Parallel Design Patterns using Higher-order Actors

Chang-Seo Park
Christos Stergiou

Project Goals

- Multicore execution of Ptolemy models
  - Scalable to multiple cores
- Exploit task and data parallelism
- Extend existing static scheduling domains
  - SDF
  - (Others suggested)

Extracting Parallelism

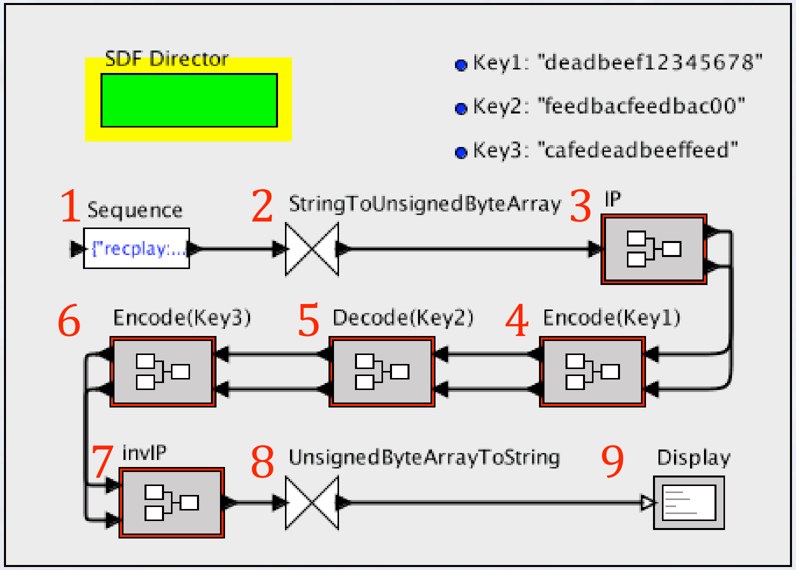
- Task & Pipeline parallelism
  - Give each actor a thread
  - What if more cores than actors?
  - What if too many actors?
- Data parallelism
  - Run same schedule on different data independently

Assumptions

- Assume actors have no state
  - Can’t use Expression, FIR
  - Loops are also problematic
- Computation bound application

Synchronous Dataflow

- Each actor consumes and produces fixed amount of token on each firing (usually 1)
- Firing sequence of actors can be determined statically

