Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages

Alexander Dean

Rochester Institute of Technology
Golisano College of Computer and Information Sciences

August 12, 2014

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages

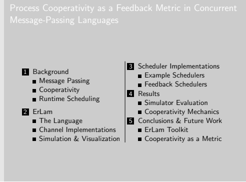
Alexander Dean

Rochester Institute of Technology
Golisano College of Computer and Information Sciences

August 12, 2014

- Thank Fluet, Heliotis, and Raj.
- Dedicate to parents, who are unable to be present.

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages


Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages

1 Background
- Message Passing
- Cooperativity
- Runtime Scheduling

2 ErLam
- The Language
- Channel Implementations
- Simulation & Visualization

3 Scheduler Implementations
- Example Schedulers
- Feedback Schedulers

4 Results
- Simulator Evaluation
- Cooperativity Mechanics

5 Conclusions & Future Work
- ErLam Toolkit
- Cooperativity as a Metric

- Break apart title = Background.
- Then simulator and test primitives (process behaviours).
- Schedulers and how comparisons are made.
- Results, Conclusions (of both simulator and cooperativity).

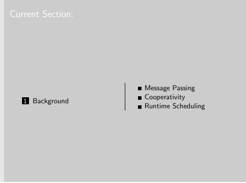
Current Section:

2014-08-07

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages

└─Background

└─Current Section:

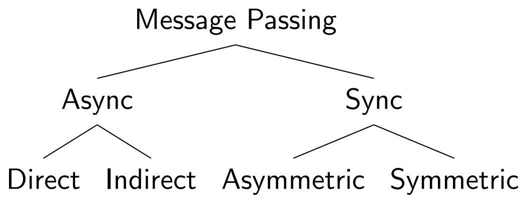

Current Section:

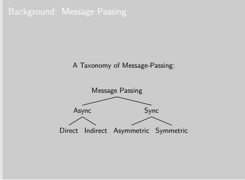
1 Background

- Message Passing
- Cooperativity
- Runtime Scheduling

Background: Message Passing

A Taxonomy of Message-Passing:




Background: Message Passing
A Taxonomy of Message-Passing:

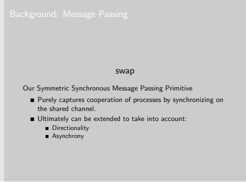
- Async: while user code doesn't block, there is blocking in terms of the channel implementation. This is not indicated (in most cases) to the scheduler.
- Direct = Mailboxes
- Indirect = Sockets
- Sync: The issue of process cooperation has been elevated to the process level for the scheduler to directly involve itself.

Background: Message Passing

swap

Our Symmetric Synchronous Message Passing Primitive

- Purely captures cooperation of processes by synchronizing on the shared channel.
- Ultimately can be extended to take into account:
  - Directionality
  - Asynchrony


Background: Message Passing

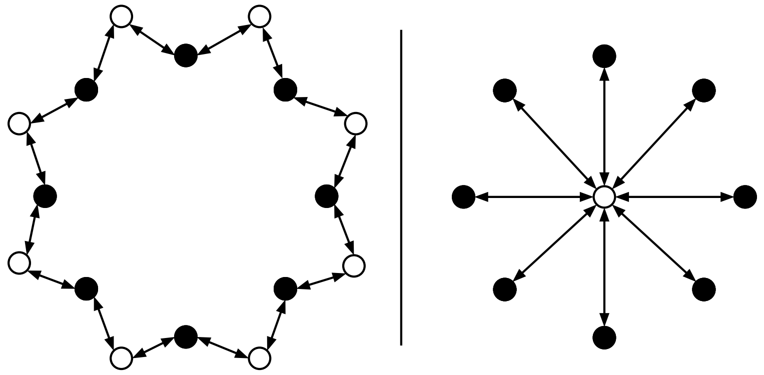
swap

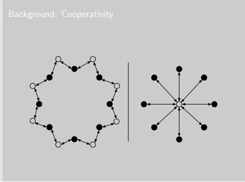
Our Symmetric Synchronous Message Passing Primitive

- Purely captures cooperation of processes by synchronizing on the shared channel.
- Ultimately can be extended to take into account:
  - Directionality
  - Asynchrony

Ultimately there is nothing stopping us from choosing the other types of message passing, but it would conflate the issue of process cooperativity if all we are after is whether two processes are cooperating on some task.

