

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

THE UNIVERSITY OF
TEXAS
— AT AUSTIN —

1 Introduction

2 Our Approach

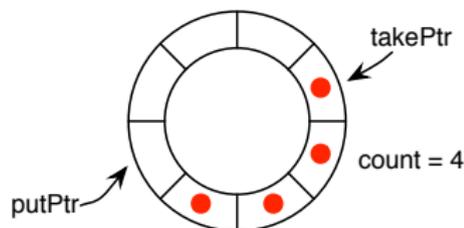
- Evaluate Predicate: Closure
- Avoid signalAll Calls: Relay Signaling Rule
- Reduce Predicate Evaluations: Predicate Tagging

3 Results

4 Conclusions and Future Work

Bounded Buffer

```
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;  
    }  
}
```



Bounded Buffer

```
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();  
}
```

Bounded Buffer

```
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();
}
```

Bounded Buffer: Common Bugs

```
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();  
}
```

Bounded Buffer: Common Bugs

```
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();
}
```

Bounded Buffer: Common Bugs

```
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();
}
```

AutoSynch Bounded Buffer

```
public AutoSynch class BoundedBuffer {  
    Object[] buff;  
    int putPtr, takePtr, count;  
    public BoundedBuffer(int n) {  
        buff = new Object[n];  
        putPtr = takePtr = count = 0 ;  
    }  
}
```

AutoSynch Bounded Buffer

```
public Object take() {  
    waituntil (count > 0);  
    Object ret = buff[takePtr++];  
    takePtr %= buff.length;  
    count--;  
}
```

AutoSynch vs. Explicit Signaling

```
1 public AutoSynch class BoundedBuffer {
2   Object[] buff;
3   int putPtr, takePtr, count;
4   public BoundedBuffer(int n) {
5     buff = new Object[n];
6     putPtr = takePtr = count = 0 ;
7   }
8   public Object take() {
9     waituntil (count > 0);
10    Object ret = buff[takePtr++];
11    takePtr %= buff.length;
12    count--;
13  }
14 }

1 public class BoundedBuffer {
2   Object[] buff;
3   int putPtr, takePtr, count;
4   Lock mutex = new ReentrantLock();
5   Condition full = mutex.newCondition();
6   Condition empty = mutex.newCondition();
7   public BoundedBuffer(int n) {
8     buff = new Object[n];
9     putPtr = takePtr = count = 0;
10  }
11  public Object take() {
12    mutex.lock();
13    while (count == 0) {
14      empty.await();
15    }
16    Object ret = buff[takePtr++];
17    takePtr %= buff.length;
18    count--;
19    if (count == buff.length - 1) {
20      full.signalAll();
21    }
22    mutex.unlock();
23  }
24 }
```

- 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)

Reduce number of context switches and predicate evaluations

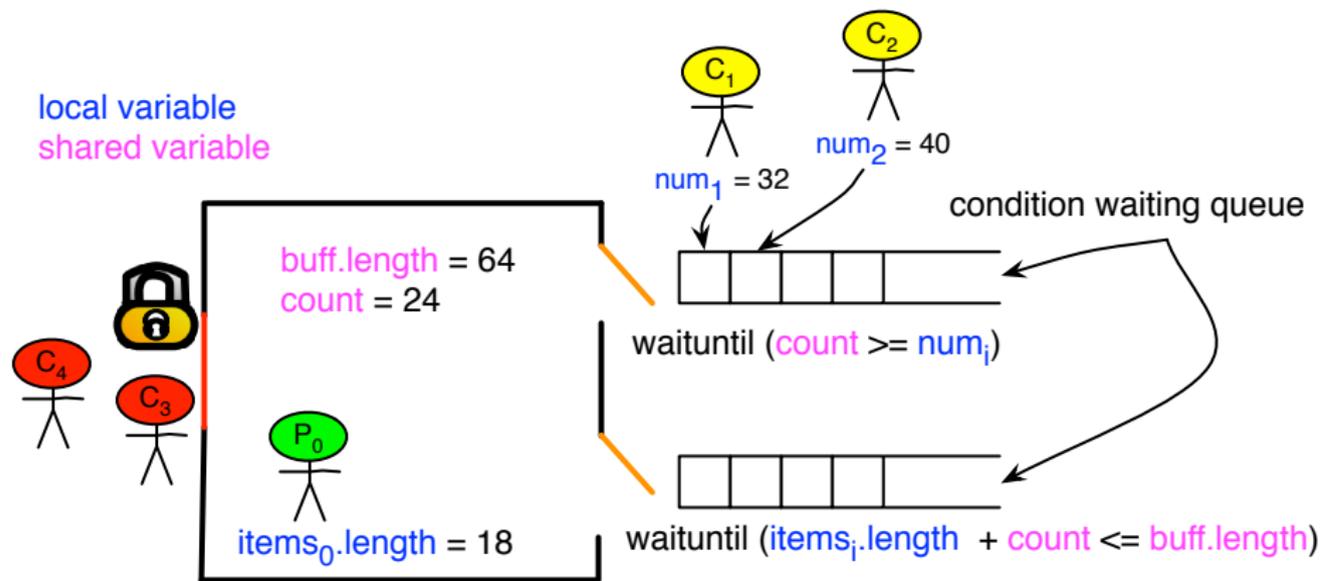
Predicate evaluation

- 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

Parameterized Bounded Buffer

```
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;
}
```

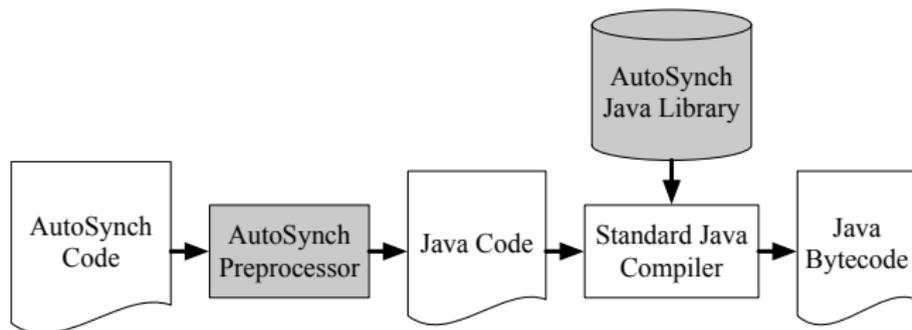
Parameterized Bounded Buffer

```
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;
}
```

