Combining P2 and RDL to Build Dataflow Hardware Programs

Greg Gibeling, Nathan Burkhart, and Andrew Schultz
University of California, Berkeley   {gdgib, burkhart, alschult}@berkeley.edu

Project Overview

The Overlog language in the P2 system provides a flexible, high level language for describing Overlay networks, with good performance and significantly lower code maintenance costs.

RDL aims to provide similar benefits to computer architecture researchers on the multi-University RAMP project, by tying together hardware and software components from different researchers.

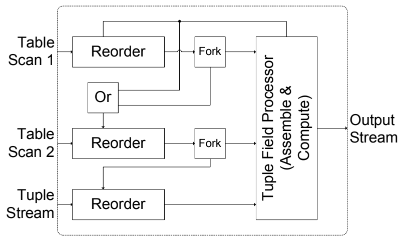
P2 is based on the Click dataflow engine, which is similar in concept (though very different in execution) to the RDL model of message passing units. By merging P2 with RDL we can improve the flexibility of P2, while supporting new applications such as hardware implementations.

Applications

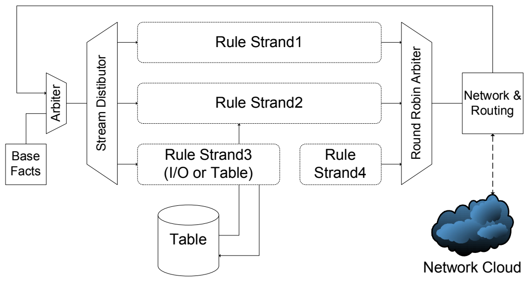
- Overlay Networks & P2:
  Chord and other DHTs are decent examples, but other P2 applications such as Narada or Paxos should show interesting results. FPGAs could reduce test time and space, in addition to providing wire-speed implementations.
- Distributed System Debugging:
  The Overlog language could be used to specify breakpoints and watch expressions in a large distributed system, given a debugging interface such as the one RDLC will be able to automatically insert into both hardware and software desgins.
- Computing Clusters:
  The BEE2 is very useful for supercomputing, but designs are often highly network specific. By using Overlog with RDL, computation and communication can be loosely, but automatically coupled across heterogeneous hardware topologies.

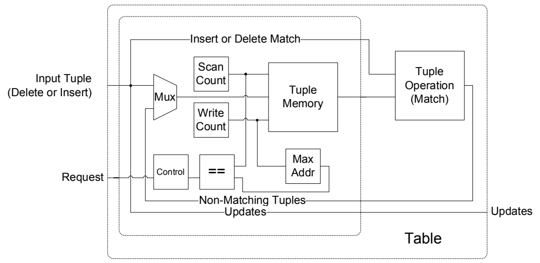
System Architecture

- Data Representation:
  P2 and Overlog deal in both streams and materialized tables of "tuples." In RDL these are represented by serial a stream of tuple field messages. Node addresses and stream ID numbers are prepended during transmission.
- Tables and Storage:
  Materialized tables in hardware are realized as a memory containing a list of tuples in the format described above. Most operations are implemented as a linear table scan. This obviates the need for CAMs at the cost of higher latency for operations (especially joins). We aim to prove or disprove the validity of this design tradeoff.
- Rule Strands:
  Overlog programs are a series of rules for processing tuples. Each rule is translated into a (roughly) linear dataflow through reorder and tuple operation units. A rule can only have a single event-driven input, the remainder of them will be table scans or lookups. The Overlog compiler supports very general rule-rewriting to ensure that rules are localized to a single node, and have exactly one event source.



Tuple Operation (Rule Strand)





Languages and Compilers

```
unit <width> {
   instance  IO::BooleanInput       BooleanInputX(Value(InChannel));
   instance  Counter<$width>        CounterX(InChannel, OutChannel);
   instance  IO::DisplayNum<$width> DisplayNumX;

   channel                    InChannel;
   channel                    OutChannel { -> DisplayNumX.Value };
}                             CounterExample;

unit <width = 32, saturate = 1> {
   input     bit<1>           UpDown;
   output    bit<$width>      Count;
}                             Counter;

unit {
   output    bit<1>           Value;
}                             BooleanInput;

unit <width = 32> {
   input     bit<$width>      Value;
}                             DisplayNum;
```

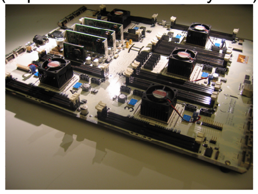
RDL CounterExample (http://ramp.eecs.berkeley.edu)

- RDL:
  Structural netlisting language for mixed hardware and software systems. Includes complex message type handling. Output is effectively a distributed event simulator, in a dataflow or Kahn/MacQueen style.
- Overlog:
  Declarative dataflow language centered around tuple streams. A derivative of Datalog. Useful for overlay networks, and other distributed message passing systems.
- RDLC2:
  RDL Compiler, includes hardware generation and an advanced plugin architecture, which has been used to integrate the Overlog planner and elements.

Simple Overlog Fragment

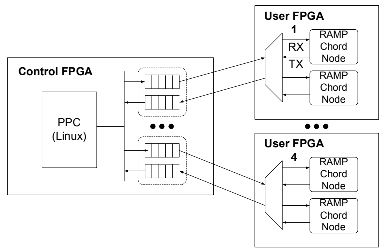
```
stream Stream0(Int, Int);
stream Stream1(Bool);

watch Stream0;
watch Stream1;

Stream1(true);
Stream0(0, 1);
Stream0(2, 3);
Stream1(false);
```

BEE2
(http://bee2.eecs.berkeley.edu)



Test Infrastructure (BEE2)

- BEE2 Test Bed:
  The test bed is built on the BEE2 FPGA computing module. Each module consists of five FPGAs with a variety of parallel and serial connections.
- Control FPGA:
  Runs linux on the embedded PowerPC 405, along with a simple network, to allow full monitoring and injection of data.
- Network:
  Simple network uses 64b star connections between user and control FPGAs. A higher performance network has been built which uses the 138b ring between user FPGAs.


Control FPGA
PPC (Linux)
User FPGA 1
RX
TX
RAMP Chord Node
RAMP Chord Node
User FPGA 4
RAMP Chord Node
RAMP Chord Node