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

2014-08-10

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

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

Alexander Dean

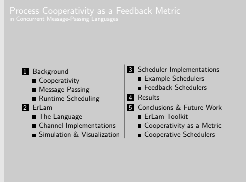
Rochester Institute of Technology
Golisano College of Computer and Information Sciences

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
- Cooperativity
- Message Passing
- Runtime Scheduling

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

3 Scheduler Implementations
- Example Schedulers
- Feedback Schedulers

4 Results

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

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

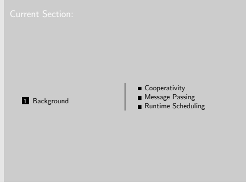
Current Section:

1 Background

- Cooperativity
- Message Passing
- Runtime Scheduling

2014-08-10

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages
└─Background
  └─Current Section:

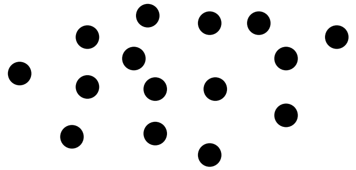

Current Section:

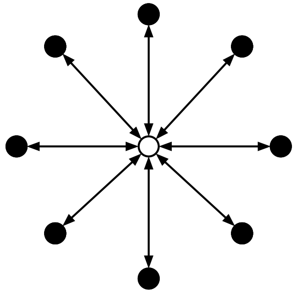
1 Background

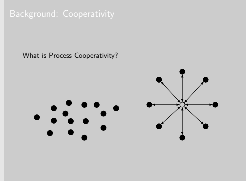
- Cooperativity
- Message Passing
- Runtime Scheduling

Background: Cooperativity

What is Process Cooperativity?





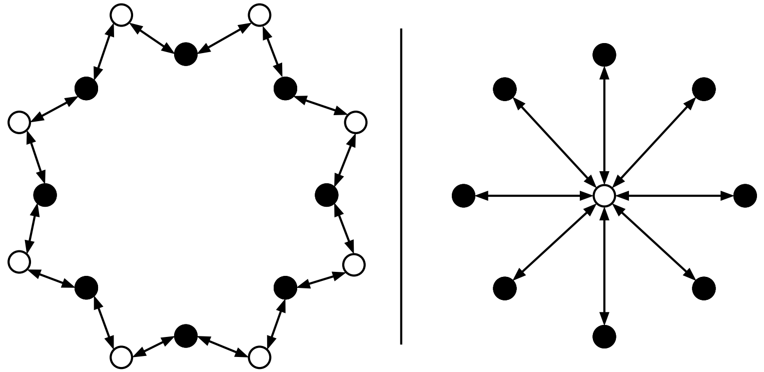


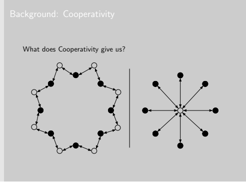
- Where white represents a channel and black represents a process.
- Channel = Source of synchronization (*i.e.* locks, abstractions, *etc.*)

- Left: Cloud of processes with no interaction.
- Right: Some sort of batch job? A bit of shared state amongst all processes these processes could all really be parallel or be competing, but in either case they cooperate.
- DEGREE OF COOPERATIVITY:
  - *of process:* flux of interaction with inter-proc comm method.
  - *of system:* rate of interaction between Ps and Cs as both flux in size.

Background: Cooperativity

What does Cooperativity give us?





Whats the difference in cooperativity in the left and right set of processes?

- 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.

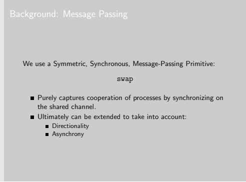
Next: what is a minimal inter-proc comm method?

Background: Message Passing

We use a Symmetric, Synchronous, Message-Passing Primitive:

`swap`

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



Background: Message Passing

We use a Symmetric, Synchronous, Message-Passing Primitive:

swap

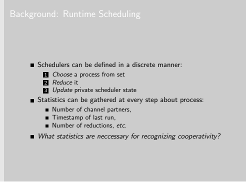
- Purely captures cooperation of processes by synchronizing on the shared channel.
- Ultimately can be extended to take into account:
  - Directionality
  - Asynchrony

- 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.
- Sync: The issue of process cooperation has been elevated to the process level for the scheduler to directly involve itself.
- 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.