Background: Cooperativity





Whats the difference in cooperativity in the left and right set of processes? Where white represents a channel and black represents a process.

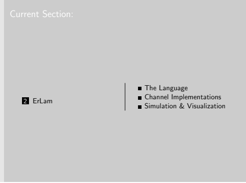
- Left: A Ring, the level of parallelism is nearly nil. Each process is cooperating yes, but the granularity of the application is very fine.
- Right: A Star, the level of parallelism is nearly full. Each process is cooperating, and is not reliant on more than one other.

Overall, what can be gained by looking at cooperativity in terms of understanding the applications behaviour? Knowing the granularity of parallelism.

Background: Runtime Scheduling

- How much of basic runtime scheduling do I need to talk about here? I may want to save it until Schedulers section and just discuss it all there.

Current Section:



Current Section:

2 ErLam

- The Language
- Channel Implementations
- Simulation & Visualization

ErLam: The Language

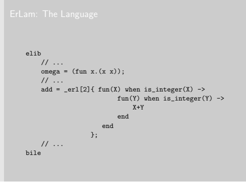
```
<Expression> ::= <Variable>
              |  <Integer>
              |  ‘newchan’
              |  ‘(’ <Expression> ‘)’
              |  <Expression> <Expression>
              |  ‘if’ <Expression> <Expression> <Expression>
              |  ‘swap’ <Expression> <Expression>
              |  ‘spawn’ <Expression>
              |  ‘fun’ <Variable> ‘.’ <Expression>
```

ErLam: The Language
<Expression> ::= <Variable> | <Integer> | ‘newchan’ | ‘(’ <Expression> ‘)’ | <Expression> <Expression> | ‘if’ <Expression> <Expression> <Expression> | ‘swap’ <Expression> <Expression> | ‘spawn’ <Expression> | ‘fun’ <Variable> ‘.’ <Expression>

- Extremely simple on purpose (5 keywords).
- Issue now began to be how to build up test primitives
- Made a library which allowed for built ins.

ErLam: The Language

```
elib
    // ...
    omega = (fun x.(x x));
    // ...
    add = _erl[2]{ fun(X) when is_integer(X) ->
                        fun(Y) when is_integer(Y) ->
                            X+Y
                        end
                    end
                 };
    // ...
bile
```


ErLam: The Language

```
elib
    // ...
    omega = (fun x.(x x));
    // ...
    add = _erl[2]{ fun(X) when is_integer(X) ->
                        fun(Y) when is_integer(Y) ->
                            X+Y
                        end
                    end
                 };
    // ...
bile
```

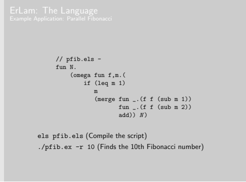
- Theres options for built-ins as well as macros.

ErLam: The Language
Example Application: Parallel Fibonacci

```
// pfib.els -
fun N.
    (omega fun f,m.(
        if (leq m 1)
           m
           (merge fun _.(f f (sub m 1))
                  fun _.(f f (sub m 2))
                  add)) N)
```

`els pfib.els` (Compile the script)

`./pfib.ex -r 10` (Finds the 10th Fibonacci number)

