FRACTAL

AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

SUVINAY SUBRAMANIAN, MARK C. JEFFREY, MALEEN ABAYDEERA, HYUN RYONG LEE, VICTOR A. YING, JOEL EMER, DANIEL SANCHEZ

ISCA 2017
Current speculative systems scale poorly

Speculative parallelization, e.g. TM, simplifies parallel programming

Performs poorly on real world applications...

...because applications comprise large atomic tasks
Large atomic tasks limit performance

Database Transaction

query X
...
...
update Z
...
...
query U
...
...
update V

Millions of cycles
Prone to aborts
Challenging to track
Serial (misses parallelism)
Large atomic tasks have abundant nested parallelism!

How to
- extract parallelism?
- maintain atomicity?
- achieve high performance?
Prior TMs fail to exploit nested parallelism

1. Merging of “nested” speculative state with parent
   Large speculative state, prone to aborts

2. Cyclic dependence between parent and nested children
   Deadlock and livelock issues

See the paper for more details!
Ordering tasks to guarantee atomicity

FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
Fractal decouples atomicity from parallelism

1. Decouples unit of atomicity from unit of parallelism
   - Domain: All tasks belonging to a domain appear to execute atomically

2. Implementation guarantees atomicity by ordering tasks
   - No merging speculative state

Benefits of Fractal

- Tiny tasks
- Easy to track
- Composable speculative parallelism
Fractal Execution Model

DECOUPLING ATOMICITY FROM PARALLELISM
Domains to group tasks into atomic units

Fractal programs consist of atomic tasks

Tasks may access arbitrary data

Tasks may create child tasks

Tasks belong to a hierarchy of nested domains
Semantics across domains

Each task:
- can create a single subdomain
- can enqueue child tasks to subdomain or current domain

(All tasks in domain + creator of domain)

↓

Appear to execute as single atomic unit
Semantics within a domain

Unordered
- Arbitrary order while respecting parent-child dependences

Timestamp-ordered
- Tasks appear to execute in increasing timestamp order
- Children appear to execute after parent
Fractal software API

Creating and enqueueing tasks

fractal::enqueue(function_pointer, timestamp, arguments...);

Creating sub-domains

fractal::create_subdomain(<domain_type>);

High-level programming interface, e.g.

forall(), callcc(), parallel_reduce()
Example: Database transactions in Fractal

**TXN 1**
- query X
- query Z
- update Z
- query U
- update V

**Root domain**
- T1
  - qry X
  - qry Z
  - upd Z
  - 1
  - 2
  - 3
  - 4
  - 5

**TXN 2**
- query A
- query B
- update C
- update Z
- update K

**TXN 2**
- upd Z
  - 1
  - 2
  - 3
  - 4
  - 5

**Root domain**
- T2
  - qry A
  - qry B
  - upd C
  - upd Z
  - upd K
Fractal Implementation

ATOMICITY THROUGH ORDERING
**Fractal Virtual Time (VT)**

*Fractal* assigns a *fractal virtual time (VT)* to each task. Captures the ordering of tasks across domains, within a domain.

Fractal VT = \[
\begin{array}{cccccc}
45 & 23 & 108 & \ldots & 9 \\
\end{array}
\]

Domain VT
Example: Database transactions in Fractal
Example: Database transactions in Fractal

Fractal VT captures all ordering information
Swarm [MICRO’15] : An efficient substrate for ordered speculation

Swarm executes tasks speculatively and out of order

Large hardware task queues
Scalable ordered commits
Scalable ordered speculation

Efficiently supports tiny speculative tasks
Fractal features

Fractal VT construction requires no centralized structures

Fractal VT assigns order dynamically

Hardware supports a few number of concurrent depths

- “Zooming” operations allow for unbounded nesting
- Spill tasks from shallower domains to memory
- Parallelism compounds quickly with depth

See the paper for more details!
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

TXN 1
query X
query Z
update Z
query U
update V

TXN 2
query A
query B
update C
update Z
update K
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

TXN 1
query X
query Z
update Z
query U
update V

TXN 2
query A
query B
update C
update Z
update K
Time

TXN 1

query X
query Z
update Z
query U
update V

TXN 2

query A
query B
update C
update Z
update K

FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
FRACHTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
TXN 1
- query X
- query Z
- update Z
- query U
- update V

TXN 2
- query A
- query B
- update C
- update Z
- update K

FRACTAL: AN EXECUTION MODEL FOR FINE-GRANNE STED SPECULATIVE PARALLELISM
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

Tracking, conflict detection at level of fine-grain tasks

TXN 1

- query X
- query Z
- update V

TXN 2

- query A
- query B
- update C
- update K
Tracking, conflict detection at level of fine-grain tasks
Tracking, conflict detection at level of fine-grain tasks