Next: How would we use coop as a feedback metric?
What does that mean?

Background: Runtime Scheduling

- Schedulers can be defined in a discrete manner:
  1. *Choose* a process from set
  2. *Reduce* it
  3. *Update* private scheduler state
- Statistics can be gathered at every step about process:
  - Number of channel partners,
  - Timestamp of last run,
  - Number of reductions, *etc.*
- *What statistics are neccessary for recognizing cooperativity?*


Background: Runtime Scheduling

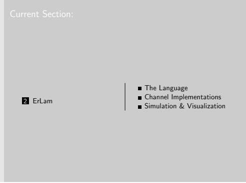
- Schedulers can be defined in a discrete manner:
  1. *Choose* a process from set
  2. *Reduce* it
  3. *Update* private scheduler state
- Statistics can be gathered at every step about process:
  - Number of channel partners,
  - Timestamp of last run,
  - Number of reductions, *etc.*
- *What statistics are necessary for recognizing cooperativity?*

- Choosing from set: Top Always (batching), Queue (Round-Robin)
- Reductions Return some indication: yield/blocked, unblocked, completed, nothing
- Update state based on historical data.
  - Timestamp of last run = Separate between batching/round-robin
  - Cooperativity metrics: yields, partners, thus longevity and granularity
- gets back to why we chose swap:
  - (if asymmetrical, we would have a harder time with 'partner')
  - (if async, we would have a harder time with 'yield' and noticing longevity/progress)

Current Section:

2014-08-10

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages
└ErLam
  └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>
```

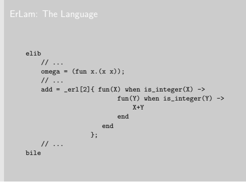
ErLam: The Language

```
<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
```



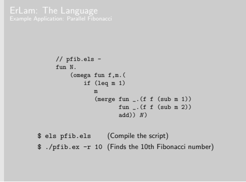
- There's options for built-ins as well as macros.
- Built-ins are raw Erlang, gets wrapped up into AST, and still "reduces" the same (*i.e.* no multi-variable functions).

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)
```

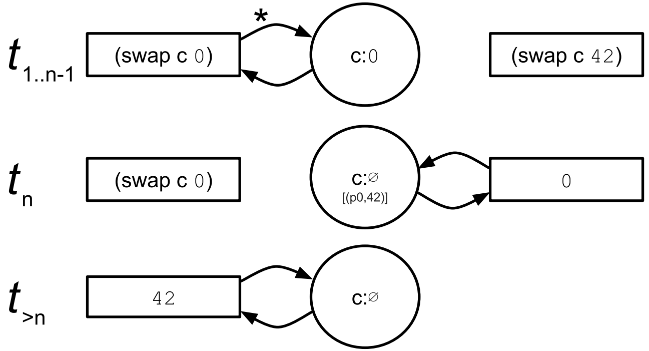

ErLam: The Language
Example Application: Parallel Fibonacci

```
// 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)
```

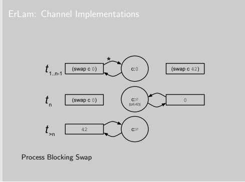
```
$ els pfib.els      (Compile the script)
$ ./pfib.ex -r 10   (Finds the 10th Fibonacci number)
```