Parameterized Bounded Buffer

```
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



1 Introduction

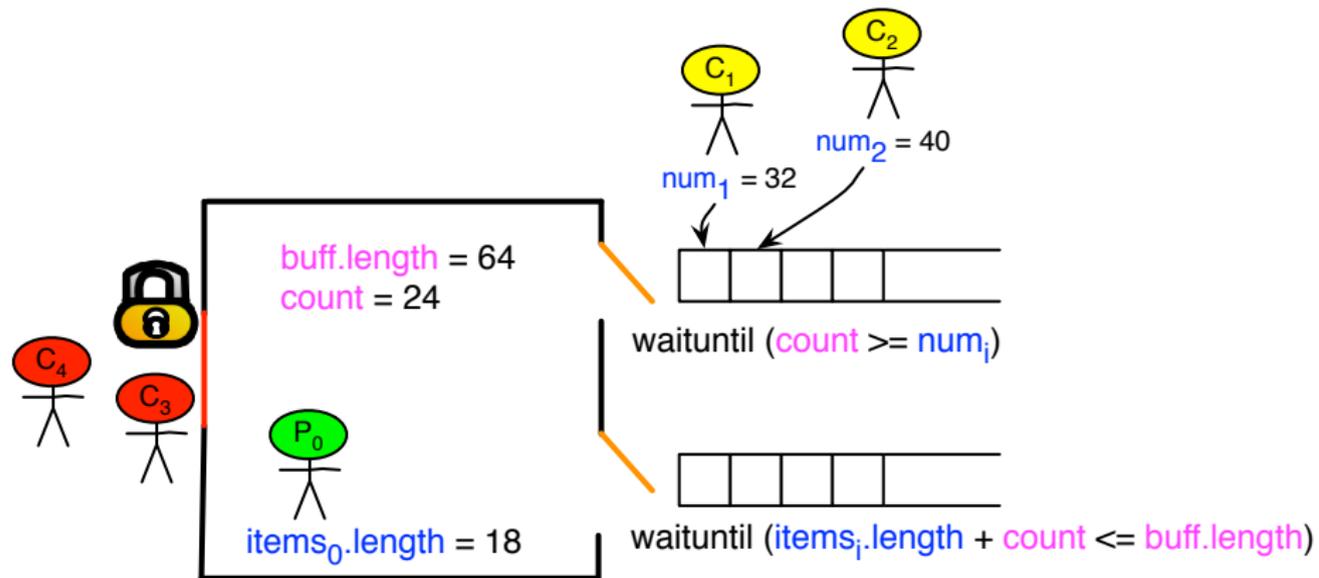
2 Our Approach

- Evaluate Predicate: Closure
- Avoid signalAll Calls: Relay Signaling Rule
- Reduce Predicate Evaluations: Predicate Tagging

3 Results

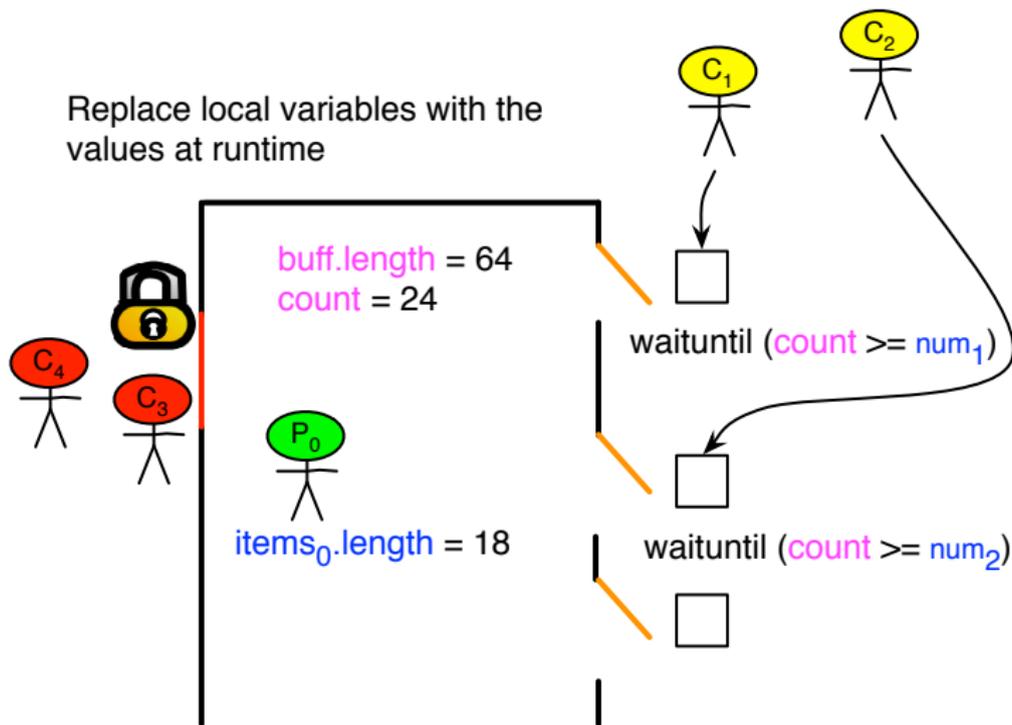
4 Conclusions and Future Work

Closure



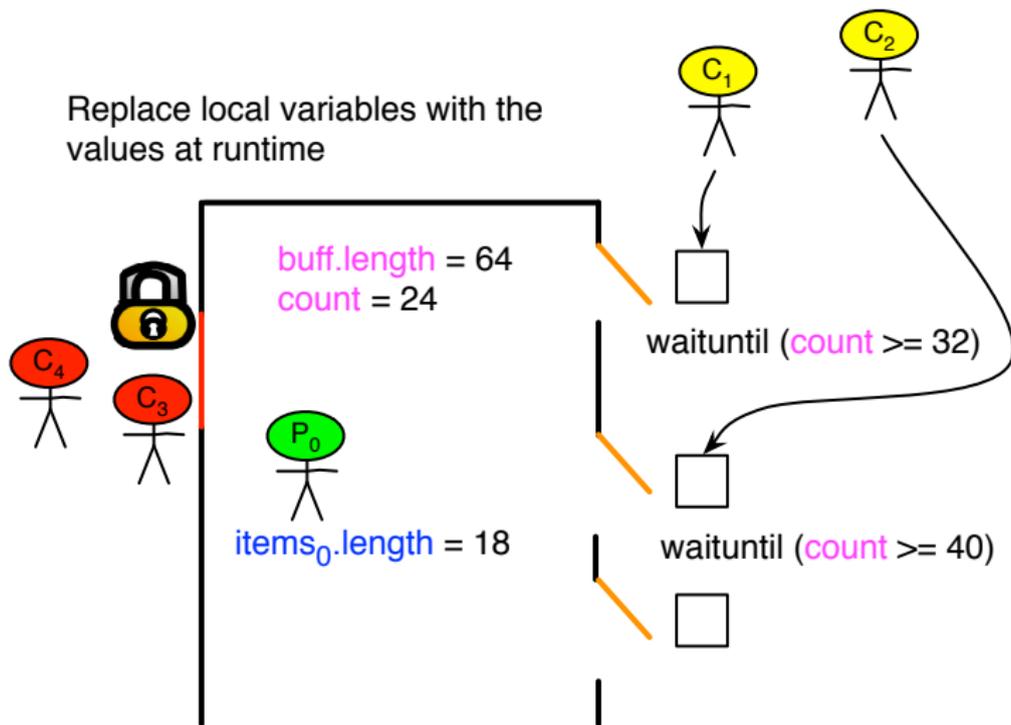
Closure

Replace local variables with the values at runtime



