Unlocking Energy

Babak Falsafi, Rachid Guerraoui, Javier Picorel, Vasileios Trigonakis
Conference Dilemma: To Sleep or Not to Sleep?

For the next 25 minutes

Do not sleep for such short duration

sleep

busy wait

Unlocking Energy
Motivation
Energy Efficiency Through Synchronization (Locking)

Why lock-based synchronization?
1. Concurrent systems synchronize with locks
2. Locks are well-defined abstractions → lock() / unlock()
3. Locking strategies affect power consumption

Busy lock

lock() | wait!
(waste time and energy)
Lock Waiting Techniques

busy lock

sleeping (blocking)

busy waiting (spinning)

Q. Is sleeping energy-friendly?

Locking is a good candidate for reducing energy consumption.
Energy Efficiency By Improving Locking

1. Concurrent systems synchronize with locks
2. Locks are well-defined abstractions
3. Locking strategies affect power consumption

4. **Energy efficiency** and **throughput** go **hand in hand** in the context of lock algorithms
Outline

• Motivation
  
  busy lock
  
  busy waiting $\rightarrow$ hurts power consumption

  spin-then-sleep “cleverly”

  sleeping $\rightarrow$ saves power, but hurts throughput

• Improving the energy efficiency of systems

Target platform
2-socket Intel Ivy Bridge
20 cores, 40 hyper-threads
Power Consumption of Waiting

Observations

1. Sleeping power-friendly
2. Busy waiting
   ```
   while (*lock != FREE) {} 
   ```

Busy waiting is very power hungry
Reducing Power of Busy Waiting

while (*lock != FREE) { ?? }

Observations

1. empty: 1 iteration / cycle
2. Intel docs: use pause
3. pause not ideal
4. mfence > pause
5. Still, mfence ~5% better
6. DVFS and mwait are **not practical**

Power consumption of busy waiting cannot be practically reduced
• Motivation

busy waiting $\rightarrow$ hurts power consumption

spin-then-sleep “cleverly”

sleeping $\rightarrow$ saves power, but hurts throughput

• Improving the energy efficiency of systems
Sleeping Might Be Necessary (For Two Reasons)

1. Power Consumption—Waiting
   - Sleeping can reduce power consumption (and more)

<table>
<thead>
<tr>
<th># Threads</th>
<th>Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>30</td>
<td>150</td>
</tr>
<tr>
<td>40</td>
<td>200</td>
</tr>
</tbody>
</table>

2. Locks with Multiprogramming
   - MySQL In-Memory
     - # threads > hw contexts

<table>
<thead>
<tr>
<th>Normalized Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleeping</td>
</tr>
<tr>
<td>1.0</td>
</tr>
</tbody>
</table>
Latency: The Price of Sleeping

Observations

1. Sleep call: release context
2. Wake-up call: to handover the lock
3. Turnaround latency ≈ lock handover latency

Frequent sleep/wake-up calls reduce throughput without saving energy
Outline

• Motivation

busy waiting $\rightarrow$ hurts power consumption

spin-then-sleep “cleverly”

sleeping $\rightarrow$ saves power, but hurts throughput

• Improving the energy efficiency of systems
Reducing Fairness: Sleeping for Long Durations

**Power Consumption**

- sleep
- spin
- unfair

**Communication Throughput**

- sleep
- spin
- unfair

### Unfair: 1000:1 spin-to-sleep ratio (while 2 threads spin, the rest sleep)

Trade fairness for energy efficiency
How can we use these results in designing locks?
# Problems of Pthread MUTEX lock

## lock()

| MUTEX | 1. Spins less than sleep latencies  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>For up to 100 attempts</td>
<td>spin with pause</td>
</tr>
<tr>
<td>if still busy, sleep</td>
<td>2. Spins with pause</td>
</tr>
</tbody>
</table>

## unlock()

<table>
<thead>
<tr>
<th>MUTEX</th>
<th>3. Always wakes up a thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>release in user space</td>
<td></td>
</tr>
<tr>
<td>wake up a thread</td>
<td></td>
</tr>
</tbody>
</table>

Pthread MUTEX does not take into account the sleep overheads.
### MUTEXEE: An Optimized MUTEX Lock

#### lock()

<table>
<thead>
<tr>
<th>MUTEX</th>
<th>MUTEXEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>For up to 100 attempts</td>
<td>For up to ~8000 cycles</td>
</tr>
<tr>
<td>spin with pause</td>
<td>spin with mfence</td>
</tr>
<tr>
<td>if still busy, sleep</td>
<td></td>
</tr>
</tbody>
</table>

#### unlock()

<table>
<thead>
<tr>
<th>MUTEX</th>
<th>MUTEXEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>release in user space (lock-&gt;locked = 0)</td>
<td>wait in user space (~300 cycles)</td>
</tr>
<tr>
<td>wake up a thread</td>
<td></td>
</tr>
</tbody>
</table>
Performance of MUTEXEE over MUTEX

Throughput

Energy Efficiency

One lock

MUTEXEE better

Critical Section (cycles)

Critical Section (cycles)

Fairness: results and analysis in the paper

MUTEXEE fixes the problematic cases of MUTEX
• Motivation

busy waiting $\rightarrow$ hurts power consumption

spin-then-sleep “cleverly”

sleeping $\rightarrow$ saves power, but hurts throughput

• Improving the energy efficiency of systems
Evaluation: Improving Energy Efficiency of Systems Through Locks

Six modern software systems – Overload pthread mutex

Average Throughput and Energy Efficiency

<table>
<thead>
<tr>
<th>Throughput</th>
<th>Energy Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICKET</td>
<td>1.26</td>
</tr>
<tr>
<td>MUTEXEE</td>
<td>1.28</td>
</tr>
</tbody>
</table>

Results
1. Benefits: Avoid sleeping
2. Sleeping is sometimes necessary
3. Throughput-driven benefits
4. MUTEXEE >> MUTEX

Locking can indeed be used to improve the energy efficiency of systems
Concluding Remarks

• An analysis of the energy efficiency of lock-based synchronization
  – Energy efficiency of locks goes hand in hand with throughput
  – MUTEXEE: an optimized MUTEX lock

→ Locking can be used to improve the energy efficiency of systems

• LOCKIN: https://github.com/LPD-EPFL/lockin

THANK YOU! QUESTIONS?