Channels – the CSP/CML Model

Radu Nicolescu
Department of Computer Science
University of Auckland

28 March 2019
31 March 2019

Outline

CSP (wiki)

- Communicating sequential processes (CSP) is a formal language for describing patterns of interaction in concurrent systems
  - inspired channels, rendezvous in programming
  - family of process calculi/algebras: CSP, CCS, π, join, ...
- CSP was first described in a 1978 paper by Tony Hoare, but has since evolved substantially
- Tony Hoare
  - algorithm: quicksort
  - formal language: CSP
  - concurrency: monitors, dining philosophers
  - programming languages: occam language, Algol W
  - **null** = billion dollar mistake!

CSP (wiki)

- Communicating sequential processes (CSP) is a formal language for describing patterns of interaction in concurrent systems
  - inspired channels, rendezvous in programming
  - family of process calculi/algebras: CSP, CCS, π, join, ...
- CSP was first described in a 1978 paper by Tony Hoare, but has since evolved substantially
- Tony Hoare
  - algorithm: quicksort
  - formal language: CSP
  - concurrency: monitors, dining philosophers
  - programming languages: occam language, Algol W
  - **null** = billion dollar mistake!

CSP (wiki)

- Communicating sequential processes (CSP) is a formal language for describing patterns of interaction in concurrent systems
  - inspired channels, rendezvous in programming
  - family of process calculi/algebras: CSP, CCS, π, join, ...
- CSP was first described in a 1978 paper by Tony Hoare, but has since evolved substantially
- Tony Hoare
  - algorithm: quicksort
  - formal language: CSP
  - concurrency: monitors, dining philosophers
  - programming languages: occam language, Algol W
  - **null** = billion dollar mistake!

CML (wiki)

- First functional language: LISP (John McCarthy, 1958) = List Processor (Lots of Insipid Stupid Parentheses ☺)
- LISP family: LISP, Scheme, Clojure... JavaScript!
- ML family: ML (Meta Language), Standard ML, CAML (Categorical Abstract Machine Language), OCAML (Object CAML), Scala, F# (OCAML for .NET)
- John Reppy CML (Concurrent ML): channels, rendezvous, events from CSP
- Channels: CML (language), Go (language), F# Hopac (library), C# Channels (library), JavaScript, Rust, Kotlin, ...
- Rendezvous, Joins: Ada, JoCAML (Joins CAML), Cω, ...

CML (wiki)

- First functional language: LISP (John McCarthy, 1958) = List Processor (Lots of Insipid Stupid Parentheses ☺)
- LISP family: LISP, Scheme, Clojure... JavaScript!
- ML family: ML (Meta Language), Standard ML, CAML (Categorical Abstract Machine Language), OCAML (Object CAML), Scala, F# (OCAML for .NET)
- John Reppy CML (Concurrent ML): channels, rendezvous, events from CSP
- Channels: CML (language), Go (language), F# Hopac (library), C# Channels (library), JavaScript, Rust, Kotlin, ...
- Rendezvous, Joins: Ada, JoCAML (Joins CAML), Cω, ...

CML (wiki)

- First functional language: LISP (John McCarthy, 1958) = List Processor (Lots of Insipid Stupid Parentheses ☺)
- LISP family: LISP, Scheme, Clojure... JavaScript!
- ML family: ML (Meta Language), Standard ML, CAML (Categorical Abstract Machine Language), OCAML (Object CAML), Scala, F# (OCAML for .NET)
- John Reppy CML (Concurrent ML): channels, rendezvous, events from CSP
- Channels: CML (language), Go (language), F# Hopac (library), C# Channels (library), JavaScript, Rust, Kotlin, ...
- Rendezvous, Joins: Ada, JoCAML (Joins CAML), Cω, ...

CML (wiki)

