Auto-Parallelization for Declarative Network Monitoring
Robert Soulé (NYU), Robert Grimm (NYU), and Petros Maniatis (Intel Research Berkeley)

### Network Monitoring Problem

- **Network monitoring is too hard**
  - A variety of tasks in the same setting (signature matching, anomaly detection, forensic analysis, etc.)
  - Different operational patterns (local vs. distributed, long-running and infrequent vs. interactive and one time, etc.)
  - Deployed across different environments (single heavy-duty compute server, large clusters of inexpensive machines, the internet, etc.)
  - In many cases, computations may be interdependent

### Approach

- **Declarative programming helps**
  - Higher level language abstracts away many details
  - Not quite a complete solution
  - Need to focus on scalability
- **Static analysis identifies parallelism**
  - Scheduling decisions more accessible to the compiler
- **Additional declarations inform scheduler**
  - Augmented code allows concurrent execution

### Source Code

```
r1 synIn(@LclAddr, SrcAddr, f_now()) :-
  pktIn(@LclAddr, SrcAddr, D),
  f_topSyn(D).

r2 rstIn(@LclAddr, SrcAddr, f_now()) :-
  pktIn(@LclAddr, SrcAddr, D),
  f_topRst(D).

r3 synackIn(@LclAddr, SrcAddr, f_now()) :-
  pktIn(@LclAddr, SrcAddr, D),
  f_topSynAck(D).

r4 ackIn(@LclAddr, SrcAddr, f_now()) :-
  pktIn(@LclAddr, SrcAddr, D),
  f_topSynAck(D).

r5 alarm(@LclAddr, SrcAddr, "CONNRRST", f_now()) :-
  rstIn(@LclAddr, SrcAddr, TRST),
  synIn(@LclAddr, SrcAddr, TSYN),
  synackOut(@LclAddr, SrcAddr, TSYNACK),
  not ackIn(@LclAddr, SrcAddr, TACK),
  TACK > TSYNACK > TSYN,
  TRST > TSYNACK.

r6 alarm(@LclAddr, SrcAddr, "SPIKE", f_now()) :-
  cpuSpike(@LclAddr, USAGE),
  USAGE > UMAX.
```

### Types of Parallelism

- **Inter-fixpoint**
  - Multiple fixpoint computations may proceed at the same time
  - Compute the transitive closure of all rules, partition tuples into read and write sets
- **Intra-fixpoint**
  - Rules within a fixpoint can execute in parallel because the state is static and language is single assignment
  - Need to be careful with side effects from function calls
- **Data Parallelism**
  - Single instruction stream operating on multiple data set (applying the same operation on every item in a list)
  - Requires runtime analysis (an estimate of data set size and latency of state transfer)

### Analysis

1. **Dependencies**

   - Compute the dependencies between tuples on a per-rule basis.

   ![Diagram of dependencies](dependency_diagram.png)

2. **Transitive Closure**

   - Perform a depth first search of the dependency graph.
   - Start at the non-materialized tuple on the right hand side of a rule.
   - Stop recursion at an external or materialized tuple on the left of a rule.
   - Mark tuples as read or write.

   ![Diagram of transitive closure](transitive_closure_diagram.png)

3. **Conflict Analysis**

   - Remove the next event from the external event queue.
   - Consult the concurrency table to compare read and write sets for event and running fixpoints.

   ![Diagram of conflict analysis](conflict_analysis_diagram.png)

4. **Scheduling**

   - Support multiple internal event queues.
   - Check if:
     - `WriteNew ∩ (ReadCurrent ∪ WriteCurrent) = ∅`
     - `WriteCurrent ∩ (ReadNew ∪ WriteNew) = ∅`
   - If no conflict, enqueue event on its own internal event queue and start execution.

5. **Speculative Scheduling**

   - An alternative is to speculatively execute and check for conflict afterwards.
   - Best strategy depends on expected frequency of conflict.

   ![Diagram of speculative scheduling](speculative_scheduling_diagram.png)