Cloud BDDs

Randal E. Bryant
Carnegie Mellon University

http://www.cs.cmu.edu/~bryant
Ed’s PhD Students

* Will Klieber
* Sicun Gao
Himanshu Jain
Nishant Sinha
* Pankaj Chauhan
* Muralidhar Talupur
* Anubhav Gupta
Alex Groce
* Sagar Chaki
* Dong Wang
* Sergey Berezin
Wilfredo Marrero
* Yuan Lu
* Marius Minea

Vicky Hartonas-Garmhausen
Somesh Jha
Sergios Campos
* Xudong Zhao
* David E. Long
* Jerry. R. Burch
* Ken L. McMillan
M.C. Browne
* David L. Dill
Bud Mishra
A. Prasad Sistla
Christos N. Nikolaou
E. Allen Emerson
Proposed System Structure

Clients: $C_1, C_2, \ldots, C_m$

Routers: $X_1, X_2, \ldots, X_k$

Workers: $S_1, S_2, \ldots, S_n$

BDD Database

$m \approx 100$  $k \approx 20$  $n \approx 1000$
Traditional BDD Representation

Based on Pointers
- Node represented by address
- Location of information about node

All data within single address space
Shared-Nothing Implementation

Only Way to Achieve True Scalability
- Large number of low-cost nodes
- Single resource shared by many users

Distribute Data Structures Across Processors

Must find alternative to pointer-based representation
Ref-Based BDD Representation

“Ref” Encodes

- Node type
  - Constant, Variable, Function
- Top-level variable
- Hash of components
- Uniquifier
  - To resolve hash collisions
Storing a Ref

- Entry describing node stored according to its hash signature
- Unique table distributed across workers according to hash

<table>
<thead>
<tr>
<th>Ref</th>
<th>F8.ed4c70.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>8</td>
</tr>
<tr>
<td>TRef</td>
<td>V10</td>
</tr>
<tr>
<td>ERef</td>
<td>F9.71a09b.1</td>
</tr>
</tbody>
</table>

BDD Servers

$h \rightarrow \text{mod } n \rightarrow i \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_i \rightarrow S_n$
Dereferencing a Ref

Hash signature in Ref enables retrieval of components
**Data Flow Execution Model**

**Concept**

- Computation expressed as dynamically generated network of operators
- Operator has fixed number of operands + destination
- When all operands available, operator *fires*
  - Perform computation
  - Send one or more operands to other operators
  - Generate one or more operators
  - Disappear
Implementing ITE

**Request**
- Compute \((f \land g) \lor (\neg f \land h)\)
- Send result \(r\) to \(\text{dest}\)

**Outcomes**
- Early termination if special case or result found in memo table
- Otherwise, up to 9 operations + 2 recursive calls
Implementation

Data Flow BDD combination of:

- General-purpose data flow on top of sockets interface
- Ref-based BDD
  - Can also execute with standard, depth-first traversal

Client Interface

- Any combination of data flow, sequential, CUDD
  - Isomorphic results
  - For testing and performance comparison
Some Results

N = 14

- With help from Hemanth Kini
- Boolean function representing all legal configurations
- Peak nodes = 23M
- Total ITEs = 233M
- Total OPs = 837M
More Results

N = 15

- Require 8 processors to have enough memory
- Peak nodes = 95M
- Total ITEs = 1.1B
- Total OPs = 3.9B
Implications

For BDDs

- Scale to much larger sizes
- Allow sharing across multiple runs and users
  - View as dynamically constructed, distributed database

For Parallel Computation

- Execution model to support dynamic graph algorithms
- Combines data flow + distributed hash table
  - Actions triggered by message passing
  - Locate objects by hash function
- Features
  - Overcome latency with high throughput
  - Scalable to arbitrary number of processors