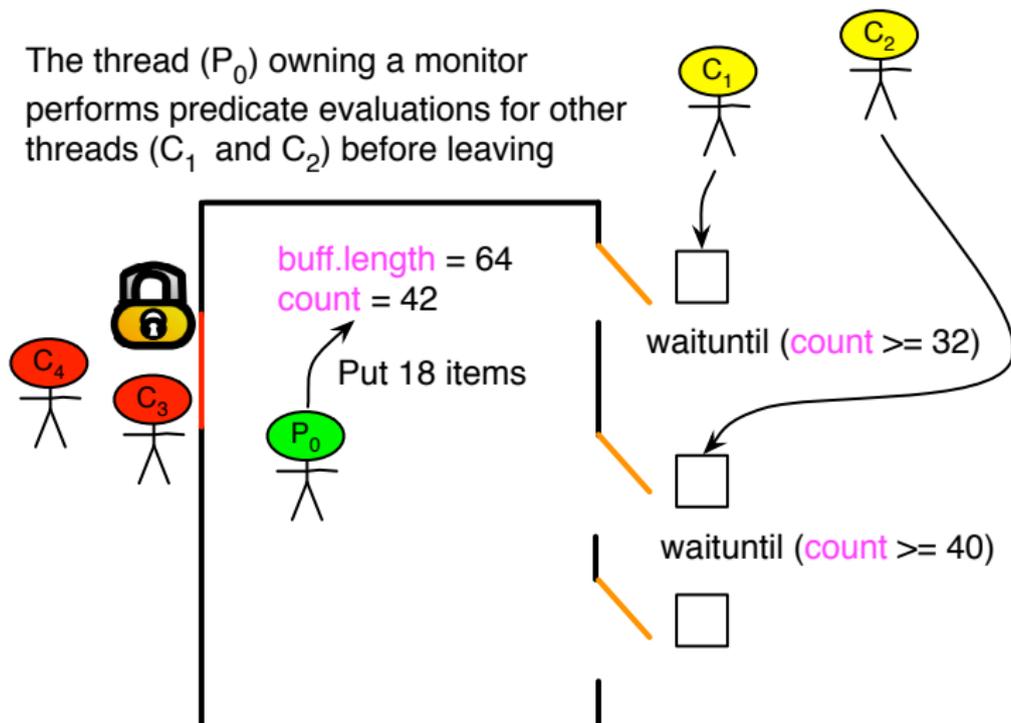
Closure

Replace local variables with the values at runtime



Closure

The thread (P_0) owning a monitor performs predicate evaluations for other threads (C_1 and C_2) before leaving



1 Introduction

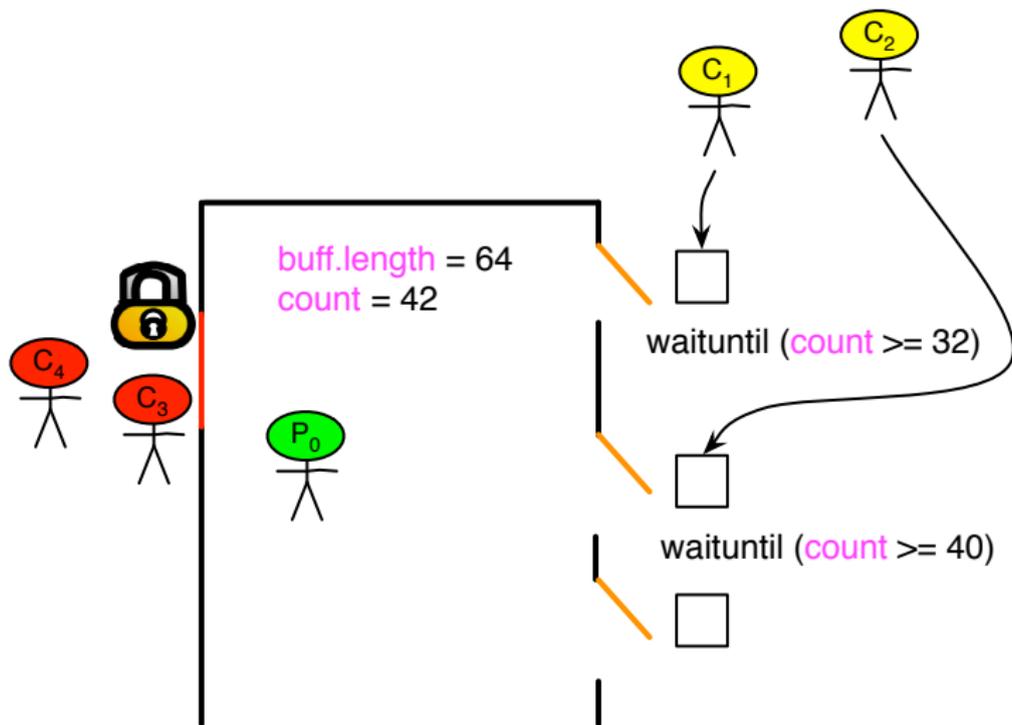
2 Our Approach

- Evaluate Predicate: Closure
- Avoid signalAll Calls: Relay Signaling Rule
- Reduce Predicate Evaluations: Predicate Tagging

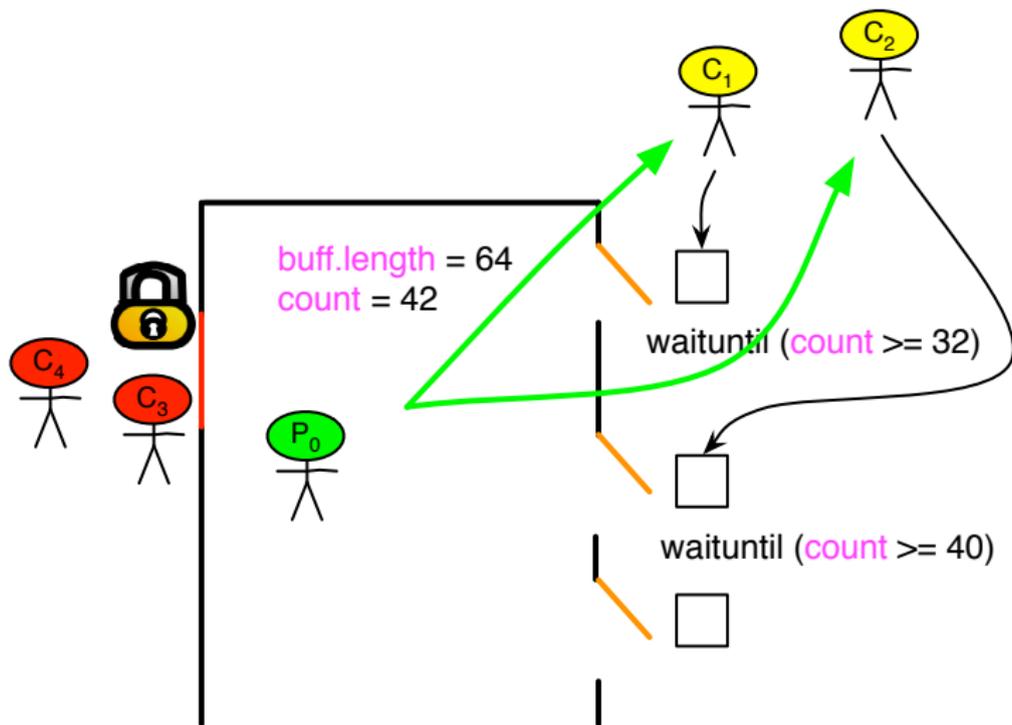