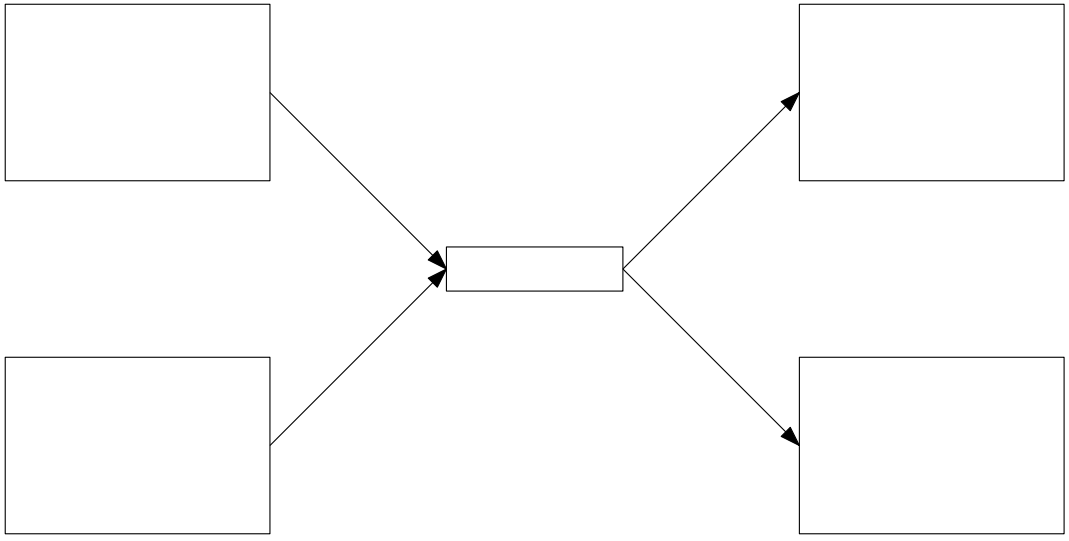
- First functional language: LISP (John McCarthy, 1958) = List Processor (Lots of Insipid Stupid Parentheses ☺)
- LISP family: LISP, Scheme, Clojure... JavaScript!
- ML family: ML (Meta Language), Standard ML, CAML (Categorical Abstract Machine Language), OCAML (Object CAML), Scala, F# (OCAML for .NET)
- John Reppy CML (Concurrent ML): channels, rendezvous, events from CSP
- Channels: CML (language), Go (language), F# Hopac (library), C# Channels (library), JavaScript, Rust, Kotlin, ...
- Rendezvous, Joins: Ada, JoCAML (Joins CAML), Cω, ...

CML (wiki)

- First functional language: LISP (John McCarthy, 1958) = List Processor (Lots of Insipid Stupid Parentheses ☺)
- LISP family: LISP, Scheme, Clojure... JavaScript!
- ML family: ML (Meta Language), Standard ML, CAML (Categorical Abstract Machine Language), OCAML (Object CAML), Scala, F# (OCAML for .NET)
- John Reppy CML (Concurrent ML): channels, rendezvous, events from CSP
- Channels: CML (language), Go (language), F# Hopac (library), C# Channels (library), JavaScript, Rust, Kotlin, ...
- Rendezvous, Joins: Ada, JoCAML (Joins CAML), Cω, ...

Channels – bird's eye view

- Named communication channels between concurrent tasks



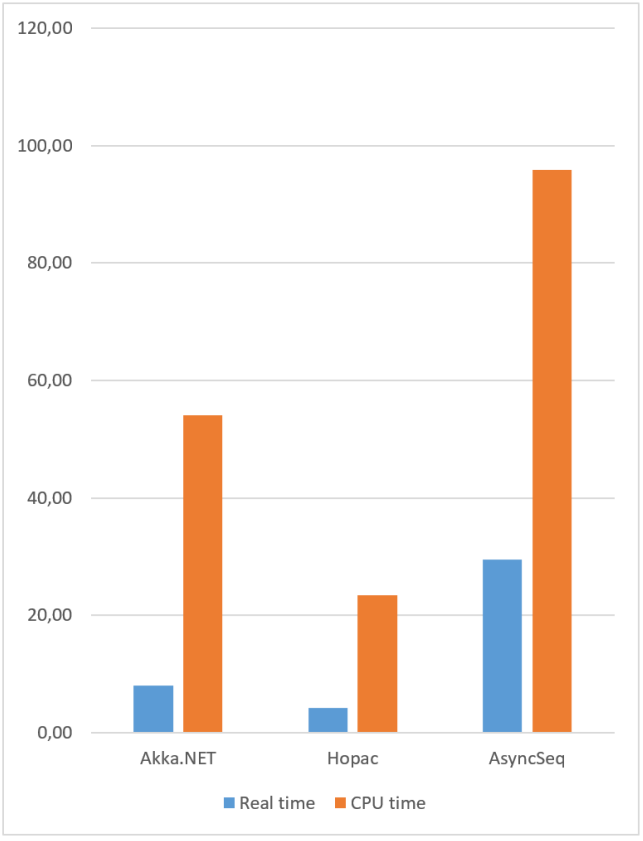
- Default: multiple writers, aka multiple readers
- Default: zero-size buffer, aka rendezvous (sync handshake)
- Extensions: bounded buffers, even unbounded ($\approx$ actors)
- Many lightweight async tasks (aka green threads)
- Theory: channels $\equiv$ actors (but shine for different apps)

Outline

Hopac Readings

- Hopac repository
  `https://github.com/Hopac/Hopac`
- Hopac Programming Guide `https://github.com/Hopac/Hopac/blob/master/Docs/Programming.md`
- Hopac Reference – monads, high-order combinatorics...
  `https://hopac.github.io/Hopac/Hopac.html`
- Demistify FP – Hopac
  `https://www.demystifyfp.com/tags/hopac/`

