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
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
Programmer encapsulates parallelizable region in a composite actor
• Run schedule on multiple cores
Assume we have 4 cores

Schedule
S S S S T D D D D
Multicore Synchronous Dataflow

➤ Assume we have 4 cores

Worker 1 Schedule: S I E D E i U
Worker 2 Schedule: S I E D E i U
Worker 3 Schedule: S I E D E i U
Worker 4 Schedule: S I E D E i U
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
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

- $X_{low} = -1.0$
- $X_{high} = 1.0$
- $Y_{low} = -1.5$
- $Y_{high} = 1.5$
- $width = 64$
- $height = 96$
- $numWorkers = 8$
Application: Mandelbrot Set

- Speedup

- Verdict: near linear scaling to 4 cores, can extend to more cores with larger problem
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
Compute the $n$-th Fibonacci number

Naïve algorithm runs in $O(2^n)$
Application: Fibonacci number

Results

<table>
<thead>
<tr>
<th>n</th>
<th>Recurse fib₁(n)</th>
<th>Recurse fib₂(n)</th>
<th>ActorRecursion fib₁(n)</th>
<th>ActorRecursion fib₂(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>32</td>
<td>22</td>
<td>909 (12,922)</td>
<td>62 (542)</td>
</tr>
<tr>
<td>20</td>
<td>2627</td>
<td>26</td>
<td>- (&gt;10min)</td>
<td>101 (1,065)</td>
</tr>
<tr>
<td>40</td>
<td>&gt;3min</td>
<td>29</td>
<td>- (-)</td>
<td>217 (2,633)</td>
</tr>
</tbody>
</table>

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.
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

Conclusion