ROSS for PDES Research

Elsa Gonsiorowski
Justin LaPre
CD Carothers
CD Carothers
“Efficient Optimistic Parallel Simulations Using Reverse Computation”
“Efficient Optimistic Parallel Simulations Using Reverse Computation”
“On the Role of Burst Buffers in Leadership-Class Storage Systems”
Research vs Software Engineering

- Doxygen Automated Documentation
- GitHub (and GitHub Issues)
- LP-Printf
- Continuous Integration with Travis
- AVL Tree
- LORAIN
- Delta Encoding
Research vs Software Engineering

- AVL Tree (PADS 2014)
- LORAIN (PADS 2014)
- Delta Encoding (WSC 2015)
- Gates (MASCOTS 2012)
- Automatic Model Generation (PADS 2015)
Issues

Open
- gonsie: 1
- carns: 1
- markplagge: 5
- carn: 14

Closed
- JohnPJenkins: 1
- laprej: 1
- mmubarak: 2
- Open issue: 12
# Punch Card

<table>
<thead>
<tr>
<th></th>
<th>12a</th>
<th>1a</th>
<th>2a</th>
<th>3a</th>
<th>4a</th>
<th>5a</th>
<th>6a</th>
<th>7a</th>
<th>8a</th>
<th>9a</th>
<th>10a</th>
<th>11a</th>
<th>12p</th>
<th>1p</th>
<th>2p</th>
<th>3p</th>
<th>4p</th>
<th>5p</th>
<th>6p</th>
<th>7p</th>
<th>8p</th>
<th>9p</th>
<th>10p</th>
<th>11p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuesday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wednesday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thursday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saturday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Research

- AVL Tree (PADS 2014)
- LORAIN (PADS 2014)
- Delta Encoding (WSC 2015)
- Gates (MASCOTS 2012)
- Automatic Model Generation (PADS 2015)
- RIO (??)
Gate-Level Circuits Simulation

- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates

*Timestamp: 1  2  3  4*
Gate-Level Circuits Simulation

- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates

Timestamp: 1 2 3 4
Gate-Level Circuits Simulation

- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates
Gate-Level Circuits Simulation

- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates
Gate-Level Circuits Simulation

- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates
Gate-Level Circuits Simulation

- *Events* are electrical signals

- *Logical processes* (LPs) are the boolean logic gates

```
Timestamp: 1                  2                 3                  4
```
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving

Timestamp: 1  2  3  4
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving

Timestamp: 1       2       3       4
PDES: Optimistic Synchronization

- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving

Timestamp: 1  2  3  4
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving

Timestamp: 1 2 3 4
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving

Timestamp: 1  2  3  4
PDES: Optimistic Synchronization

- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving

Timestamp: 1  2  3  4
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving
PDES: Optimistic Synchronization

- Time Warp algorithm
- minimal global time synchronization
- local recovery when out-of-order event detected
- Reverse computation and/or state-saving

Timestamp: 1  2  3  4
PDES Research

• How can we automate event rollback?
• Incremental state-saving with delta encoding
• Automatic reverse computation with LORAIN
Delta Encoding

• Motivation: not all event handlers are reversible, e.g., while loop containing if statements

• We have to fall back to state-saving, but consumes too much memory!

• Source control systems typically store changes as deltas
  • Often a substantial win when changes are small

• PDES state changes typically small

• Deltas on sizable models will likely have many zeroes, i.e., things are the same

• Excellent opportunity for compression!
Approach

- In the main event loop in ROSS:
  - `memcpy()` LP state into a buffer `state_{before}`
  - run event
  - Now we have `state_{after}`
  - Call `delta_diff` function on `state_{before}` and `state_{after}`
  - Compress diff with LZ4, save with event
Example: OLSR Protocol
OLSR Performance for Various Approaches

OLSR run time comparison for delta encoding, conservative, and optimistic with state saving

Running time (seconds)

Cores used

128 518 2048

Delta encoding
Conservative
Optimistic w/ state saving
Delta Encoding Uses Far Less Memory!

OLSR model memory consumption for delta encoding and state saving

Memory used (KB)

LP State
Event Memory
Buddy System

Delta Encoding
State Saving
Motivation Behind LORAIN

- Writing reverse code is hard!

- Example: swapping into messages is common!

```c
// Forward
if (msg->val == 1) {
    SWAP(lp->val, msg->val);
}

// Reverse
if (msg->val == 1) {
    SWAP(lp->val, msg->val);
}
```

Looks “obviously” correct forward and backward, but it’s not.

```c
// Forward
if (msg->val == 1) {
    SWAP(lp->val, msg->val);
}

// Reverse
if (lp->val == 1) {
    SWAP(lp->val, msg->val);
}
```

Hard error to spot!
void test_if(void)
{
    if (test_if_x_condition) {
        test_if_x = test_if_x + 1;
    }
}

define void @test_if() nounwind uwtable ssp {
    %1 = load i32* @test_if_x_condition, align 4
    %2 = icmp ne i32 %1, 0
    br i1 %2, label %3, label %6

    ; <label>:3 ; preds = %0
    %4 = load i32* @test_if_x, align 4
    %5 = add nsw i32 %4, 1
    store i32 %5, i32* @test_if_x, align 4
    br label %6

    ; <label>:6 ; preds = %3, %0
    ret void
}
3 Steps

- Augment the appropriate “message” data structure to support swap operations
  - Catch “destructive” operations e.g., \( x = 5 \)
- Instrument / analyze the forward event handler
  - Mark non-important store instructions with metadata
- Clone and invert the forward event handler
Continuing Research
RIO: Checkpointing API For ROSS

- checkpoint.readme.txt
- checkpoint.metadata.mh
- checkpoint.data-#
Using RIO

- Compile in with CMake USE_RIO0 option

- --io-parts=n and --io-files=n flags or function call

- io_lptype struct

- Mapping extension

- Implement: Model size function

- Implement: Serialize and deserialize functions

