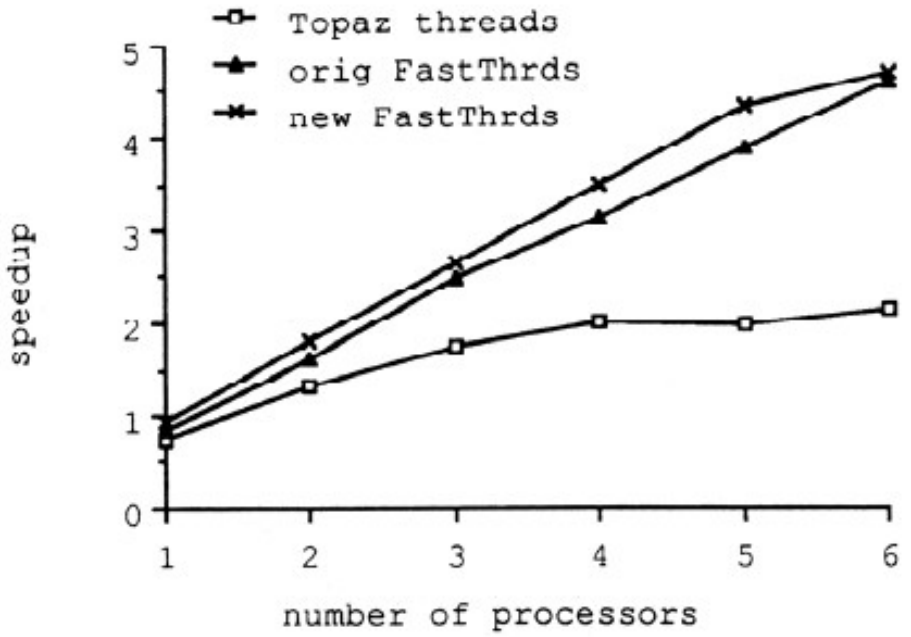

Fig. 2. Speedup of N-Body application versus number of processors, 100% of memory available.

- Almost linear speedup
- No I/O
- As good as original FastThreads

Performance

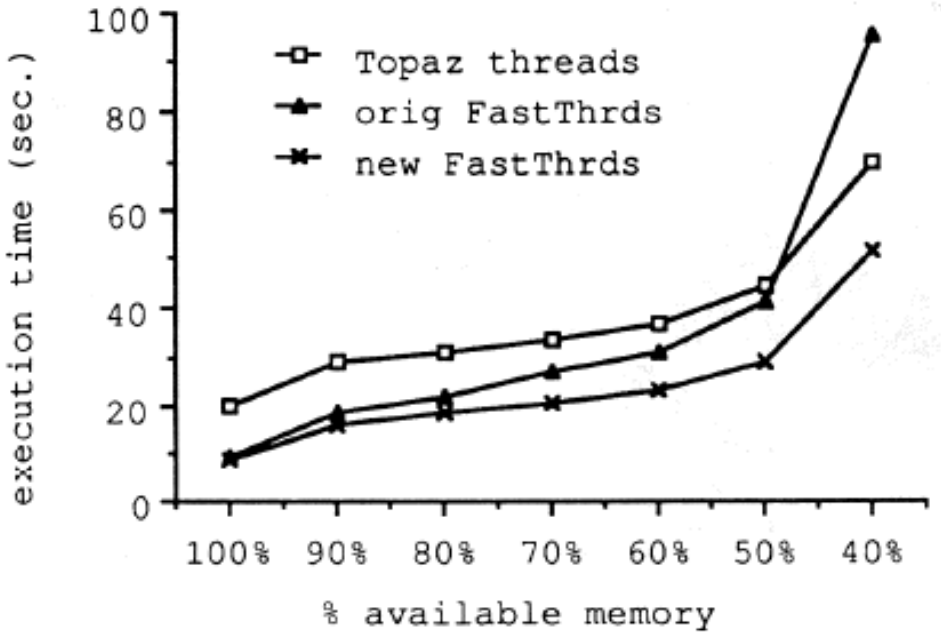

Fig. 3. Execution time of N-Body application versus amount of available memory, 6 processors.

- With I/O
- As good as kernel threads

Performance

- Speedup in the presence of multiprogramming with 2 applications running and 6 processors:
  - Kernel threads: 1.29
  - Original user-level threads: 1.26
  - User-level threads with scheduler activations: 2.45

Summary

- Scheduler activations allow us to achieve the combined wins of lightweight user-level threads and the intelligent handling of I/O and multiprogramming of kernel-level threads.
- With 2,4,8, and 16 core systems, parallel applications will be a standard development practice which cannot be bogged down by the inefficiencies of kernel-level threads or insufficient kernel interfaces.

Thanks

- Graphics copied from previous class presentation by Shiyan Tao.