Concurrent Programming with Revisions and Isolation Types

2010

Authors:
Sebastian Burckhardt (Microsoft Research)
Alexandro Baldassin (State University of Campinas)
Daan Leijen (Microsoft Research)

Presented by: Yves Bonjour
Motivation

\begin{itemize}
\item x\(=\)5;
\item x = x + 1;
\item x = 2*x;
\item assert(x==?);
\end{itemize}

Thread P

Thread Q
Concurrent Revisions

• Concept from Source Control Systems

• Revisions
  – Run in isolation
  – Operate on copy of shared state
  – Resolve write conflicts
Revision diagrams

\[ x = 5 \]

fork

revision

join

main

rev

revision

fork

revision

join

main
Example - Versioned

```
x=5
```

```
x=x+1
```

```
x=2*x
```

```
x == 5
```

```
x == 6
```

```
x == 10
```

```
x == 10
```

```
x == 5
```

```
Versioned<int>
```

```
main
```
Example - Counter

Cumulative<int, f>

```
c = 2
```

```
c += 1
```

```
c += 1
```

```
c == 3
```

```
c == 3
```

```
c == 3
```

```
c == 3
```
Cumulative Types

Merge function f:

```c
int f(int orig, int master, int revised)
master+(revised-orig)
```

```c
main
c += 1
c += 1

c = 2
c += 1
```

```c
rev
c += 1
c += 1
c = 3+3-2 = 4
```
... [Versioned]
int x=5;
...
RevisionTask r = CurrentRevision.Fork(() => {
    x = 2*x;

});
x=x+1
CurrentRevision.Join(r);
...
# Implementation

<table>
<thead>
<tr>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

- Lazy copy
- Release unused values
- Lock free
Case Study – SpaceWars3D

12’000 lines of code
Execute tasks in parallel

Architecture

Simulate Physics
Send
Parallel Collision Detection

Render Screen
Play Sounds
Process Inputs

Graphics Card
Keyboard

Network Connection
Disk
Receive Autosave

e.g. positions

Causes freezes

[Concurrent Programming with Revisions and Isolation Types, S. Burckhardt et. al., 2010]
Methodology

• Typical game
  – 2 players
  – 800 asteroids

• #asteroids → proportion of parallelizable computation

• Record and Replay

• 2000 frames
Revision Overhead

Average frame time breakdown [ns]
Revision Overhead (2)

Minimum overhead

Reads dominate

[Concurrent Programming with Revisions and Isolation Types, S. Burckhardt et. al., 2010]
Runtime Of Join Operations

- Join operations: 1.2%
- Other operations: 98.8%
Parallel Speed Up (4 cores)

Frames per second [fps]

[Concurrent Programming with Revisions and Isolation Types, S. Burckhardt et. al., 2010]
Conclusion

- Simple
- Data centric
- Lock free
- Reasoning with revision diagrams
  - Deterministic

- Performance with more writes?
- Join time accuracy?
  - More complex data structures
rise4fun.com/Revisions
Additional slides
Control The Sideeffects
Proportion Of Isolation Types

- Versioned: 82%
- Cumulative: 18%
Memory Overhead

Allocated managed memory [bytes]

# asteroids

<table>
<thead>
<tr>
<th># asteroids</th>
<th>Allocated managed memory [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>1,000,000</td>
</tr>
<tr>
<td>900</td>
<td>1,100,000</td>
</tr>
<tr>
<td>1000</td>
<td>1,200,000</td>
</tr>
<tr>
<td>1100</td>
<td>1,300,000</td>
</tr>
<tr>
<td>1200</td>
<td>1,400,000</td>
</tr>
<tr>
<td>1300</td>
<td>1,500,000</td>
</tr>
<tr>
<td>1400</td>
<td>1,600,000</td>
</tr>
<tr>
<td>1500</td>
<td>1,700,000</td>
</tr>
</tbody>
</table>

Revisions: 1.36
Sequential: 1.49