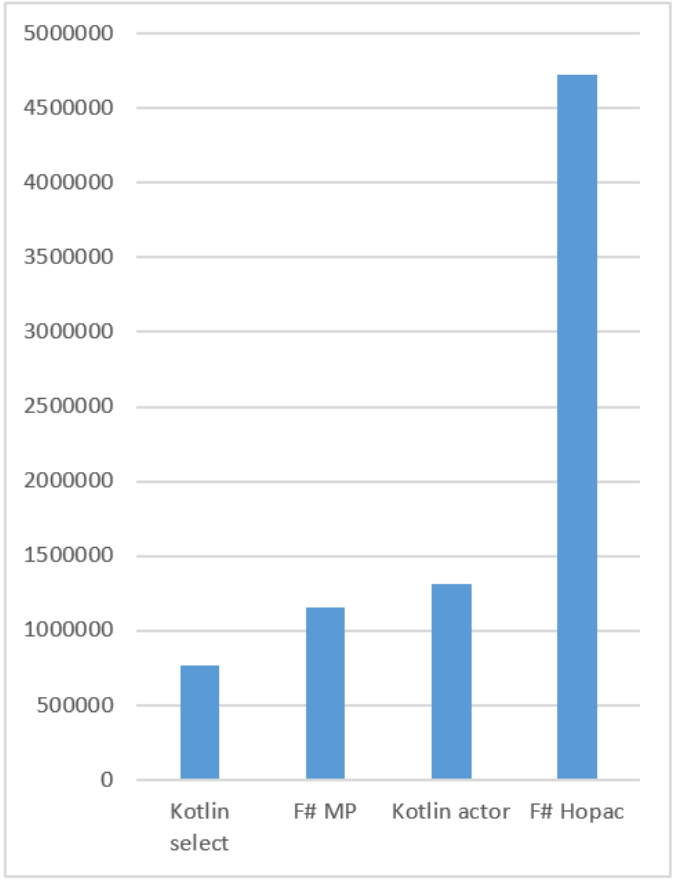
Peformance – Hopac vs Akka



Stream performance – the lower the better

`http://vaskir.blogspot.com/2016/05/akkanet-streams-vs-hopac.html`

Peformance – Hopac vs Mailbox vs Kotlin



Actor-like performance – the higher the better

`https://vasily-kirichenko.github.io/fsharpblog/actors`

Hopac Basics

- Create a default typed channel

```
1 let ch = Ch<string> ()
```

- A job = lightweight **async** task (as a sugared monad) here w/ async sleep

```
1 job {
2   do! timeOutMillis 100
3 }
```

- Communicate via a channel

```
1 job {
2   do! ch *<- m // do! Ch.give ch m     // sync post
3   do! ch *<+ m // do! Ch.send ch m     // async post
4   let! m = ch  // let! m = Ch.take ch  // receive
5 }
```

Outline

C# Channels Readings

- MSDN System.Threading.Channels Namespace
  `https://docs.microsoft.com/en-us/dotnet/api/system.threading.channels?view=dotnet-plat-ext-2.1`
- Github System.Threading.Channels
  `https://github.com/dotnet/corefx/tree/master/src/System.Threading.Channels`
- Exploring System.Threading.Channels
  `https://ndportmann.com/system-threading-channels/`

C# Channels Basics

- Create a size 1 one-to-one typed channel (no size 0)

```
1 var opt = new BoundedChannelOptions (1)
2     { SingleWriter=true, SingleReader=true ,};
3 var ch = Channel.CreateBounded<string> (opt);
```

- Use the existing async/await framework
- Communicate via a channel

```
1 async Task ... {
2   await ch.Writer.WriteAsync (m);          // async post
3
4   var m = await ch.Reader.ReadAsync (); // receive
5 }
```

Outline

Counter – F# Mailbox

- actor = inbox + async function

```
1  let agent = MailboxProcessor.Start (fun inbox ->
2    let rec loop count =
3      async {
4        let! msg = inbox.Receive()
5        do! Async.Sleep 100
6        return! loop (count+1)
7      }
8    loop 0
9    )
```

- easy to post message from outside

```
1  agent.Post m
```

Counter – F# Hopac

- separate channel, job (async-like) function

```
1  let ch = Ch<string> ()
2
3  let agent =
4    let rec loop count = job {
5      let! msg = ch // let! msg = Ch.take ch
6      do! timeOutMillis 100
7      return! loop (count+1)
8      }
9    start (loop 0)
```

- easier to post message from another job, that could run sync

```
1  let setup = job {
2    do! ch *<- m  // do! Ch.give ch m
3    }
4
5  run setup
```

