Fine-grained parallelism requires communication
Fine-grained parallelism requires communication.
Fine-grained parallelism requires communication
Fine-grained parallelism requires communication

**Barrier**

**Exchanger**
Fine-grained parallelism requires communication

Barrier

Exchanger

Lock
Fine-grained parallelism requires communication
Fine-grained parallelism requires communication

- Synchronization protocols
- Blocking & context switches
- Cache coherence & memory bandwidth
State of the art?

Leave it to the experts:

Research Literature

Industrial-strength Libraries

java.util.concurrent
.NET 4.0
Intel TBB
The problem

Libraries are an enormous undertaking, not extensible or customizable by users
This work

Use *join patterns* for synchronization:

**Expressive**
- Write synchronization primitives declaratively and concisely

**Scalable**
- Competitive with industrial libraries;
  can recover existing algorithms

[Fournet & Gonthier]
Expressive
Joins: a crash course

class Lock {
    public Synchronous.Channel Acquire = new ...
    public Asynchronous.Channel Release = new ...
    public Lock() {
        When(Acquire).And(Release).Do(() => {});
    }

    // initially available
    Release();
}

Joins: a crash course

class Lock {
    public Synchronous.Channel Acquire = new ...
    public Asynchronous.Channel Release = new ...
    public Lock() {
        When(Acquire).And(Release).Do(() => {});
        // initially available
        Release();
    }
}

Join pattern
Joins: a crash course

class Lock {
    public Synchronous.Channel Acquire = new ...
    public Asynchronous.Channel Release = new ...
    public Lock() {
        When(Acquire).And(Release).Do(() => {});

        // initially available
        Release();
    }
}

Join body
Joins: a crash course

class Lock {
    public Synchronous.Channel Acquire = new ...
    public Asynchronous.Channel Release = new ...
    public Lock() {
        When(Acquire).And(Release).Do(() => {});

        // initially available
        Release();
    }
}

Message send
Joins: a crash course

class Semaphore {
    public Synchronous.Channel Acquire = new ... 
    public Asynchronous.Channel Release = new ... 
    public Semaphore(int n) {
        When(Acquire).And(Release).Do(() => {});

        // initially n available 
        for (; n > 0; n--) Release();
    }
}
Joins: a crash course

class ProducerConsumer<T> {
    public Synchronous<T>.Channel Get = new ...
    public Asynchronous.Channel<T> Put = new ...
    public ProducerConsumer() {
        When(Get).And(Put).Do(t => t);
    }
}
Joins: a crash course

class ProducerConsumer\<T\> {
    public Synchronous\<T\>.Channel Get = new ...  
    public Asynchronous.Channel\<T\> Put = new ...  
    public ProducerConsumer() {
        When(Get).And(Put).Do(t => t);
    }
}

Join: a crash course

class ProducerConsumer<T> {
    public Synchronous<T>.Channel Get = new ... 
    public Asynchronous.Channel<T> Put = new ... 
    public ProducerConsumer() {
        When(Get).And(Put).Do(t => t);
    }
}
Joins: a crash course

Synchronous.Channel[] hungry = new ...
Asynchronous.Channel[] chopstick = new ...

for (int i = 0; i < n; i++) {
    var left = chopstick[i];
    var right = chopstick[(i+1) % n];
    When(hungry[i]).And(left).And(right).Do(() => {
        eat(); left(); right();
    });
}
Scalable
Existing joins implementations

Coarse-grained locks:
Serialized matching
Extra memory bus traffic
Our implementation (in C#)

Key idea:

Messages are resources
Our implementation (in C#)

Key idea:

Messages are resources

Store in lock-free bags

⇒ parallelized matching

⇒ decreased communication
Our implementation (in C#)

Key idea:

Messages are resources

Store in lock-free bags

- parallelized matching
- decreased communication

Acquire in global order
When(Get).And(PutA).And(PutB).Do(....)
When(Get).And(PutA).And(PutB).Do(...)

Get

PutA

PutB
When(Get).And(PutA).And(PutB).Do(...)

Get

PutA

PutB

Pending
When(Get).And(PutA).And(PutB).Do(...)