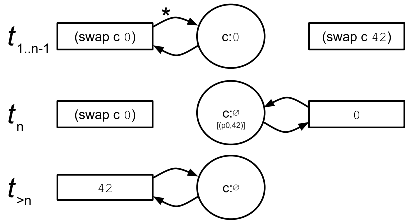

ErLam: The Language
Example Application: Parallel Fibonacci

```
// pfib.els -
fun N.
    (omega fun f,m.(
        if (leq m 1)
           m
           (merge fun _.(f f (sub m 1))
                  fun _.(f f (sub m 2))
                  add)) N)
```

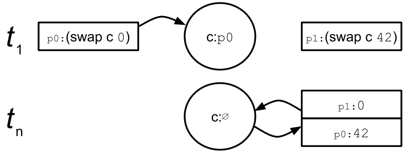
els pfib.els (Compile the script)
./pfib.ex -r 10 (Finds the 10th Fibonacci number)

- R option is to run program applied to 10.
- To add more to this presentation than just reading paper I want to also give more detail as to system usage.
- Explain this Common Usage Pattern.

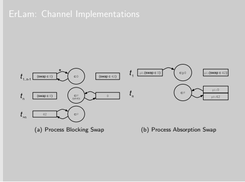
ErLam: Channel Implementations



(a) Process Blocking Swap



(b) Process Absorption Swap



- Blocking: Maintains state of current and previous swap value until swap is completed.
- Absorption: Stores process which initializes the swap and returns it along with completion.
- Mention expected effects on scheduler.
- Blocking is default, passing '-a' to els

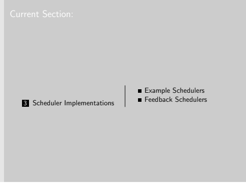
ErLam: Simulation & Visualization

2014-08-07

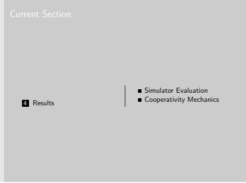
Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages
└─ErLam
  └─Simulation & Visualization
    └─ErLam: Simulation & Visualization

- Need to talk about Chart types and chart generation.

Current Section:



Current Section:

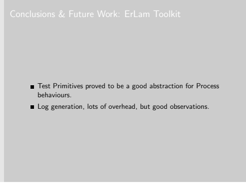


Current Section:

Current Section:
5 Conclusions & Future Work
ErLam Toolkit
Cooperativity as a Metric

Conclusions & Future Work: ErLam Toolkit

- Test Primitives proved to be a good abstraction for Process behaviours.
- Log generation, lots of overhead, but good observations.

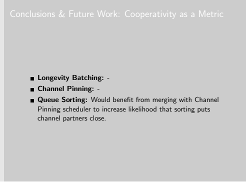

Conclusions & Future Work: ErLam Toolkit

- Test Primitives proved to be a good abstraction for Process behaviours.
- Log generation, lots of overhead, but good observations.

- Overall pleased with simulator and achieved its goal.
- Future Work:
  - Generate more test primitives.
  - Clean up log generation (reduce overhead).
  - Process evaluation uses alpha-reduction (can be sped up substantially).
  - Make schedulers more adjustable (different spawn/yields/etc).
  - More Channel implementations

Conclusions & Future Work: Cooperativity as a Metric

- **Longevity Batching:** -
- **Channel Pinning:** -
- **Queue Sorting:** Would benefit from merging with Channel Pinning scheduler to increase likelihood that sorting puts channel partners close.


Conclusions & Future Work: Cooperativity as a Metric

- **Longevity Batching:** -
- **Channel Pinning:** -
- **Queue Sorting:** Would benefit from merging with Channel Pinning scheduler to increase likelihood that sorting puts channel partners close.

- Alternate message-passing types (asymmetric).
- Merging Schedulers

Questions/Comments?

Questions/Comments?

Questions/Comments?

Thank You!

Questions/Comments?

Thank You!

Links:

- https://github.com/dstar4138/erlam
- https://github.com/dstar4138/thesis_cooperativity
- http://dstar4138.com

Links:

- https://github.com/dstar4138/erlam
- https://github.com/dstar4138/thesis_cooperativity
- http://dstar4138.com