Selective aborts waste less work
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
TXN 1

query X
query Z
update Z
query U
update V

TXN 2

query A
query B
update C
update Z
update K
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

TXN 1
query X
query Z
update Z
query U
update V

TXN 2
query A
query B
update C
update Z
update K

Task-level tracking
Task-level CD
Selective aborts
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM

TXN 1
- query X
- query Z
- update Z
- query U
- update V

TXN 2
- query A
- query B
- update C
- update Z
- update K

Core 1
- T1
  - 1
  - 1
  - 1
  - 1
  - 1
- query U
- update V

Core 2
- query X
- query Z
- update Z
- update V

Core 3
- query A
- query B
- update C
- update K

Core 4
- Task-level tracking
- Task-level CD
- Selective aborts

Time

Abort task
Conflict
FRACTAL: AN EXECUTION MODEL FOR FINE-GRAIN NESTED SPECULATIVE PARALLELISM
Task-level tracking
Task-level CD
Selective aborts

TXN 1
query X
query Z
update Z
query U
update V

TXN 2
query A
query B
update C
update Z
update K

FRACTAL: AN EXECUTION MODEL FOR FINE-GRAN NESTED SPECULATIVE PARALLELISM
Commit parent before child completes

TXN 1
query X
query Z
update Z
query U
update V

TXN 2
query A
query B
update C
update Z
update K

FRACTAL: AN EXECUTION MODEL FOR FINE-GRAN NESTED SPECULATIVE PARALLELISM

Core 1

Core 2

Core 3

Core 4

Task-level tracking

Task-level CD

Selective aborts

Abort task

Conflict

Time
Fractal unlocks the benefits of fine-grain parallelism.

- Query X
- Query Z
- Update Z
- Query U
- Update V

Commit parent before child completes.

Task-level tracking
Task-level CD
Selective aborts
Evaluation
Methodology

Event-driven, Pin-based simulator

Target system: 256-core, 64-tile chip

Scalability experiments from 1–256 cores
○ Scaled-down systems have fewer tiles

Applications
○ Unordered (STAMP): labyrinth, bayes
○ Ordered: color, msf, silo, maxflow, mis

64 MB shared L3 (1MB/tile)
256 KB per-tile L2s
16 KB per-core L1s
In-order, single-issue, scoreboarded

16K task queue entries (64/core)
4K commit queue entries (16/core)
Fractal uncovers abundant nested parallelism

- Flat
- Fractal

Large atomic tasks

Nested parallelism exposed through fine-grained tasks
Fractal uncovers abundant nested parallelism

Flat

<table>
<thead>
<tr>
<th></th>
<th>maxflow</th>
<th>bayes</th>
<th>labyrinth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1c</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>128c</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>256c</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Flat

<table>
<thead>
<tr>
<th></th>
<th>maxflow</th>
<th>bayes</th>
<th>labyrinth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>322x</td>
<td>128</td>
<td></td>
</tr>
</tbody>
</table>

Fractal

<table>
<thead>
<tr>
<th></th>
<th>maxflow</th>
<th>bayes</th>
<th>labyrinth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3590</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average task length (cycles)

Flat

<table>
<thead>
<tr>
<th></th>
<th>3260</th>
<th>1.8 M</th>
<th>16 M</th>
</tr>
</thead>
</table>

Fractal

<table>
<thead>
<tr>
<th></th>
<th>373</th>
<th>3590</th>
<th>220</th>
</tr>
</thead>
</table>

Average task length (cycles)
Fractal avoids over-serialization

<table>
<thead>
<tr>
<th>Flat</th>
<th>Swarm</th>
<th>Fractal</th>
</tr>
</thead>
<tbody>
<tr>
<td>162</td>
<td>633</td>
<td>113</td>
</tr>
<tr>
<td>96</td>
<td>49</td>
<td></td>
</tr>
</tbody>
</table>

Flat: 26x—98x
Swarm: 21x—119x
Fractal: 40x—145x

Average task length (cycles)
Conclusion

Speculative systems must extract nested parallelism in order to scale large, complex, real-world applications

**Fractal**: An execution model for fine-grain nested speculative parallelism
- Decouple atomicity from parallelism
- Guarantee atomicity by ordering tasks

**Fractal** unlocks the benefits of fine-grain speculative parallelism
- Parallelizes many challenging workloads
- Enables composition of speculative parallel algorithms
Thank You! Questions?

Speculative systems must extract nested parallelism in order to scale large, complex, real-world applications

**Fractal**: An execution model for fine-grain nested speculative parallelism
  - Decouple atomicity from parallelism
  - Guarantee atomicity by ordering tasks

**Fractal** unlocks the benefits of fine-grain speculative parallelism
  - Parallelizes many challenging workloads
  - Enables composition of speculative parallel algorithms