Introduction to Parallel Computing

Introduction to non-blocking algorithms

Alexandre David

Concurrent non-blocking algorithms

- Concurrent: Several threads can execute the algorithms simultaneously.
- Blocking algorithms: Algorithms for which processes may isolate or block part of the data-structure to access it without interference. May cause deadlocks.
- Non-blocking algorithms: They ensure that the data-structure is always accessible to all processes. Independent from other halted/delayed processes.

Compare and swap (CAS)

- Atomic instruction available on most processors.
- Most common building block for non-blocking algorithms.
- Available in Java
  AtomicInteger.compareAndSet(int, int) -> bool

- If the memory is equal to some expected value (compare) then set the memory to a new value.

- Intel:
  cmpxchg r/m, r (needs lock prefix)
  if eax == r then r/m = r, ZF=0
  else eax = r/m, ZF=1
Other atomic instructions (Intel at least)

- Increment.  
  \texttt{(lock inc r/m)}
- Decrement.  
  \texttt{(lock dec r/m)}
- Exchange.  
  \texttt{(xchg r/m, r)}
- Fetch and add.  
  \texttt{(lock xadd r/m, r)}

- They can be used to implement simple and efficient synchronizations primitives.

Non-blocking algorithm

- The key:
  - Try to compute speculatively.
  - CAS before committing the result.
  - Retry if CAS fails.

- Good practice:
  - Work with a state-machine.
  - Every state must be consistent.
  - States = committed (intermediate) results.

Non-blocking counter

<table>
<thead>
<tr>
<th>Standard blocking algorithm</th>
<th>Non-blocking algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{proc inc(A)}</td>
<td>\texttt{proc inc(A)}</td>
</tr>
<tr>
<td>\texttt{lock}</td>
<td>\texttt{lock}</td>
</tr>
<tr>
<td>\texttt{tmp = A}</td>
<td>\texttt{tmp = A}</td>
</tr>
<tr>
<td>\texttt{tmp = tmp+1}</td>
<td>\texttt{while CAS(A, tmp, tmp+1)}</td>
</tr>
<tr>
<td>\texttt{A = tmp}</td>
<td>\texttt{end}</td>
</tr>
<tr>
<td>\texttt{unlock}</td>
<td>\texttt{end}</td>
</tr>
<tr>
<td>\texttt{end}</td>
<td></td>
</tr>
</tbody>
</table>
Non-blocking stack [Treiber's algorithm]

```plaintext
proc push(new)
do
  old = top
  new.next = old
  while not CAS(top, old, new)
end

proc pop
do
  old = top
  return null if old == null
  new = old.next
  while not CAS(top, old, new)
    return old
end
```

While trying to push a new element, if the CAS operation fails, the element is not added to the stack. If it's a pop operation and the top of the stack is null, a null value is returned.

The ABA problem

- Suppose that the value of V is A.
- Try a CAS to change A to X.
- Another thread can change A to B and back to A.
- The CAS won't see it and will succeed.
- Usual solution: Add a version number to V.

```
V: A
V=B;...V=A;
exec CAS(V, A, X)
V:X
```
The ABA problem

- Some algorithms may suffer from it.
- Example: Linked list.

Expected behavior:

```
head → pop → head
```

Fixes:

- Reference counter (implicit in Java).
- Allocation/de-allocation problems.
- Version number.
- ABA problems.
Insertion in a queue [Michael-Scott’s algorithm]

```
proc put(new)
  do
    last = tail
    nxt = last.next
    if last == tail
      if nxt == null
        if CAS(last.next, null, new)
          CAS(tail, last, new)
          break
        fi
      else
        CAS(tail, last, nxt)
      fi
    fi
  fi
loop
end
```