3 Results

4 Conclusions and Future Work

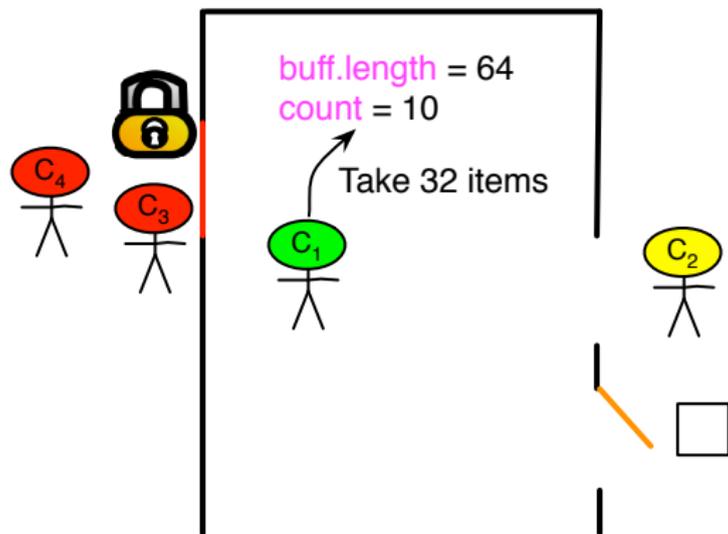
Relay Signaling Rule



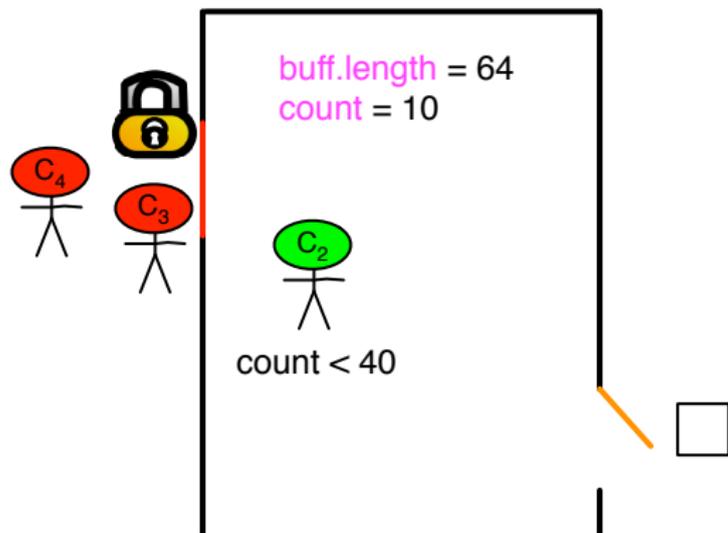
Relay Signaling Rule



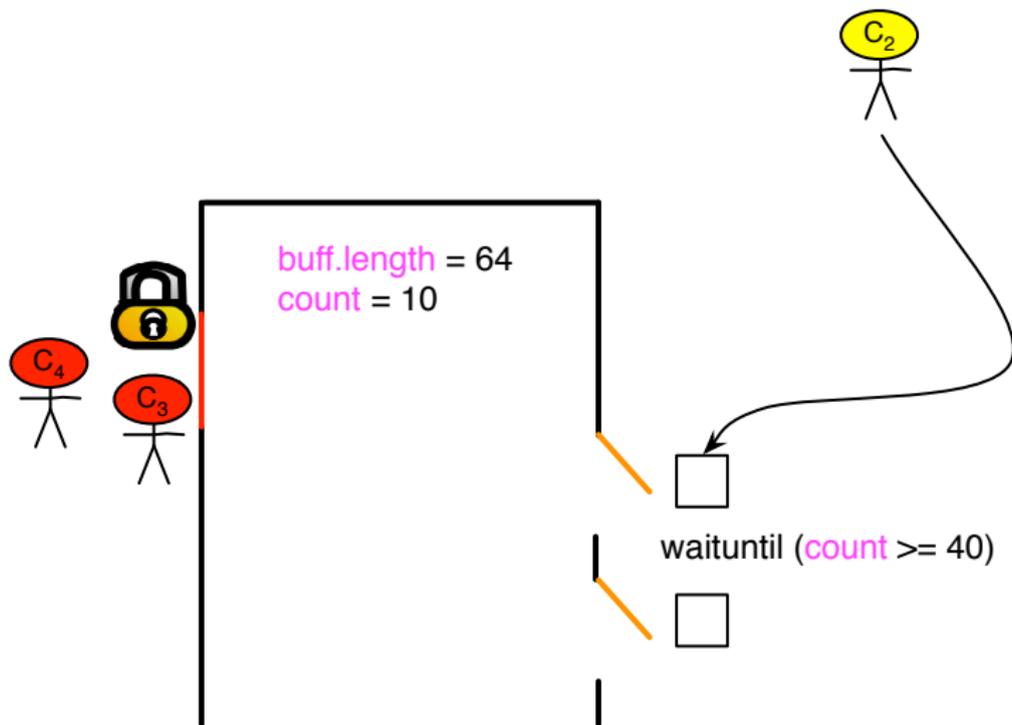
Relay Signaling Rule



Relay Signaling Rule



Relay Signaling Rule



Relay Signaling Rule

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

Relay Signaling Rule

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

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