Get  PutA  PutB

Pending  Pending
When(Get).And(PutA).And(PutB).Do(....)
When(Get).And(PutA).And(PutB).Do(....)

Get

Claimed

PutA

Pending

PutB

Pending
When(Get).And(PutA).And(PutB).Do(...)

Get  
- Claimed

PutA  
- Claimed

PutB  
- Pending
When(Get).And(PutA).And(PutB).Do(....)

Get

PutA

PutB

Claimed

Claimed

Claimed
When(Get).And(PutA).And(PutB).Do(...)

Get

Consumed

PutA

Consumed

PutB

Consumed
When(Get).And(PutA).And(PutB).Do(....)
When\((Get)\).And\((PutA)\).And\((PutB)\).Do(....)

Get

PutA

PutB

Pending

Pending

Pending

Pending

Pending

Pending

Pending
When( Get ). And( PutA ). And( PutB ). Do( . . . )
When:Get).And(PutA).And(PutB).Do(...)

Get
- Claimed
- Pending
- Pending

PutA
- Pending
- Pending

PutB
- Pending
- Pending
When(Get).And(PutA).And(PutB).Do(...)

Get
- Claimed
- Pending
- Claimed
- Pending

PutA
- ???
- Pending
- Pending

PutB
- Pending
- Pending
When(Get).And(PutA).And(PutB).Do(...)
When(Get).And(PutA).And(PutB).Do(...)
When(Get).And(PutA).And(PutB).Do(...)

- Get
  - Claimed
  - Pending
  - Claimed
  - Pending
- PutA
  - Claimed
- PutB
  - Claimed
When(Get).And(PutA).And(PutB).Do(....)
When(Get).And(PutA).And(PutB).Do(....)
When(Get).And(PutA).And(PutB).Do(...)

Get

PutA

PutB

Pending

Pending

Pending

Pending

Pending

Pending
When(Get).And(PutA).And(PutB).Do(....)
When(Get).And(PutA).And(PutB).Do(....)
When(Get).And(PutA).And(PutB).Do(....)

Get

- Claimed
- Pending
- Claimed
- Pending

PutA

- Claimed

PutB

- ???
When(Get).And(PutA).And(PutB).Do(...)

Get

- Claimed
- Pending
- Claimed
- Pending

PutA

- Claimed
- Claimed

PutB

- Claimed
When(Get).And(PutA).And(PutB).Do(....)

Get
- Claimed
- Pending
- Claimed
- Pending

PutA
- Pending
- Claimed

PutB
- Claimed
When(Get).And(PutA).And(PutB).Do(...)
The Protocol

1. Add message
2. Match?
   - Yes: Claim?
     - Yes: Fire pattern
     - No: Give up
   - No: Add message

18
The Protocol

Add message

Match?

No

Give up
The Protocol

Add message

Match?

No

Give up

Add message

Match?

No

Give up
The Protocol

Add message

Match?

No

Give up

Add message

Match?

No

Give up
The Paper

• “Lazy” message adding
• Stealing
• Counter channels
The Paper

- “Lazy” message adding
- Stealing
- Counter channels

Recover existing algorithms
Benchmarks

48 core AMD machine

Simulated fine-grained parallel workload
Semaphore

Throughput (iters/10μs)

Threads

Scalable Joins
Locking Joins
.NET
Producer/Consumer

Throughput (iters/10μs)

Threads

Scalable Joins
Locking Joins
.NET
This work

Use *join patterns* for synchronization:

**Expressive**
Write synchronization primitives declaratively and concisely

**Scalable**
Competitive with industrial libraries; can recover existing algorithms

[Fournet & Gonthier]
The Takeaway

Have your cake and eat it, too

Just one algorithm to implement, endlessly extensible by the user, competitive with custom solutions
The Takeaway
Have your cake and eat it, too

Just one algorithm to implement, endlessly extensible by the user, competitive with custom solutions

Thank you
Producer/Consumer (no work)

Throughput (iters/10μs)

Threads

Scalable Joins
Locking Joins
.NET-queue
.NET-bag