- 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

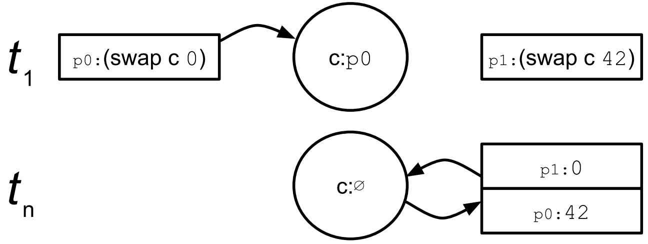


Process Blocking Swap

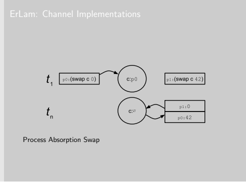


- Blocking: Maintains state of current and previous swap value until swap is completed.
- Mention expected effects on scheduler.
  - Scheduler will keep hold of the process, needs to recheck if blocked.
  - If all communicating, large process queue of blocked processes.
- Blocking is default, passing '-a' to els for absorb

ErLam: Channel Implementations



Process Absorption Swap



Absorption swap is visually easier to recognize.

1. Whole process gets absorbed by channel, away from scheduler.
2. When the second process completes the swap, the process gets removed from channel.
   - The p0 process can go back to its original scheduler,
   - OR to scheduler which unblocked it.

- Effects on scheduler are obvious.

ErLam: Simulation & Visualization

- System Behaviours:
  - Degree of Parallelism
  - Consistency of Cooperation
  - Degree of Longevity/Interactivity
  - Partial System Cooperativity
- Logging & Report Generation

ErLam: Simulation & Visualization
- System Behaviours:
  - Degree of Parallelism
  - Consistency of Cooperation
  - Degree of Longevity/Interactivity
  - Partial System Cooperativity
- Logging & Report Generation

- We first want to look at four types of system behaviour ranges:
  - Parallelism: Gets back to Ring vs Cloud, Compare the two.
  - Consistency: Ring/Star=consistent, but if given a random choice.
  - Longevity: Ratio of Communicating/Computing processes.
    - Longevity = Time spent reducing (low=communicator)
    - Interactivity, also takes into account user interaction.
  - Full vs Partial Cooperation: Multiple groups of Stars or Rings.
- Definitely won't get to all behaviour tests that were in report, but will focus on the key ones for each scheduler mechanic.
- Finally, quickly, go over the current report generation that ErLam can perform.

ErLam: Simulation & Visualization

System Behaviours: Degree of Parallelism

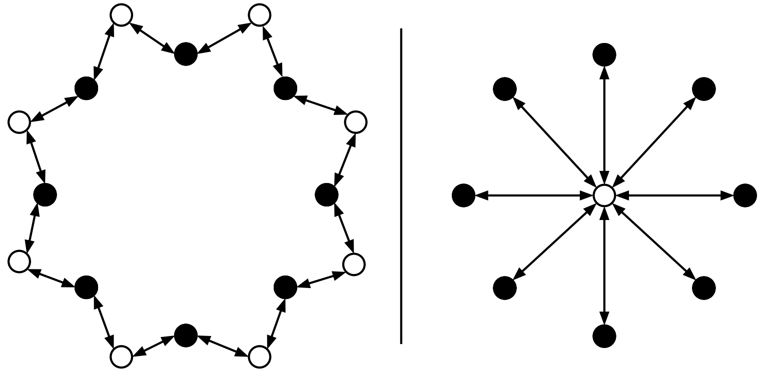
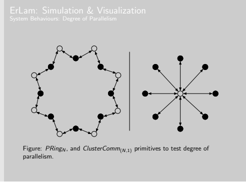


Figure: *PRing*$_N$, and *ClusterComm*$_{(N,1)}$ primitives to test degree of parallelism.



- Brings back the Ring and Star.
- We call the left, PRing, with the parameter N = number of processes.
- We call the right, *ClusterComm* with two parameters, N like *PRing*, $M = 1$ in this cas