Predicate Tagging

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
- e.g.
 $((x < 5) \wedge (y = 3)) \vee ((x > 5) \wedge \text{foo1}()) \vee \text{foo2}()$
 - ▶ $((x < 5) \wedge (y = 3))$
 - ▶ $((x > 5) \wedge \text{foo1}())$
 - ▶ $\text{foo2}()$

Predicate Tagging

$$x = 5$$

$$x = 8$$

$$x \geq 6$$

$$x < 10$$

$$x = 9$$

$$x > 11$$

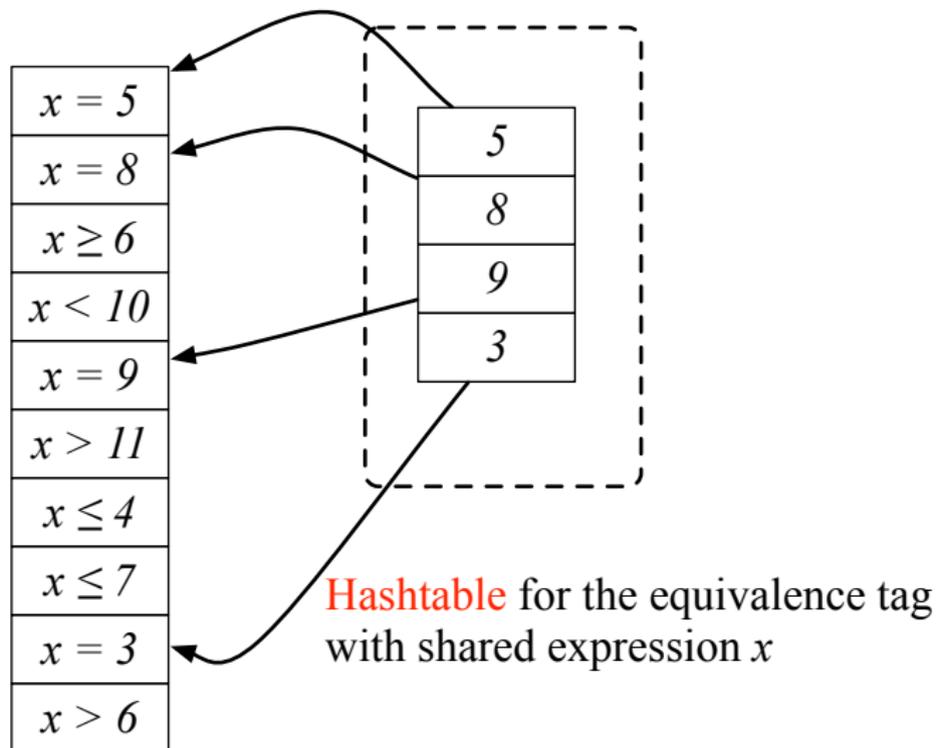
$$x \leq 4$$

$$x \leq 7$$

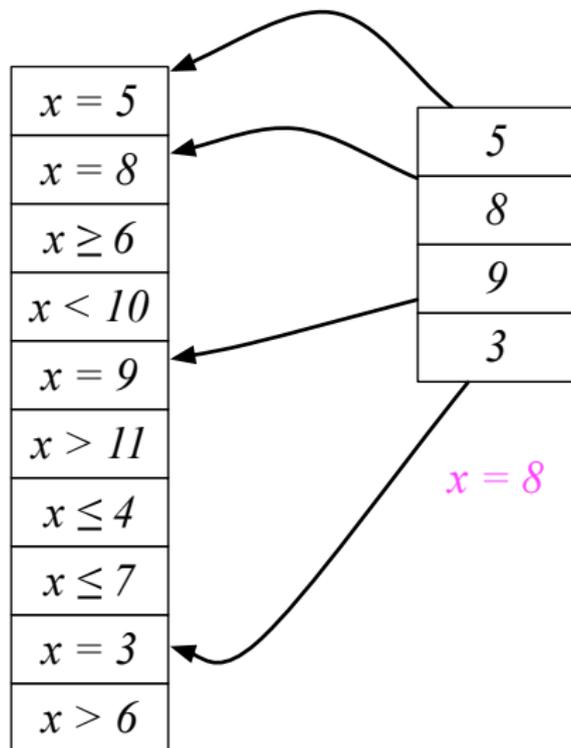
$$x = 3$$

$$x > 6$$