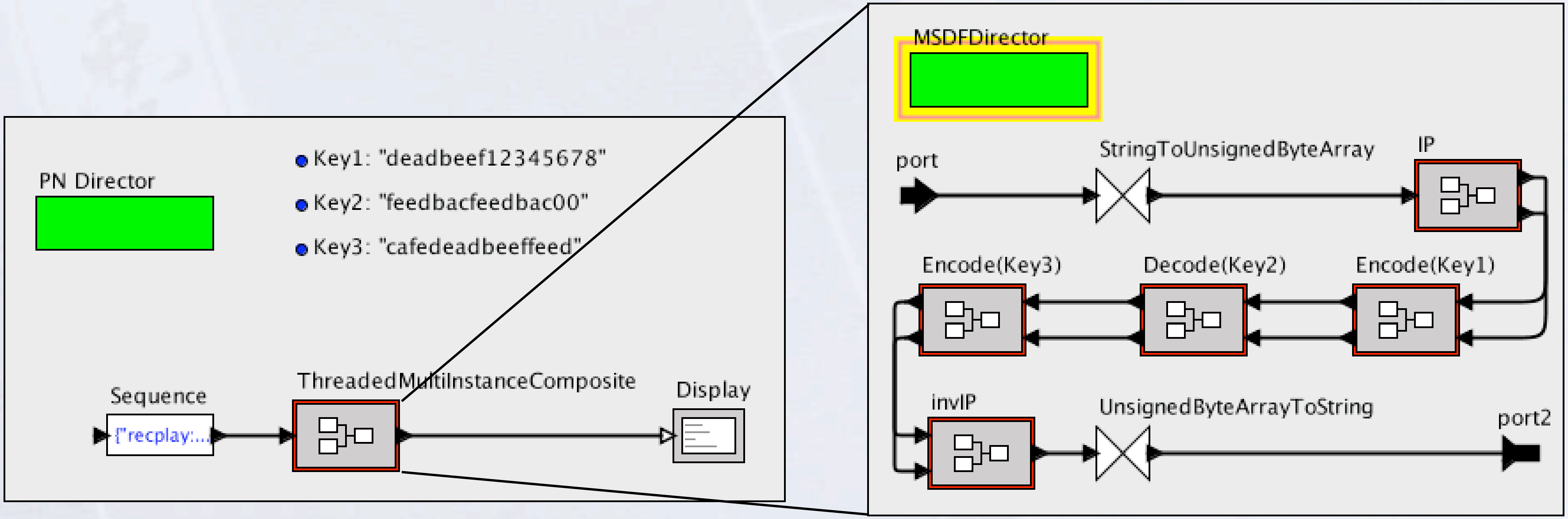


Schedule
Sequence(1), S2UBA(1), IP(1), E_Key1(1), D_Key2(1), E_Key3(1), invIP(1), UBA2S(1), Display(1)

3DES Encryption

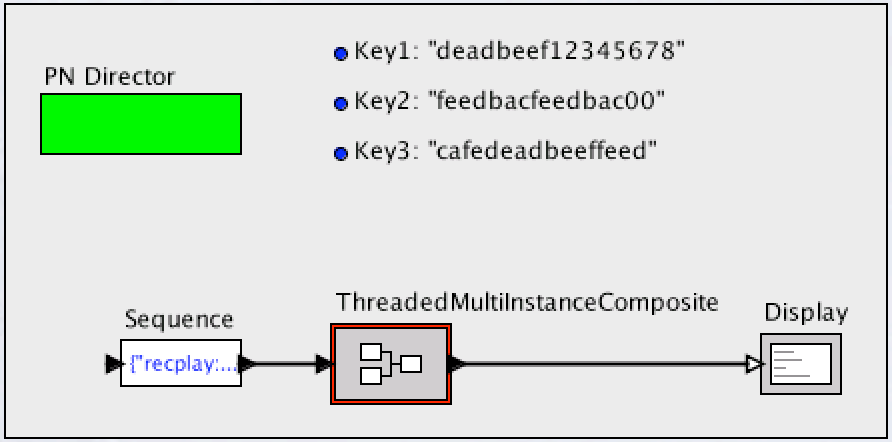
Multicore Synchronous Dataflow

- Programmer encapsulates parallelizable region in a composite actor
  - Run schedule on multiple cores



