Project LAOS
Latency Awareness in Operating Systems

Gabor Drescher, Sebastian Maier, Jacob Denker,
Wolfgang Schröder-Preikschat

Friedrich-Alexander-Universität Erlangen-Nürnberg

2013-11-07

Operating Systems for Future Hardware Architectures

- **Future Hardware Architectures:**
  - **Hundreds** or **thousands** of cores
  - Sharing memory (NUMA)
  - Massive parallel systems

- **Operating Systems:**
  - Focus on **scalability** + **latency**
  - Maybe Linux?
  - How to do it better?

→ **LAOS**

```
raw_spin_lock_irq(&curr->pi_lock);
while (!list_empty(head)) {
  [...]

  raw_spin_unlock_irq(&curr->pi_lock);
  spin_lock(&hb->lock);
  raw_spin_lock_irq(&curr->pi_lock);

  [...]

  list_del_init(&pi_state->list);
  pi_state->owner = NULL;

  raw_spin_unlock_irq(&curr->pi_lock);
  rt_mutex_unlock(&pi_state->pi_mutex);
  spin_unlock(&hb->lock);

  raw_spin_lock_irq(&curr->pi_lock);
}
raw_spin_unlock_irq(&curr->pi_lock);
```

Kernel 3.12, futex.c

Project LAOS

Latency prevention, reduction, hiding

- Domain specific design
- Wait-free and non-blocking synchronization
  - Synchronization and thread management latencies
- Detect and resolve contention
- Asynchronous system calls
- Dedicated cores for OS internals

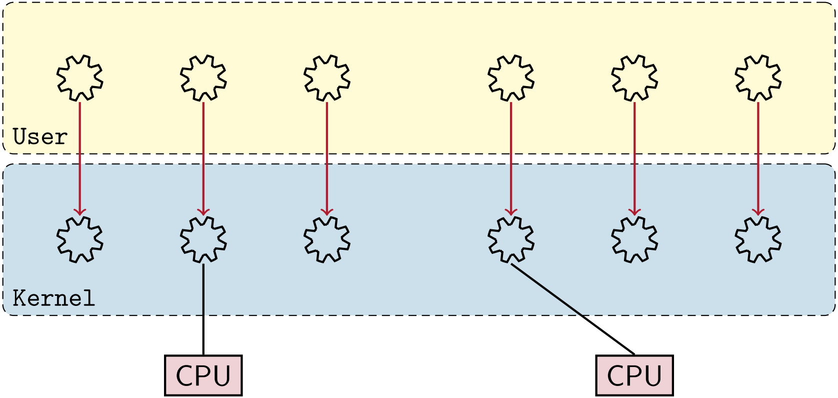
- Experiments on OS architectures
  - Adopt concepts to Linux?

LAOS Kernel Features

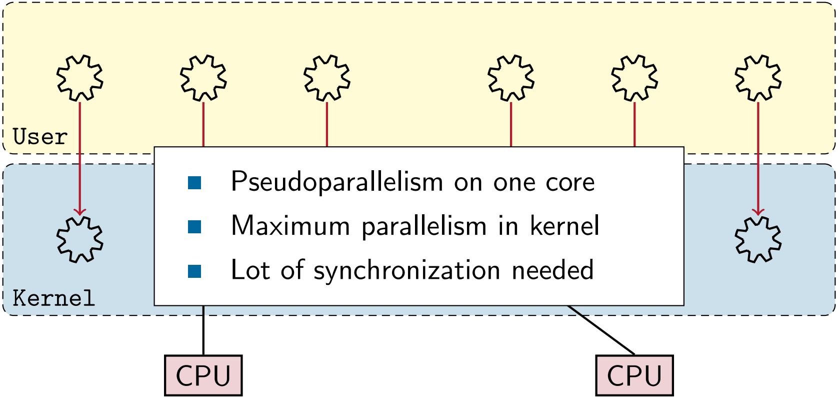
- Thread scheduling and synchronization
  - Dynamic creation of threads + sync. primitives
  - Pthread API
  - Mutex, cond. variable, semaphore, barrier
- Deferred interrupt handling
  - First and second level interrupts
- Completely **non-blocking** implementation
  - No spinlocks
  - No IRQ blocking
- On standard **x86-64** multicore hardware