ErLam: Simulation & Visualization

System Behaviours: Consistency of Cooperation

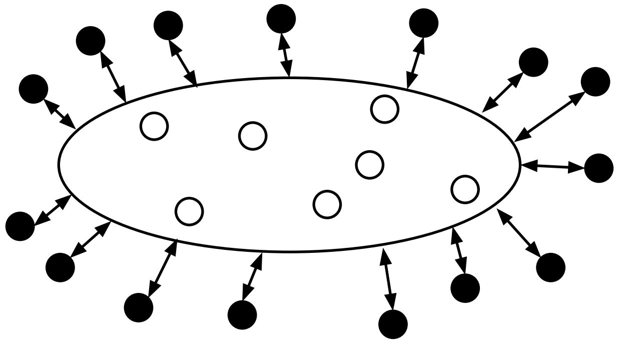
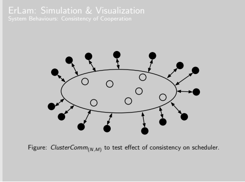


Figure: *ClusterComm*$_{(N,M)}$ to test effect of consistency on scheduler.



- We can vary number of channels in relation to processes to check the effects of inconsistency on Cooperative-Conscious schedulers.
- Worst case scenario for C-C schedulers.

ErLam: Simulation & Visualization
System Behaviours: Degree of Interactivity

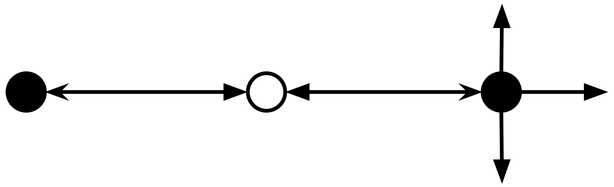
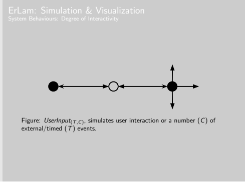


Figure: *UserInput*$_{(T,C)}$, simulates user interaction or a number (C) of external/timed (T) events.



- To test longevity we can just vary the length of time the processes chug for all tests.
- To test interactivity though, we need a way to simulate user interaction.
- *UserInput* captures hanging for a single event. We can compose these: $< NEXT >$
- With our cloud of processes (also called chugmachine) for simulating a program with consistent working processes and processes which are interactive.

ErLam: Simulation & Visualization
System Behaviours: Degree of Interactivity

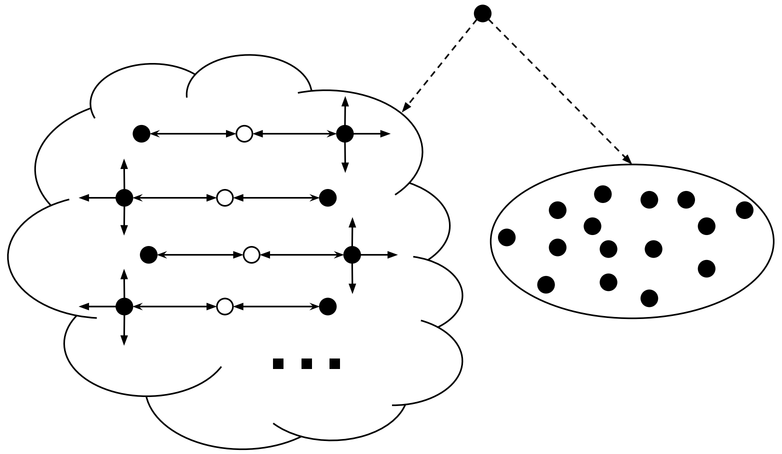
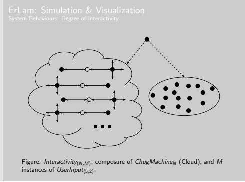


Figure: *Interactivity*$_{(N,M)}$, composure of *ChugMachine*$_N$ (Cloud), and M instances of *UserInput*$_{(5,2)}$.



- To test longevity we can just vary the length of time the processes chug for all tests.
- To test interactivity though, we need a way to simulate user interaction.
- *UserInput* captures hanging for a single event. We can compose these: $< NEXT >$
- With our cloud of processes (also called chugmachine) for simulating a program with consistent working processes and processes which are interactive.