Multicore Synchronous Dataflow

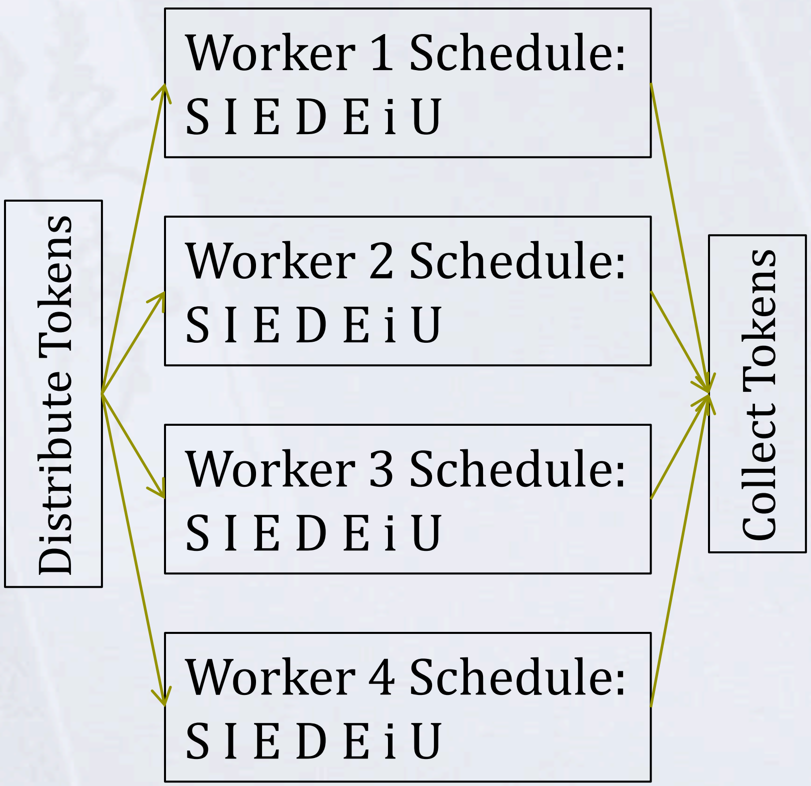
- Assume we have 4 cores

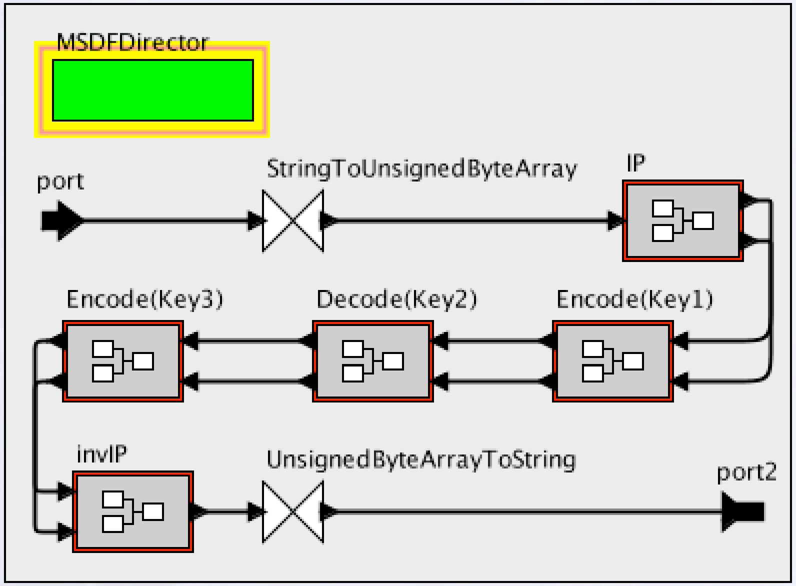


Schedule
S S S S T D D D D

Multicore Synchronous Dataflow

- Assume we have 4 cores





Parallel Fork-Join Actor

- ThreadedMultiInstanceComposite Actor
  - Given n worker threads, runs static schedule of component actors on each worker
  - Vectorization factor runs multiple schedules on each worker for less overhead
- Current Status
  - Deterministic fork-join order
  - Receiver multiplexing instead of actor cloning
  - Linear scaling for computation intensive toy benchmark

Implementation

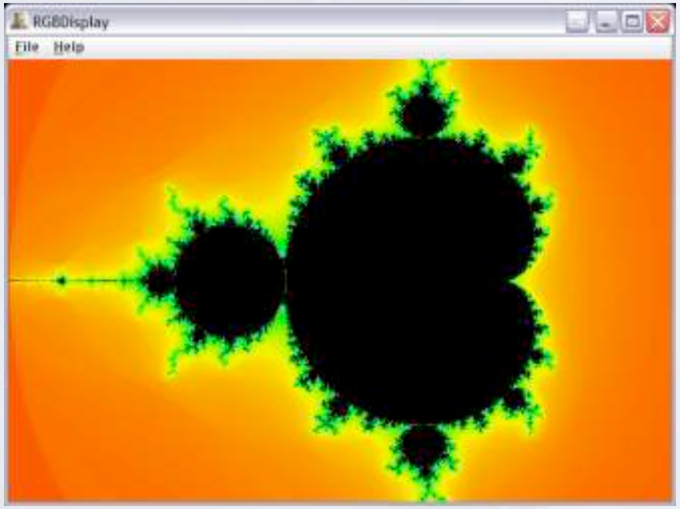
- MSDFDirector
  - Prefire inflates consumption rate
  - Fire
  - Returns msdf receivers
- MSDFReceiver
  - Get & put
  - GetWorkerReceiver : mapping from thread to receiver index

