AutoSynch
An Automatic-Signal Monitor
Based on Predicate Tagging

Wei-Lun Hung
wlhung@utexas.edu

Vijay K. Garg
garg@ece.utexas.edu

Parallel and Distributed Systems Laboratory
Department of Electrical & Computer Engineering
Outline

1. Introduction

2. Our Approach
   - Evaluate Predicate: Closure
   - Avoid signalAll Calls: Relay Signaling Rule
   - Reduce Predicate Evaluations: Predicate Tagging

3. Results

4. Conclusions and Future Work
public class BoundedBuffer {
    Object[] buff;
    int putPtr, takePtr, count;
    // for mutual exclusion and synchronization
    Lock mutex = new ReentrantLock();
    Condition full = mutex.newCondition();
    Condition empty = mutex.newCondition();
    public BoundedBuffer(int n) {
        buff = new Object[n];
        putPtr = takePtr = count = 0;
    }
}

count = 4
putPtr

takePtr
public Object take() {
    // lock before operations
    mutex.lock();
    // wait if the buffer is empty
    while (count == 0) {
        empty.await();
    }
    Object ret = buff[takePtr++];
takePtr %= buff.length;
count--;
    // signal other threads when the buffer
    // is no longer full
    if (count == buff.length - 1) {
        full.signalAll();
    }
    // unlock after operations
    mutex.unlock();
}
public Object take() {
    // lock before operations
    mutex.lock();
    // wait if the buffer is empty
    while (count == 0) {
        empty.await();
    }
    Object ret = buff[takePtr++];
    takePtr %= buff.length;
    count--;
    // signal other threads when the buffer
    // is no longer full
    if (count == buff.length - 1) {
        full.signalAll();
    }
    // unlock after operations
    mutex.unlock();
}
public Object take() {
    mutex.lock();
    if (count == 0) {
        empty.await();
    }
    Object ret = buff[takePtr++];
    takePtr %= buff.length;
    count--;
    if (count == buff.length - 1) {
        full.signal();
    }
    mutex.unlock();
}
public Object take() {
    mutex.lock();
    if while (count == 0) {
        empty.await();
    }
    Object ret = buff[takePtr++];
takePtr %= buff.length;
count--;
    if (count == buff.length - 1) {
        full.signal();
    }
    mutex.unlock();
}
public Object take() {
    mutex.lock();
    while (count == 0) {
        empty.await();
    }
    Object ret = buff[takePtr++];
    takePtr %= buff.length;
    count--;
    if (count == buff.length - 1) {
        full.signal(); full.signalAll();
    }
    mutex.unlock();
}
public AutoSynch class BoundedBuffer {
    Object[] buff;
    int putPtr, takePtr, count;
    public BoundedBuffer(int n) {
        buff = new Object[n];
        putPtr = takePtr = count = 0;
    }
}
public Object take() {
    waituntil (count > 0);
    Object ret = buff[takePtr++];
    takePtr %= buff.length;
    count--;
}
public AutoSynch class BoundedBuffer {
    Object[] buff;
    int putPtr, takePtr, count;
    public BoundedBuffer(int n) {
        buff = new Object[n];
        putPtr = takePtr = count = 0;
    }
    public Object take() {
        waituntil (count > 0);
        Object ret = buff[takePtr++];
        takePtr %= buff.length;
        count--;
    }
}

public class BoundedBuffer {
    Object[] buff;
    int putPtr, takePtr, count;
    Lock mutex = new ReentrantLock();
    Condition full = mutex.newCondition();
    Condition empty = mutex.newCondition();
    public BoundedBuffer(int n) {
        buff = new Object[n];
        putPtr = takePtr = count = 0;
    }
    public Object take() {
        mutex.lock();
        while (count == 0) {
            empty.await();
        }
        Object ret = buff[takePtr++];
        takePtr %= buff.length;
        count--;
        if (count == buff.length - 1) {
            full.signalAll();
        }
        mutex.unlock();
    }
}
The idea of automatic signaling was suggested by Hoare in (Hoa74), but rejected due to efficiency considerations.

The common belief: automatic signaling is extremely inefficient compared to explicit signaling (BFC95)(BH05)
Design Principles

Reduce number of context switches and predicate evaluations

Predicate evaluation
- Is essential in automatic signaling to decide which thread should be signaled

Context switches
- Avoid signalAll calls
  - Introduces redundant context switches
  - Is required in explicit signaling
Parameterized Bounded Buffer

A producer puts a bunch of items into the buffer
A consumer takes a number of items out of the buffer