ErLam: Simulation & Visualization
System Behaviours: Partial System Cooperativity

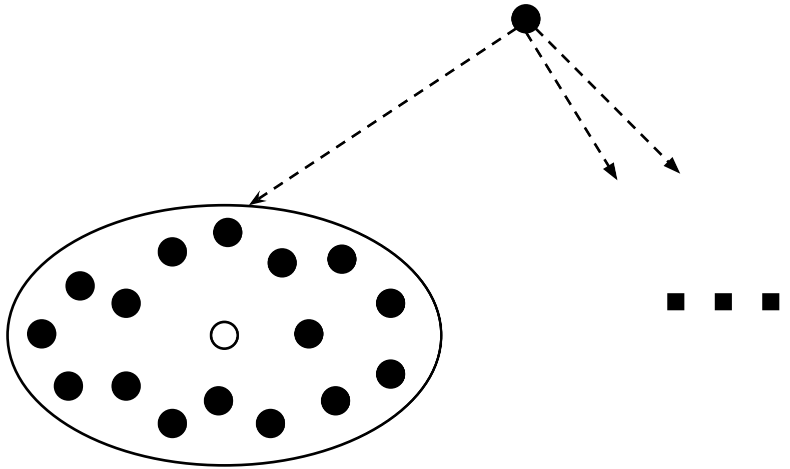
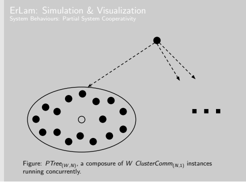


Figure: $PTree_{(W,N)}$, a composure of W $ClusterComm_{(N,1)}$ instances running concurrently.

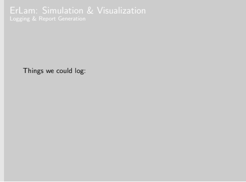


- Previously (besides Interactivity), all systems were full system cooperation.
- We can of course use Interactivity for our partial system cooperativity tests, but we would instead like to see logical grouping.
- Hence the set of Work groups (or stars).

ErLam: Simulation & Visualization

Logging & Report Generation

Things we could log:

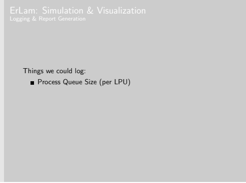


ErLam: Simulation & Visualization

Logging & Report Generation

Things we could log:

- Process Queue Size (per LPU)


ErLam: Simulation & Visualization
Logging & Report Generation

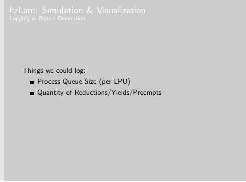
Things we could log:
- Process Queue Size (per LPU)

- Queue-Length: work-stealing mechanics and saturation ability.

ErLam: Simulation & Visualization
Logging & Report Generation

Things we could log:

- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts


ErLam: Simulation & Visualization
Logging & Report Generation
Things we could log:
- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts

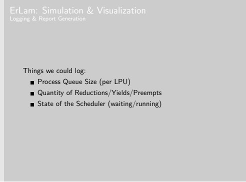
- Queue-Length: work-stealing mechanics and saturation ability.
- Tick-Action: Visualize the density of computation/communication.

ErLam: Simulation & Visualization

Logging & Report Generation

Things we could log:

- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)


ErLam: Simulation & Visualization
Logging & Report Generation

Things we could log:

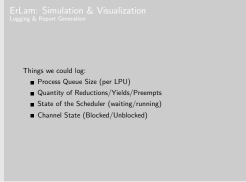
- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)

- Queue-Length: work-stealing mechanics and saturation ability.
- Tick-Action: Visualize the density of computation/communication.
- Sched-State: Useful for comparing stealing/process selection mechanics.

ErLam: Simulation & Visualization
Logging & Report Generation

Things we could log:

- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)
- Channel State (Blocked/Unblocked)

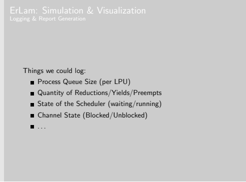

ErLam: Simulation & Visualization
Logging & Report Generation
Things we could log:
- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)
- Channel State (Blocked/Unblocked)

- Queue-Length: work-stealing mechanics and saturation ability.
- Tick-Action: Visualize the density of computation/communication.
- Sched-State: Useful for comparing stealing/process selection mechanics.
- Chan-State: Tracking interactivity, speed of unblock=attentive to cooperation.