First of its kind

LAOS Kernel Architecture

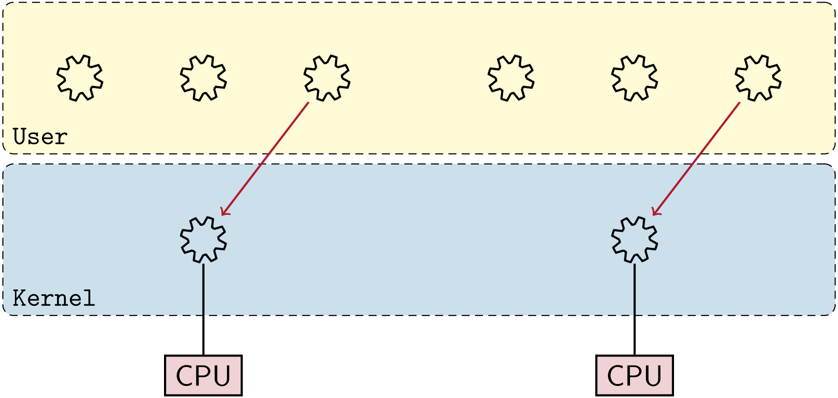


LAOS Kernel Architecture

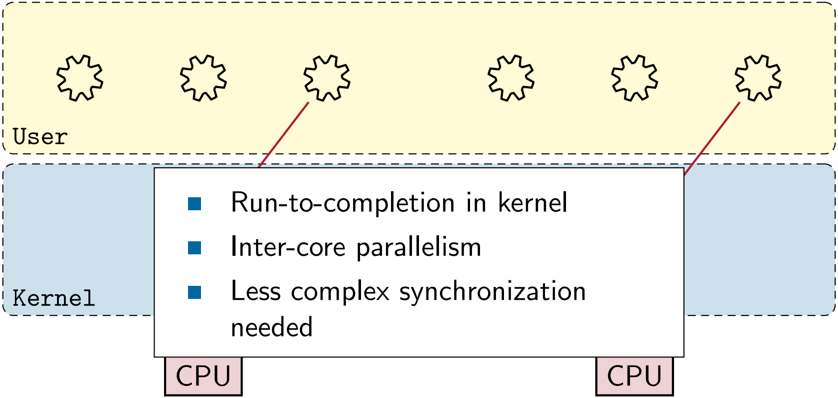


- Pseudoparallelism on one core
- Maximum parallelism in kernel
- Lot of synchronization needed

LAOS Kernel Architecture

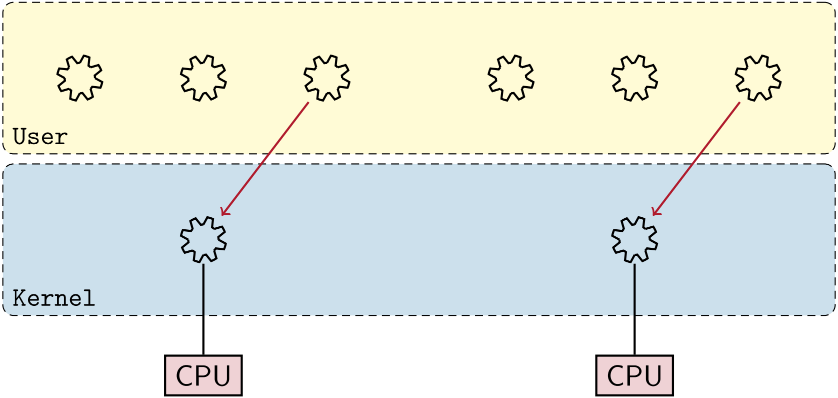


LAOS Kernel Architecture



- Run-to-completion in kernel
- Inter-core parallelism
- Less complex synchronization needed

LAOS Kernel Architecture



- **Event-based multi-core architecture**
  - TinyOS http://www.tinyos.net/

Non-Blocking Synchronization

- Lock freedom, no critical sections
- Atomic CPU operations
  - read, write, CAS, DCAS, FAA, SWAP, . . .