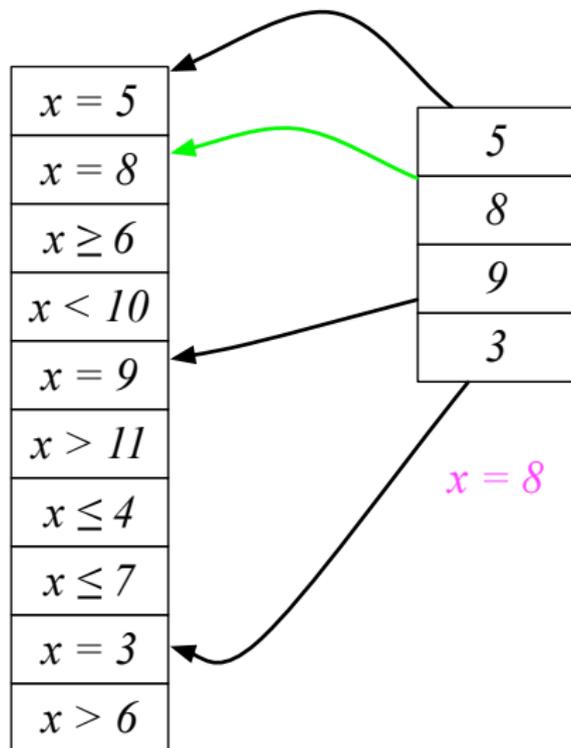
Predicate Tagging



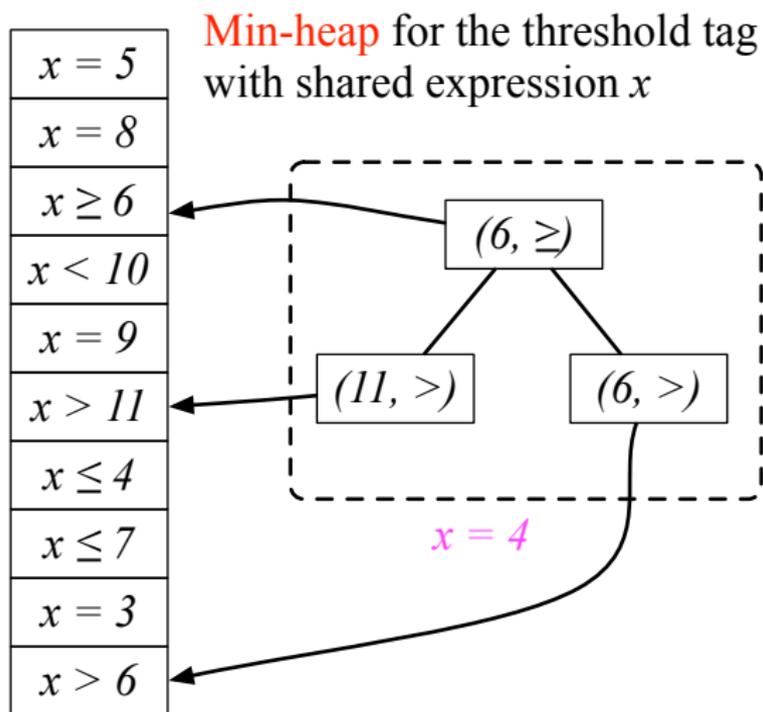
Predicate Tagging



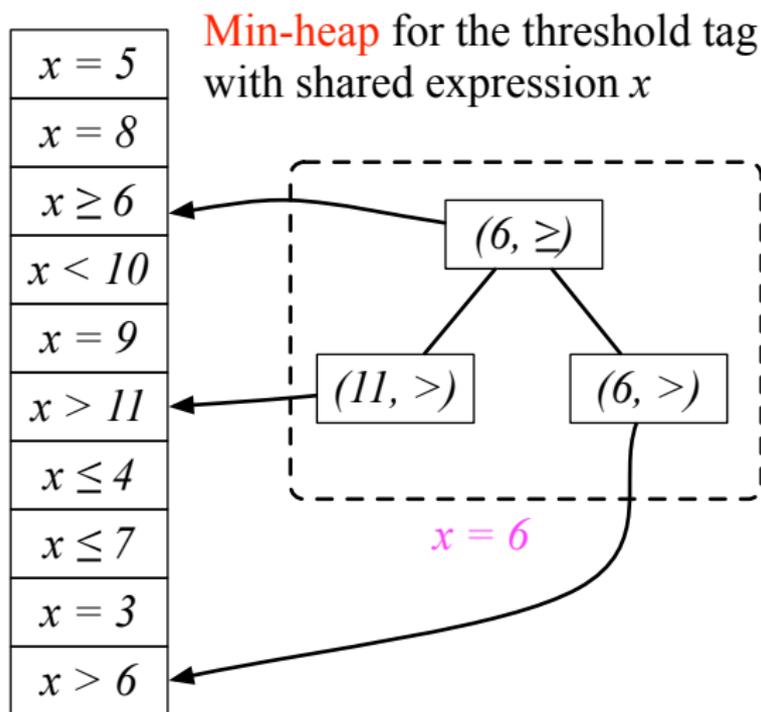
Predicate Tagging



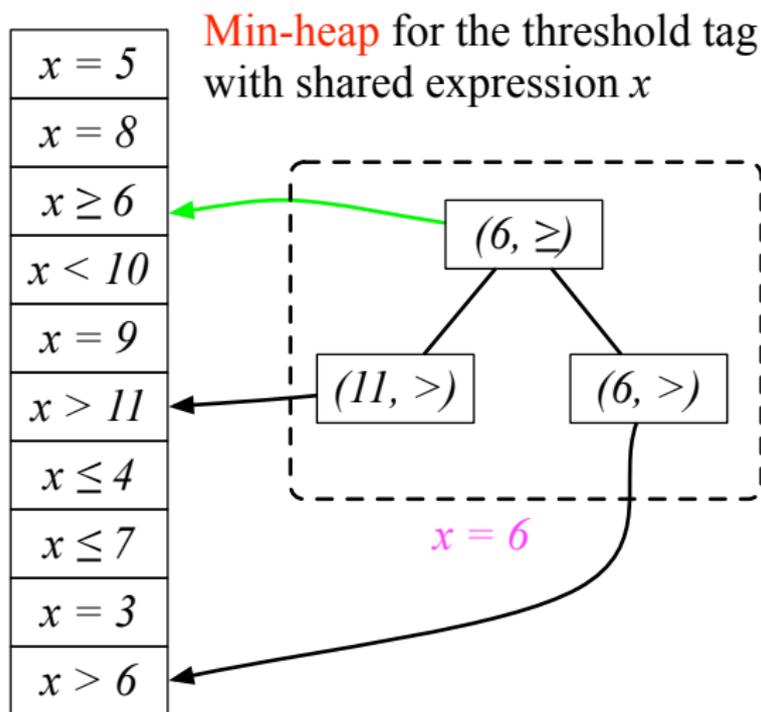
Predicate Tagging



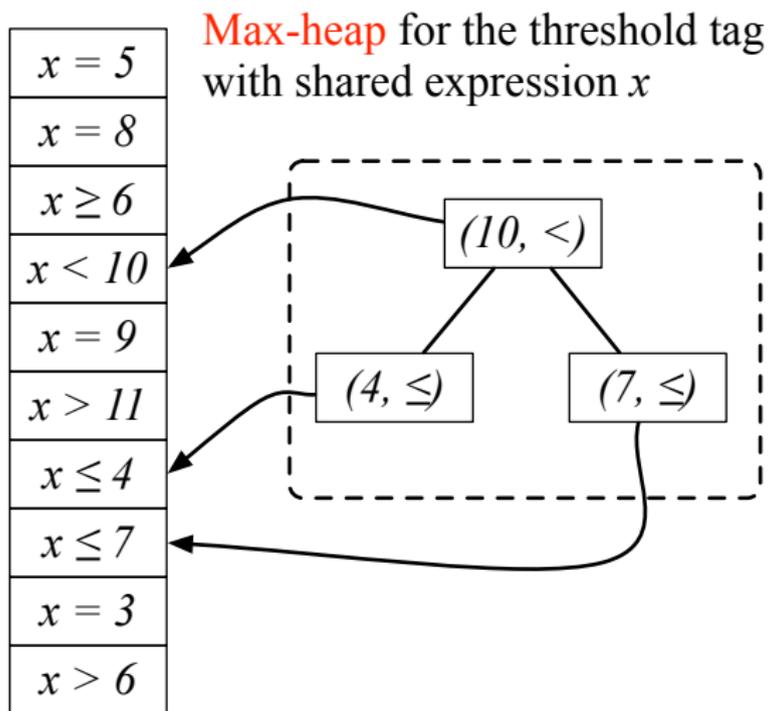
Predicate Tagging



Predicate Tagging



Predicate Tagging



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

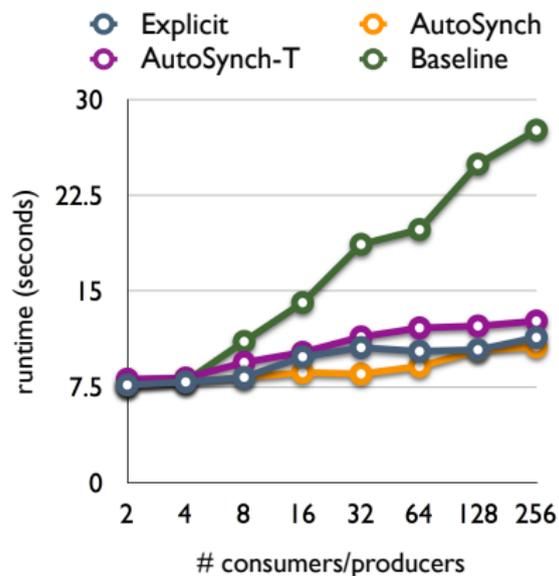
- readers-writers, round-robin access pattern