- `buff.length = 64`
- `count = 24`
- `waituntil (count >= num_i)`
- `waituntil (items_i.length + count <= buff.length)`
- `num_1 = 32`
- `num_2 = 40`
- `condition waiting queue`

local variable
shared variable

PLDI 2013
public Object[] take(int num) {
    mutex.lock();
    while (count < num) {
        insufficientItem.await();
    }
    Object[] ret = new Object[num];
    for (int i = 0; i < num; i++) {
        ret[i] = buff[takePtr++];
        takePtr %= buff.length;
    }
    count -= num;
    insufficientSpace.signalAll();
    mutex.unlock();
    return ret;
}
public Object[] take(int num) {
    mutex.lock();
    while (count < num) {
        insufficientItem.await();
    }
    Object[] ret = new Object[num];
    for (int i = 0; i < num; i++) {
        ret[i] = buff[takePtr++];
        takePtr %= buff.length;
    }
    count -= num;
    insufficientSpace.signalAll();
    mutex.unlock();
    return ret;
}
public Object[] take(int num) {
    mutex.lock();
    while (count < num) {
        insufficientItem.await();
    }
    Object[] ret = new Object[num];
    for (int i = 0; i < num; i++) {
        ret[i] = buff[takePtr++];
        takePtr %= buff.length;
    }
    count -= num;
    insufficientSpace.signalAll();
    mutex.unlock();
    return ret;
}
AutoSynch Framework
Outline

1. Introduction

2. Our Approach
   - Evaluate Predicate: Closure
   - Avoid signalAll Calls: Relay Signaling Rule
   - Reduce Predicate Evaluations: Predicate Tagging

3. Results

4. Conclusions and Future Work
Closure

$\text{buff.length} = 64$
$\text{count} = 24$

$$\text{waituntil (count} \geq \text{num}_i)$$
$$\text{waituntil (items}_i\text{.length} + \text{count} \leq \text{buff.length})$$

$\text{num}_1 = 32$
$\text{num}_2 = 40$
Replace local variables with the values at runtime

\begin{align*}
\text{buff.length} & = 64 \\
\text{count} & = 24 \\
\text{items}_0\text{.length} & = 18 \\
\text{waituntil} \ (\text{count} \geq \text{num}_1) \\
\text{waituntil} \ (\text{count} \geq \text{num}_2)
\end{align*}
Closure

Replace local variables with the values at runtime

- Buff.length = 64
- Count = 24
- Waituntil (count >= 32)
- Items0.length = 18

- P0

- C1
  - Waituntil (count >= 32)

- C2
  - Waituntil (count >= 40)

Replace local variables with the values at runtime
The thread (P₀) owning a monitor performs predicate evaluations for other threads (C₁ and C₂) before leaving.

- buff.length = 64
- count = 42
- waituntil (count >= 32)

- Put 18 items
- waituntil (count >= 40)

- C₁
- C₂
Outline

1. Introduction

2. Our Approach
   - Evaluate Predicate: Closure
   - Avoid signalAll Calls: Relay Signaling Rule
   - Reduce Predicate Evaluations: Predicate Tagging

3. Results

4. Conclusions and Future Work
Relay Signaling Rule

\[
\begin{align*}
\text{buffer.length} &= 64 \\
\text{count} &= 42 \\
\text{waituntil (count } \geq 32) \\
\text{P0} \\
\text{waituntil (count } \geq 40)
\end{align*}
\]
Relay Signaling Rule

buff.length = 64
count = 42
waituntil (count >= 32)

C1
P0
C3
C4

waituntil (count >= 40)
Relay Signaling Rule

- buff.length = 64
- count = 10

Take 32 items
 Relay Signaling Rule

 buff.length = 64
 count = 10
 count < 40
Relay Signaling Rule

\[
\begin{align*}
\text{buff.length} &= 64 \\
\text{count} &= 10 \\
\text{C}_2 \\
\text{waituntil} (\text{count} \geq 40)
\end{align*}
\]
Before exiting a monitor

- Signal at most one thread waiting on a condition that has become true
Before exiting a monitor

Signal at most one thread waiting on a condition that has become true
Outline

1. Introduction

2. Our Approach
   - Evaluate Predicate: Closure
   - Avoid signalAll Calls: Relay Signaling Rule
   - Reduce Predicate Evaluations: Predicate Tagging

3. Results

4. Conclusions and Future Work
Three types of predicates:

1. Equivalence predicate: $x = 5, y = a$
2. Threshold predicate: $x > 8, y < b, z \geq c + 3$
3. None of above: $x \neq 3, \text{assertion.isTrue()}$
**Predicate Tagging**

- Three types of tags: *equivalence*, *threshold*, and *none*
- Convert every predicate into disjunctive normal form (DNF)
- Assign a tag to every conjunction
- Assignment order: *equivalence > threshold > none*

**Example:**

\[
((x < 5) \land (y = 3)) \lor ((x > 5) \land \text{foo1}()) \lor \text{foo2}()
\]