ErLam: Simulation & Visualization
Logging & Report Generation

Things we could log:

- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)
- Channel State (Blocked/Unblocked)
- . . .

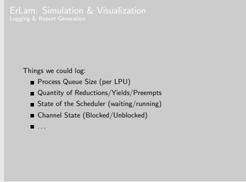

ErLam: Simulation & Visualization
Logging & Report Generation
Things we could log:
- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)
- Channel State (Blocked/Unblocked)
- . . .

- Queue-Length: work-stealing mechanics and saturation ability.
- Tick-Action: Visualize the density of computation/communication.
- Sched-State: Useful for comparing stealing/process selection mechanics.
- Chan-State: Tracking interactivity, speed of unblock=attentive to cooperation.
- Of course there are more, but we limited ourselves to the above for initial testing purposes.

ErLam: Simulation & Visualization
Logging & Report Generation

Things we could log:

- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)
- Channel State (Blocked/Unblocked)
- . . .


ErLam: Simulation & Visualization
Logging & Report Generation

Things we could log:
- Process Queue Size (per LPU)
- Quantity of Reductions/Yields/Preempts
- State of the Scheduler (waiting/running)
- Channel State (Blocked/Unblocked)
- . . .

- Queue-Length: work-stealing mechanics and saturation ability.
- Tick-Action: Visualize the density of computation/communication.
- Sched-State: Useful for comparing stealing/process selection mechanics.
- Chan-State: Tracking interactivity, speed of unblock=attentive to cooperation.
- Of course there are more, but we limited ourselves to the above for initial testing purposes.

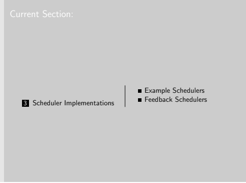
Current Section:

2014-08-10

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

└─Scheduler Implementations

└─Current Section:



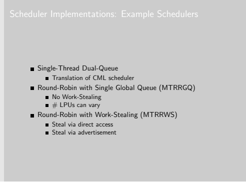
Current Section:

3 Scheduler Implementations

- Example Schedulers
- Feedback Schedulers

Scheduler Implementations: Example Schedulers

- Single-Thread Dual-Queue
  - Translation of CML scheduler
- Round-Robin with Single Global Queue (MTRRGQ)
  - No Work-Stealing
  - # LPUs can vary
- Round-Robin with Work-Stealing (MTRRWS)
  - Steal via direct access
  - Steal via advertisement

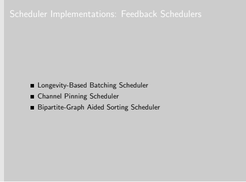

Scheduler Implementations: Example Schedulers

- Single-Thread Dual-Queue
  - Translation of CML scheduler
- Round-Robin with Single Global Queue (MTRRGQ)
  - No Work-Stealing
  - # LPUs can vary
- Round-Robin with Work-Stealing (MTRRWS)
  - Steal via direct access
  - Steal via advertisement

- First tested translating a known scheduler to the discrete step scheduler semantics.
- MTRRGQ: all synchronization around a single global queue, test P=1 or max.
- MTRRWS-SQ:
  - Schedulers can access a random LPU's queue end and steal from their bottoms.
- MTRRWS-IS:
  - Uses a secondary queue to advertise desire to steal.
  - Scheduler can check secondary queue as desired.
  - No requirement for synchronization on private process queue.