signalAll Requires signalAll calls

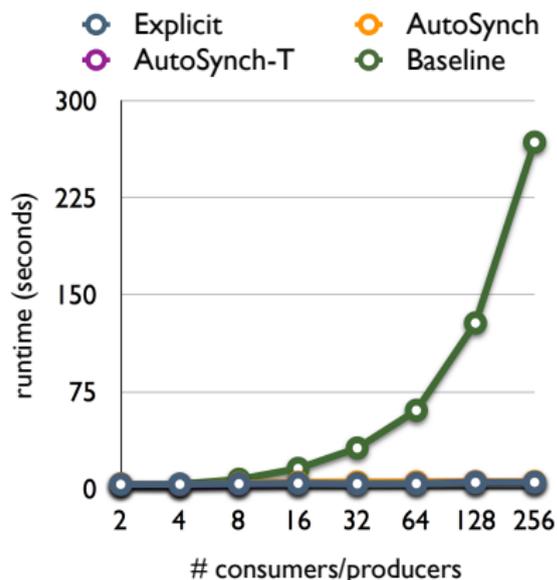
- parameterized bounded-buffer

Evaluation: Shared Predicate

Bounded Buffer Problem

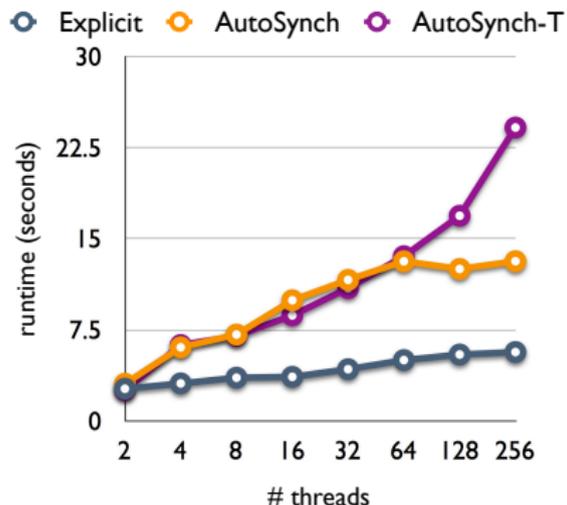


H₂O Problem

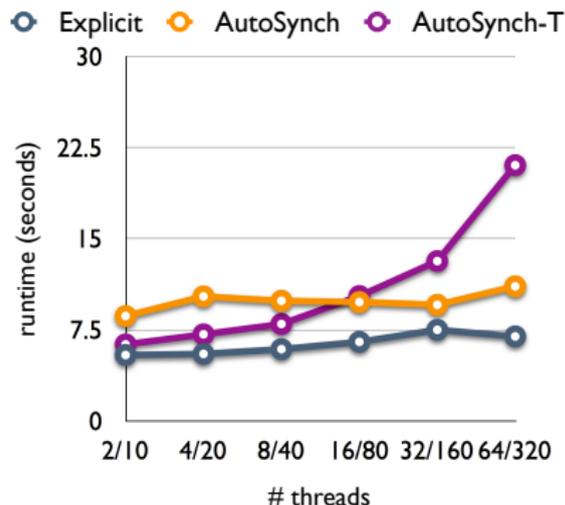


Evaluation: Complex Predicate

Round-Robin Access Pattern

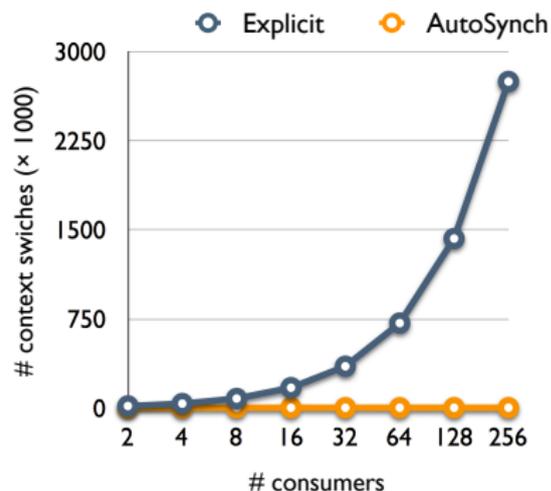
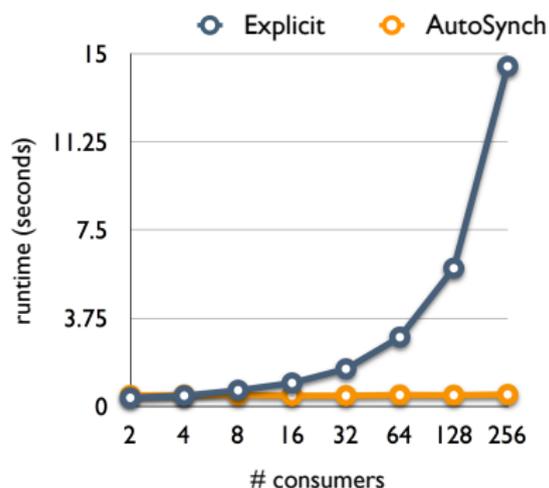


Readers-Writers Problem



Evaluation: signalAll