Advantages:

- **Guaranteed system-wide progress**
- **Deadlocks and priority inversion impossible**
- **Composable without overhead or conventions**

Disadvantages:

- Harder to write non-blocking algorithms

Semaphore

```
Semaphore {
    uint counter;
    MSQueue queue;

    void add(thread_t *t);
    thread_t* remove();
}

signal(Semaphore *sem) {
        FAA(&(sem->counter), 1);
        wakeup_thread(sem);
        leave_kernel();
}
```

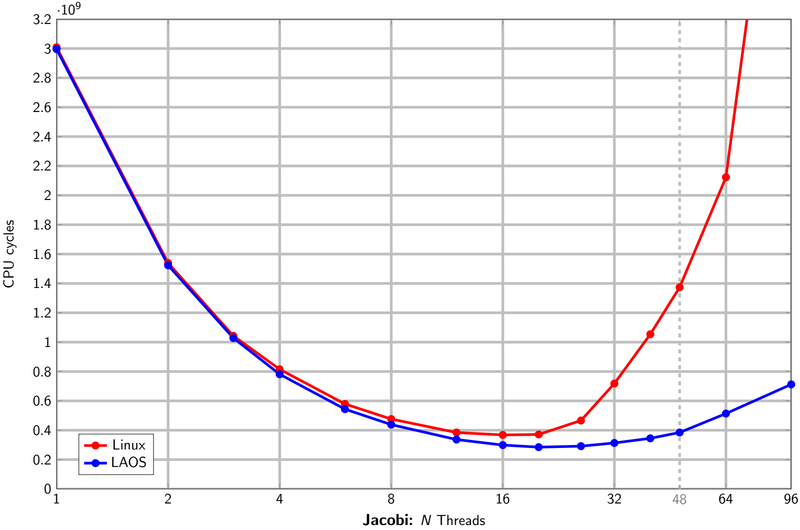
Semaphore

```
wait(thread_t *user, Semaphore *sem) {
  while (true) {

    cnt = sem->counter;
    if (cnt == 0) {
      repeat_syscall(user, wait);

      sem->add(user);
      if (sem->counter != 0) {
        wakeup_thread(sem);
      }
      leave_kernel();
    }

    if (CAS(&(sem->counter), cnt, cnt-1)) {
      leave_kernel();
    }
  }
}
```

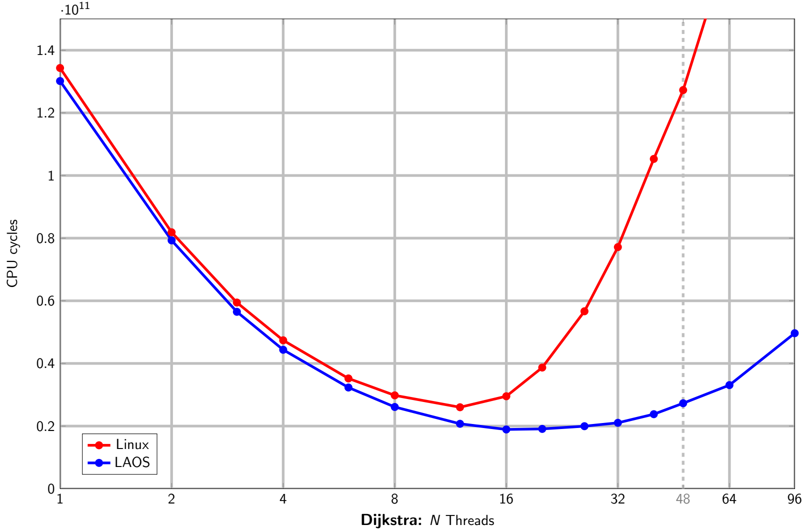
Measurements

- **Task:** show good scaling, even with high contention
- Software:
  - GCC 4.7.2, Glibc 2.17, Linux 3.11
- Hardware:
  - 48-core AMD 6180SE @ 2.6 Ghz
  - 8 ccNUMA nodes, 8 GB RAM each
- Application benchmarks:
  - **Jacobi:**
    - Iterative solution to a boundary value problem, matrix of data, geometric decompostion
  - **Dijkstra:**
    - Parallel search for shortest path

Measurements



Measurements



Future Work

- Comparable benchmarks
  - NASA Advanced Supercomputing Division (NAS)
    - Parallel numerical benchmarks
    - Various synchronization and communication patterns
    - OpenMP + libc constructs and functions
- NUMA awareness
  - Distribution of static global memory
    - Assign at runtime
    - ⇝ Paging subsystem
  - Dynamic memory allocation
    - Physical locality to specific node

Conclusion

- LAOS explores OS architectures
  - For current + future many-core systems
  - Apply insights to existing systems
- Latency prevention, reduction and hiding
- Event-based multi-core architecture
- Completely non-blocking kernel on standard hardware
  - Better scaling in high contention scenarios
  - No internal deadlocks by design, guaranteed progress
  - Pthread API
  - Easier systems programming