Simple and Fast Biased Locks

Nalini Vasudevan
Columbia University

Kedar Namjoshi       Stephen Edwards
Bell Laboratories    Columbia University
Motivation

- Packet Analyzer

- Simple and Fast Biased Locks, Nalini Vasudevan, Columbia University – p. 2
Locking

flag[i] = 1;
while (flag[j] == 1) {}
/* critical section */
flag[i] = 0;
Locking

Problematic

```c
flag[i] = 1;
while (flag[j] == 1) {}
/* critical section */
flag[i] = 0;
```
Peterson’s algorithm

```c
flag[i] = 1;
turn = j;
while (flag[j] == 1 && turn == j) {} /* critical section */
flag[i] = 0;
```
Peterson’s algorithm

```c
flag[i] = 1;
turn = j;
while (flag[j] == 1 && turn == j) {}
/* critical section */
flag[i] = 0;
```

Still problematic
Peterson’s algorithm

```c
flag[i] = 1;
turn = j;
fence(); /* force other threads to see flag and turn */
while (flag[j] && turn == j) {} /* spin */
/* critical section */
fence(); /* make visible changes made in critical section */
flag[i] = 0;
```
Peterson’s algorithm

flag[i] = 1;
turn = j;
fence(); /* force other threads to see flag and turn */
while (flag[j] && turn == j) {} /* spin */
/* critical section */
fence(); /* make visible changes made in critical section */
flag[i] = 0;

Applies to two processes only
N-process locks

```c
bool success;
do {
    while (lck == 1) {} /* wait */
    success = compare_and_swap(&lck, 0, 1);
} while (!success);

/* critical section */
lck = 0;
```
Motivation

- Packet Analyzer

Shared Memory

<table>
<thead>
<tr>
<th>H1</th>
<th>H2</th>
<th>H3</th>
<th>H4</th>
</tr>
</thead>
</table>

Core 1  
Core 2  
Core 3  
Core 4
Peterson’s algorithm

```c
flag[i] = 1;
turn = j;
fence(); /* force other threads to see flag and turn */
while (flag[j] && turn == j) {} /* spin */
/* critical section */
fence(); /* make visible changes made in critical section */
flag[i] = 0;
```

• Two process algorithm
Dominant process lock

- Contends with other processes using Peterson’s algorithm

```c
peterson_lock();
/* critical section */
peterson_unlock();
```
Non-Dominant process lock

• Contends with the dominant processes using Peterson’s algorithm
• Contends with other non-dominant processes using a normal n-process lock.

```
lockN();
peterson_lock();
/* critical section */
peterson_unlock();
unlockN();
```
Biased Lock = 2-lock + n-lock

if (this_thread_id == owner)
    lock2();
else {
    lockN();
    lock2();
}

if (this_thread_id == owner)
    unlock2();
else {
    unlock2();
    unlockN();
}
The problem

```c
flag[i] = 1;
turn = j;
fence(); /* force other threads to see flag and turn */
while (flag[j] && turn == j) {} /* spin */
/* critical section */
fence(); /* make visible changes made in critical section */
flag[i] = 0;
```

• Need fences
Asymmetric locks

- Eliminate fences in the dominant process

Dominant process

```
while (grant) {} /* wait */
/* critical section */
if (request) {
    request = 0;
    fence();
    grant = 1;
}
```

Non-dominant process

```
lockN();
request = 1;
while (grant == 0) {} /* wait */
/* critical section */
fence();
grant = 0;
unlockN();
```
Performance

Asymmetric
Peterson + Pthread
Unbiased Pthread

Increase in speed (%)

Domination Percentage

Simple and Fast Biased Locks, Nalini Vasudevan, Columbia University – p. 14
Performance - Higher domination

![Graph showing performance metrics for different domination percentages and lock types. The graph compares Asymmetric, Peterson + Pthread, and Unbiased Pthread methods. The x-axis represents domination percentage, ranging from 90 to 100, while the y-axis represents increase in speed (%). The graph shows a significant increase in speed for the Unbiased Pthread method at higher domination percentages.]
Performance - Packet analyzer

Asymmetric
Peterson + Pthread
Unbiased Pthread

Increase in speed (%)

Percentage of local operations

Simple and Fast Biased Locks, Nalini Vasudevan, Columbia University – p. 16
Bias Transfer

- Dynamic scheme for transferring bias
- Based on the frequency
- Only one thread can be declared dominant at any time
Performance - Bias Transfer

Dominance set statically and correctly
With on-the-fly ownership transfer
Unbiased Pthread

Increase in speed (%) vs Domination Percentage
Conclusions

- Simple algorithms for constructing biased locks
- Verified using the SPIN model checker
- Implemented as a library
- Extended it to read-write locks
- Good performance when high dominance
- Future work: different architectures