Parameterized Bounded Buffer (Requires signalAll calls)

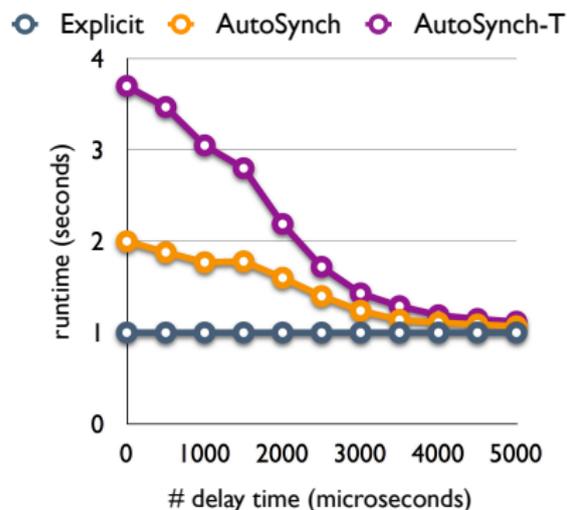


Workload Simulation

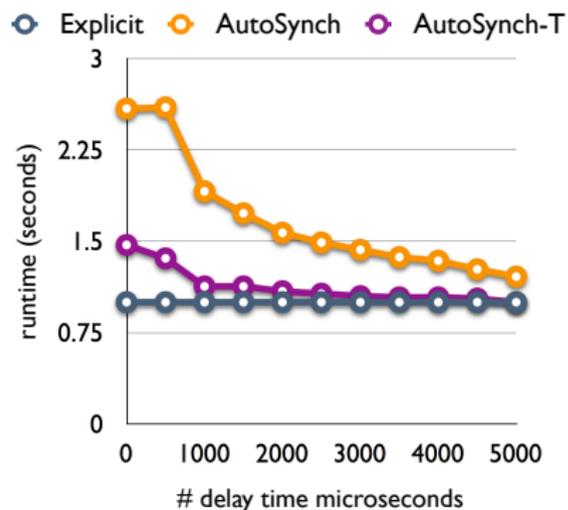
- Evaluate performance of AutoSynch under different workloads
- Perform other operations out of the monitor between every two monitor operations
- Report runtime ratio with respect to Explicit

Evaluation: Workload

Round-Robin Access Pattern



Readers-Writers Problem



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