- \((x < 5) \land (y = 3))\)
- \((x > 5) \land \text{foo1}())\)
- \(\text{foo2}())\)
| $x = 5$ | 
| $x = 8$ | 
| $x \geq 6$ | 
| $x < 10$ | 
| $x = 9$ | 
| $x > 11$ | 
| $x \leq 4$ | 
| $x \leq 7$ | 
| $x = 3$ | 
| $x > 6$ |
Predicate Tagging

Hashtable for the equivalence tag with shared expression $x$
Predicate Tagging

<table>
<thead>
<tr>
<th>x = 5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 8</td>
<td>8</td>
</tr>
<tr>
<td>x ≥ 6</td>
<td>9</td>
</tr>
<tr>
<td>x &lt; 10</td>
<td>3</td>
</tr>
<tr>
<td>x = 9</td>
<td></td>
</tr>
<tr>
<td>x &gt; 11</td>
<td></td>
</tr>
<tr>
<td>x ≤ 4</td>
<td></td>
</tr>
<tr>
<td>x ≤ 7</td>
<td></td>
</tr>
<tr>
<td>x = 3</td>
<td></td>
</tr>
<tr>
<td>x &gt; 6</td>
<td></td>
</tr>
</tbody>
</table>

x = 8
Predicate Tagging

\[ x = 5 \]
\[ x = 8 \]
\[ x \geq 6 \]
\[ x < 10 \]
\[ x = 9 \]
\[ x > 11 \]
\[ x \leq 4 \]
\[ x \leq 7 \]
\[ x = 3 \]
\[ x > 6 \]
Predicate Tagging

Min-heap for the threshold tag with shared expression $x$

- $x = 5$
- $x = 8$
- $x \geq 6$
- $x < 10$
- $x = 9$
- $x > 11$
- $x \leq 4$
- $x \leq 7$
- $x = 3$
- $x > 6$
- $x = 4$

(6, $\geq$)
(11, $>$)
(6, $>$)

$x = 4$
Predicate Tagging

Min-heap for the threshold tag with shared expression $x$

- $x = 5$
- $x = 8$
- $x \geq 6$
- $x < 10$
- $x = 9$
- $x > 11$
- $x \leq 4$
- $x \leq 7$
- $x = 3$
- $x > 6$

$(6, \geq)$

$(11, >)$

$(6, >)$

$x = 6$
Min-heap for the threshold tag with shared expression $x$

| $x = 5$ |
| $x = 8$ |
| $x \geq 6$ |
| $x < 10$ |
| $x = 9$ |
| $x > 11$ |
| $x \leq 4$ |
| $x \leq 7$ |
| $x = 3$ |
| $x > 6$ |

$\langle 6, \geq \rangle$  
$\langle 11, > \rangle$  
$\langle 6, > \rangle$

$x = 6$
Max-heap for the threshold tag with shared expression $x$
Evaluation

Four different signaling approaches:

- **Explicit** Using the Java explicit-signal
- **Baseline** Automatic-signal relying on only one condition variable.
- **AutoSynch-T** Using closure and relay signaling rule but predicate tagging
- **AutoSynch** Using closure, relay signaling rule and predicate tagging
Three types of problems:

**Shared predicate**  
Depends only on shared variables  
- bounded-buffer, $H_2O$ problem

**Complex predicate**  
Depends on both shared and local variables  
- readers-writers, round-robin access pattern

**signalAll**  
Requires signalAll calls  
- parameterized bounded-buffer
Evaluation: Shared Predicate

Bounded Buffer Problem

- Explicit
- AutoSynch
- AutoSynch-T
- Baseline

H₂O Problem

- Explicit
- AutoSynch
- AutoSynch-T
- Baseline

# consumers/producers

runtime (seconds)
Evaluation: Complex Predicate

Round-Robin Access Pattern

- Explicit
- AutoSynch
- AutoSynch-T

Readers-Writers Problem

- Explicit
- AutoSynch
- AutoSynch-T
Evaluation: signalAll

Parameterized Bounded Buffer (Requires signalAll calls)

![Graph showing runtime vs. number of consumers]

- Explicit
- AutoSynch

![Graph showing context switches vs. number of consumers]

- Explicit
- AutoSynch

# context switches (× 1000)

0 2 4 8 16 32 64 128 256

# consumers
Workload Simulation

- Evaluate performance of AutoSynch under different workloads
- Perform other operations out of the monitor between every two monitor operations
- Report runtime ratio with respect to Explicit
Evaluation: Workload

Round-Robin Access Pattern

Readers-Writers Problem

- Explicit
- AutoSynch
- AutoSynch-T

# delay time (microseconds)
Conclusions and Future Work

Conclusions

- We propose AutoSynch that supports automatic signaling with simple syntax
- AutoSynch is almost as efficient as the explicit-signal or even more efficient

Future work

- Use the architecture information
- Implement AutoSynch directly in JVM