Scheduler Implementations: Feedback Schedulers

- Longevity-Based Batching Scheduler
- Channel Pinning Scheduler
- Bipartite-Graph Aided Sorting Scheduler


Scheduler Implementations: Feedback Schedulers

- Longevity-Based Batching Scheduler
- Channel Pinning Scheduler
- Bipartite-Graph Aided Sorting Scheduler

Scheduler Implementations: Feedback Schedulers

Longevity-Based Batching Scheduler

2014-08-10

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

└─Scheduler Implementations

  └─Feedback Schedulers

    └─Scheduler Implementations: Feedback Schedulers

Scheduler Implementations: Feedback Schedulers

Channel-Pinning Scheduler

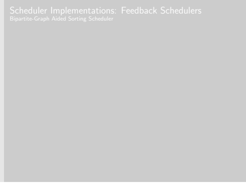
2014-08-10

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages
└─Scheduler Implementations
  └─Feedback Schedulers
    └─Scheduler Implementations: Feedback Schedulers

Scheduler Implementations: Feedback Schedulers
Channel-Pinning Scheduler

Scheduler Implementations: Feedback Schedulers

Bipartite-Graph Aided Sorting Scheduler



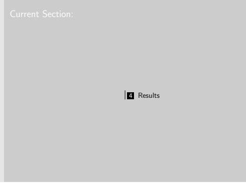
Current Section:

2014-08-10

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

└─Results

└─Current Section:



Current Section:

4 Results

Results:

Longevity-Based Batching Scheduler:

- Spawn Mechanism

  (Where does a new process go if batch is too big?)

Channel Pinning Scheduler:

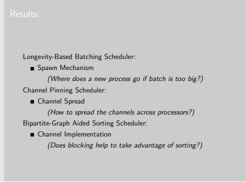
- Channel Spread

  (How to spread the channels across processors?)

Bipartite-Graph Aided Sorting Scheduler:

- Channel Implementation

  (Does blocking help to take advantage of sorting?)


Results:

Longevity-Based Batching Scheduler:
- Spawn Mechanism
  (Where does a new process go if batch is too big?)

Channel Pinning Scheduler:
- Channel Spread
  (How to spread the channels across processors?)

Bipartite-Graph Aided Sorting Scheduler:
- Channel Implementation
  (Does blocking help to take advantage of sorting?)

- LBB: Can also look at how the batch size effects different types of behaviours.
- SS: Could also look at a stealing mechanism.

Results:

Longevity-Based Batching Scheduler

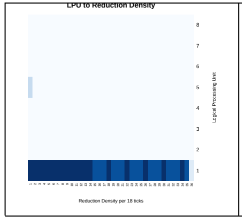
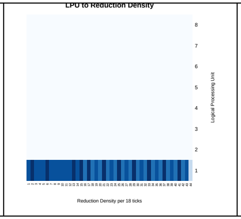
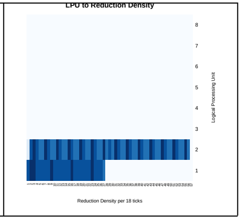
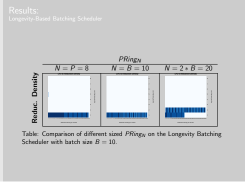
	$PRing_N$		
	$N = P = 8$	$N = B = 10$	$N = 2 * B = 20$
Reduc. Density			

Table: Comparison of different sized $PRing_N$ on the Longevity Batching Scheduler with batch size $B = 10$.

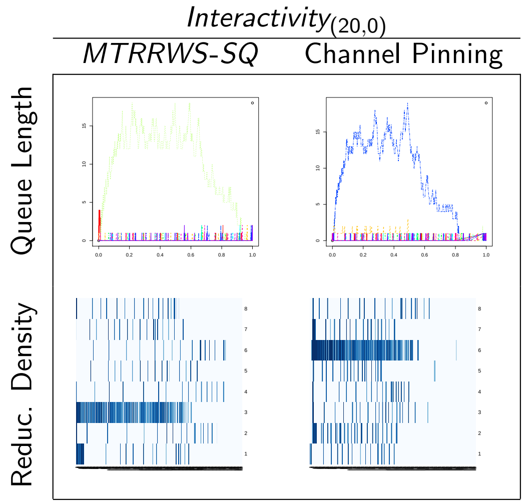

Results:
Longevity-Based Batching Scheduler
Table: Comparison of different sized $PRing_N$ on the Longevity Batching Scheduler with batch size $B = 10$.