Counter – C# Channels

- separate channel (w/ options)=

```
1 BoundedChannelOptions one2one =
2     new BoundedChannelOptions (1)
3         {SingleWriter=true, SingleReader=true,};
4
5 Channel<string> ch =
6     Channel.CreateBounded<string> (one2one);
```

- separate async function (not started)

```
1 async Task agent () {
2   var count = 0;
3   for (;;) {
4     var msg = await ch.Reader.ReadAsync ();
5     count += 1;
6     await Task.Delay (100);
7   }
8 }
```

Counter – C# Channels

- easier to post message from another async task

```
1 async Task Main() {
2   var a = agent();   // create agent and start it async
3
4   await ch.Writer.WriteAsync(m);
5 }
```

Outline

Compile and Run – F# Mailbox

- Mailbox in FSharp.Core, so no special configuration required
- All F# code: Linqpad preamble ⇔ module line in .FS

```
1 <Query Kind="FSharpProgram" />
2
3 module M
```

- Command-line compilation

```
1 fsc F#-Actor-counters.fs
```

- For all F# programs: you may also want to create a proper Main function, that will invoke the rest

```
1 [<EntryPoint>]
2 let main args =
3     ...
4     0
```

Compile and Run – F# Hopac

- Hopac NOT in FSharp.Core, additional configuration required
- Command-line compilation – two libraries

```
fsc -r:Hopac.dll -r:Hopac.Core.dll
    F#-Hopac-counters.fs
```

- Runtime – one more library

```
Hopac.Platform.dll
```

- Runtime – F#-Hopac-counters.exe.config, to ensure server garbage collection

```
<configuration>
  <runtime>
    <gcServer enabled="true"/>
  </runtime>
</configuration>
```

Compile and Run – C# Channels

- Channels lib NOT in system, additional configuration required
- Command-line compilation – three libraries + netstandard

```
csc  -r:mscorlib.dll,netstandard.dll,
     System.Threading.Channels.dll,
     System.Threading.Tasks.Extensions.dll,
     System.Runtime.CompilerServices.Unsafe.dll
     C#-Channels-counters.cs
```

- Runtime – one more library

```
System.Collections.Immutable.dll
```

- Code – see and compare the samples

```
static void Main (string[] args) {
    new Program().Main2().Wait();
}
```

Outline

State – F# Mailbox, F# Hopac, C# Channels

```
1
2  // please see samples
```

Outline

Hopac Async Post

- Async communication via a 0-size channel?

```
1  job {
2    do! ch *<- m // do! Ch.give ch m     // sync post
3
4    do! ch *<+ m // do! Ch.send ch m     // async post
5  }
```

- Ch.give is a sync post: the writer awaits (logically suspended) until the rendezvous!
- Ch.send is an async post: the writer generates a hidden helper job that takes charge of the actual writing!
- this helper job awaits until the rendezvous! Cf. code sample!
- thus the original writer job can continue!

Hopac Async Post

- Async communication via a 0-size channel?

```
1  job {
2    do! ch *<- m // do! Ch.give ch m     // sync post
3
4    do! ch *<+ m // do! Ch.send ch m     // async post
5  }
```

- Ch.give is a sync post: the writer awaits (logically suspended) until the rendezvous!
- Ch.send is an async post: the writer generates a hidden helper job that takes charge of the actual writing!
- this helper job awaits until the rendezvous! Cf. code sample!
- thus the original writer job can continue!

Hopac Async Post

- Async communication via a 0-size channel?

```
1  job {
2    do! ch *<- m // do! Ch.give ch m     // sync post
3
4    do! ch *<+ m // do! Ch.send ch m     // async post
5  }
```

- Ch.give is a sync post: the writer awaits (logically suspended) until the rendezvous!
- Ch.send is an async post: the writer generates a hidden helper job that takes charge of the actual writing!
- this helper job awaits until the rendezvous! Cf. code sample!
- thus the original writer job can continue!

Hopac Async Post

- Simulating async send on sync give

```
let ( *<++ ) ch m =  // ch *<+ m
    job {
        let help = job {
            do! ch *<- m
        }
        start help
    }
```

- Usage – awaiting for help start (not completion)

```
  do! ch *<++ m  // do! ch *<+ m
```

Async Post Samples – out of the box features

	size	sync write	async write	read
F# Hopac Chan	$= 0$	$*<-$ .give	$*<+$ .send	.take
" BoundedMb	≥ 0	.put		.take
" Mailbox	∞		.send	.take
F# Actors	∞		.Post	.Receive
C# Channels	≥ 1, ∞		.WriteAsync	.ReadAsync
GO Channels	≥ 0	$<-$		$<-$
JS Channels	≥ 0		.push	.shift

More later: backpressure, select, ...