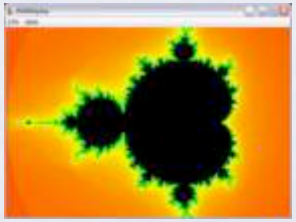
Application: Mandelbrot Set

- Compute whether for a complex number z_0,

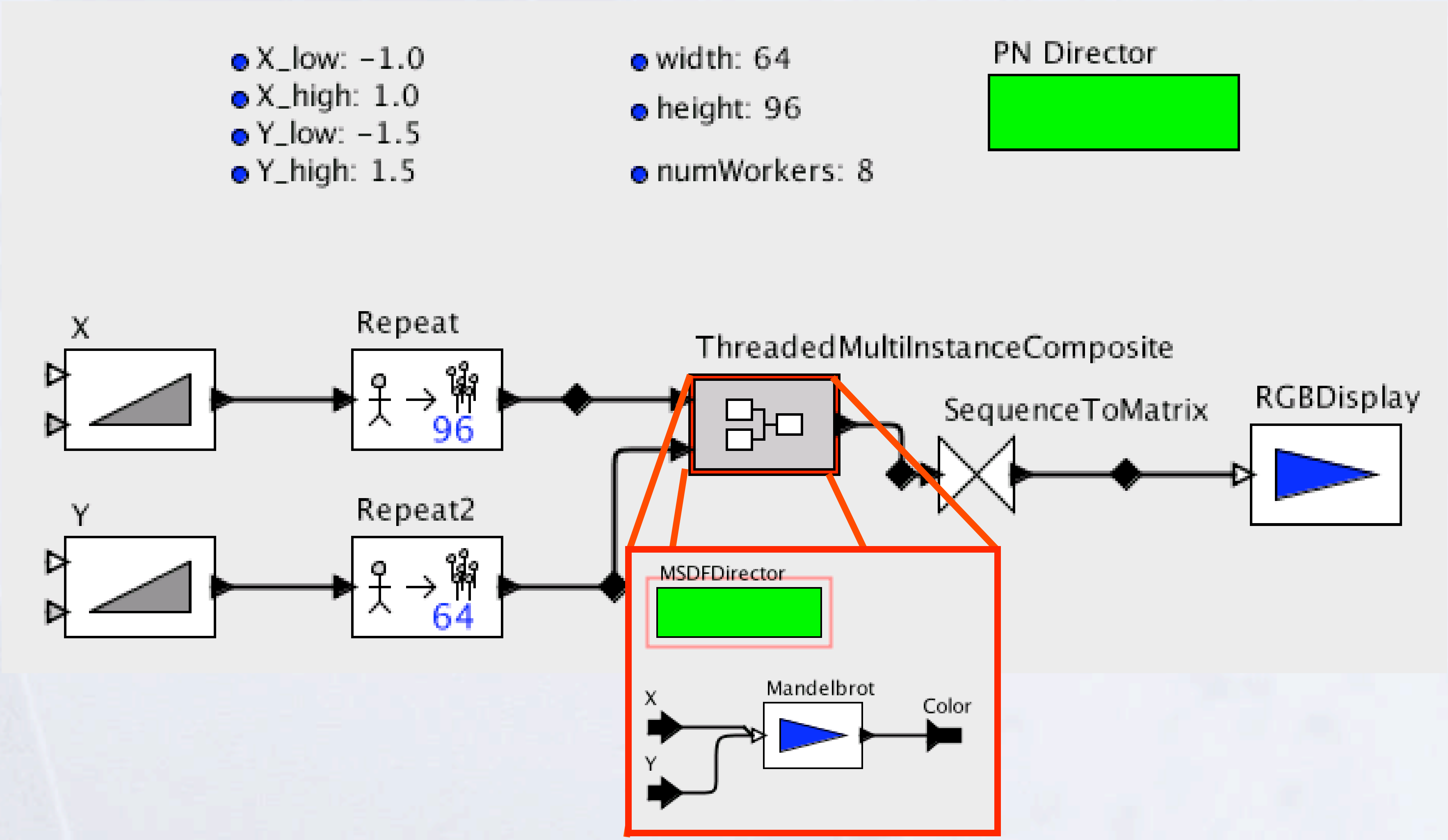
  $$z_n = z_{n-1}^p + z_0$$

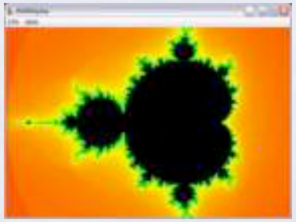
  converges or not
  - Compute intensive
  - Embarrassingly parallel for each number