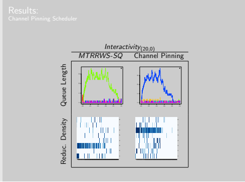
- Before talking about spawn mechanism, pointing out the primary goal and effect of batching to catch the granularity of the application.

Results:
Longevity-Based Batching Scheduler

- TODO: Get simple $\mathit{ClusterComm}_{(N,1)}$ or the PTree results to contrast with PRING
- This brings it back to an issue of behaviour recognition.

Results:
Channel Pinning Scheduler

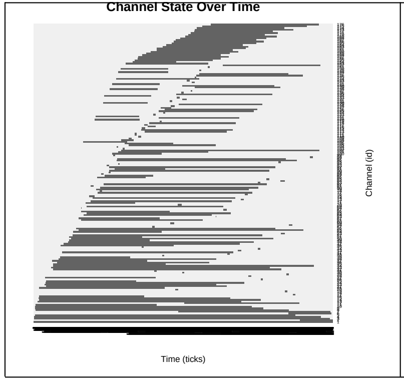
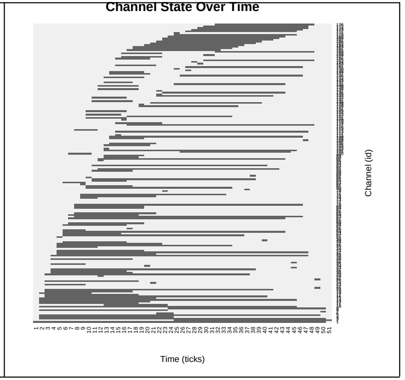


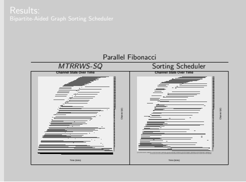


- Comparison of Uniform synchronization for *MTRRWS-SQ* and the Channel Pinning Scheduler on Absorption Channels.
- This used the *even* spread type.
- Note the speed at which it saturates all cores.
- Despite Naive WS, we still have decent spread.

Results:
Bipartite-Aided Graph Sorting Scheduler

Parallel Fibonacci

MTRRWS-SQ	Sorting Scheduler
	

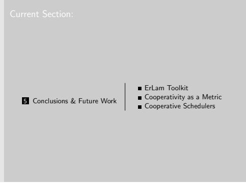


- Channel State comparison of Parallel Fibonacci executed on *MTRRWS-SQ* and the Bipartite-Graph Aided Sorting Scheduler.
- Note the large reduction in number of ticks.

Current Section:

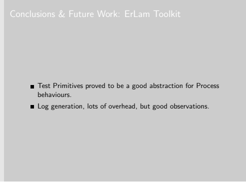
2014-08-10

Process Cooperativity as a Feedback Metric in Concurrent Message-Passing Languages
└─Conclusions & Future Work
  └─Current Section:



Conclusions & Future Work: ErLam Toolkit

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

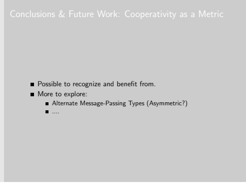

Conclusions & Future Work: ErLam Toolkit

- 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

- Possible to recognize and benefit from.
- More to explore:
  - Alternate Message-Passing Types (Asymmetric?)
  - ....

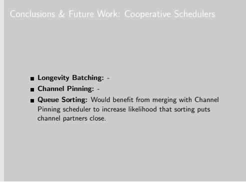


Conclusions & Future Work: Cooperativity as a Metric

- Possible to recognize and benefit from.
- More to explore:
  - Alternate Message-Passing Types (Asymmetric?)
  - ....

Conclusions & Future Work: Cooperative Schedulers

- **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: Cooperative Schedulers

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

- Merging Schedulers

Questions/Comments?

Questions/Comments?

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