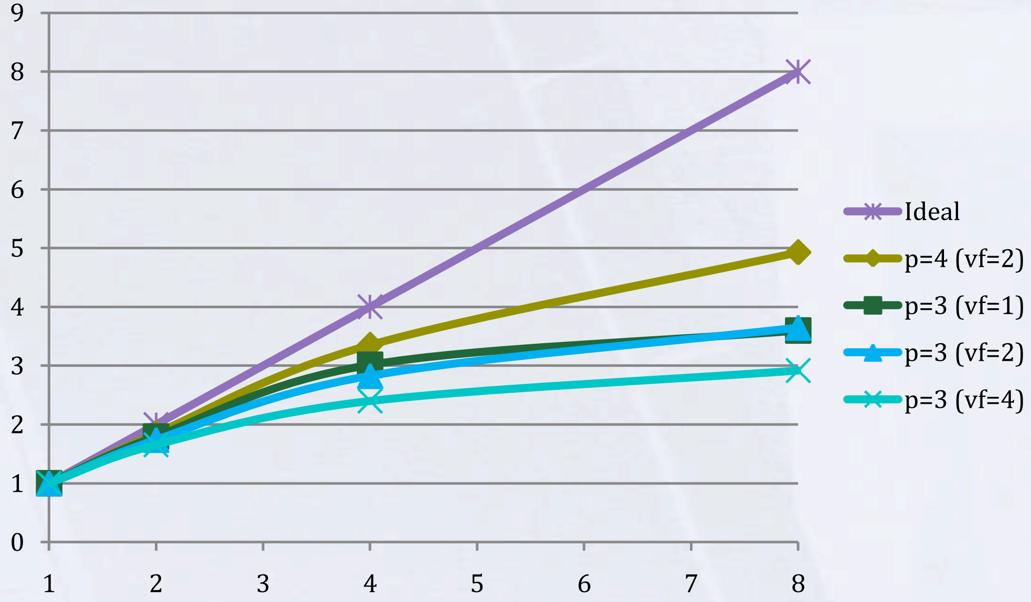
Application: Mandelbrot Set





Application: Mandelbrot Set

- Speedup



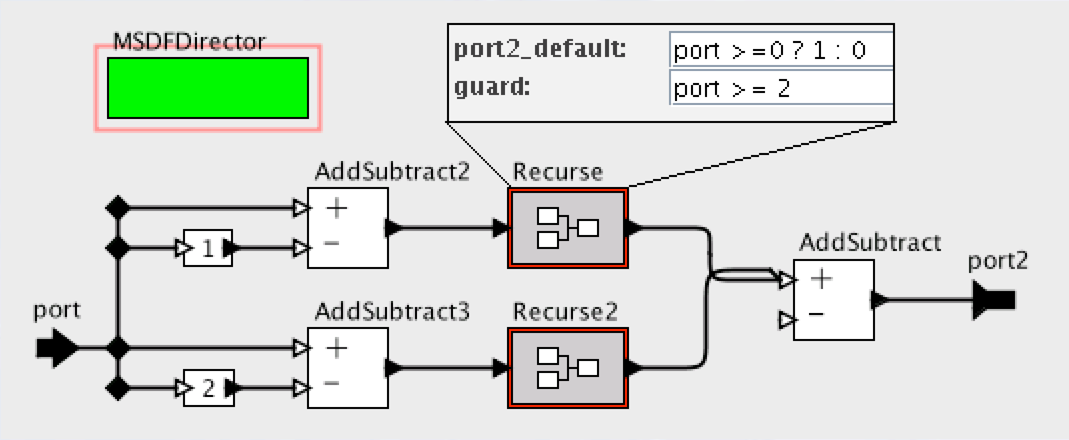
- Verdict: near linear scaling to 4 cores, can extend to more cores with larger problem

Recursion Actor

- Remains statically schedulable as long as base case and recursive case consume and produce same number of tokens
  - Have a “guard” input that decides whether to recurse
  - “default” model for base case
- Nested cloning of actors avoided by using receiver multiplexing

Application: Fibonacci

- Compute the n-th Fibonacci number
- Naïve algorithm runs in $O(2^n)$



Fibonacci

Application: Fibonacci number

- Results

n	Recurse fib$_1(n)$	Recurse fib$_2(n)$	ActorRecursion fib$_1(n)$	ActorRecursion fib$_2(n)$
10	32	22	909 (12,922)	62 (542)
20	2627	26	- (>10min)	101 (1,065)
40	>3min	29	- (-)	217 (2,633)

- Verdict: More efficient execution than actor cloning

Future Work

- Schedules do not have to be a linear order
  - Partial order schedules allows for parallelism
  - Task stealing among worker threads
- Dynamic load balancing in the presence of multiple parallelizable regions
  - Input queue length is a good indicator of "utilization" – give and take workers as necessary
- More SDF Actors to simplify programming
  - Spawn, Iterate, etc.

Conclusion

- Multicore scalability is possible
  - Nature of the problem
  - Platform overhead
- Multiplexing receivers is more efficient than explicit actor cloning
  - Allowed for a clean implementation of MSDF
  - Provided support for Recursion actor