Compiling Packet Programs to Reconfigurable Switches: Theory and Algorithms

EuroP4 2020

Balázs Vass, Budapest Univ. of Technology and Economics, Budapest, Hungary
Erika Bérczi-Kovács, Eötvös Loránd University, Budapest, Hungary
Costin Raiciu, University Politehnica of Bucharest, Romania
Gábor Rétvári, Budapest Univ. of Technology and Economics, Budapest, Hungary
Introduction: Pipeline Embedding Problem
Reconfigurable Switch Pipelines

- Programming pipelines using a high-level domain-specific language like P4 is increasingly adopted

- Applications booming →
  - dataplane programs
    - grow in complexity
  - new programmable switch ASICs:
    - more dataplane resources
    - more pipeline stages

- → Algorithmic issues
Pipeline Embedding Problem

- Dataplane programming: top-down approach
  - required behavior of the network described in a declarative P4 program
  - mapped to hardware by a P4 compiler

- The compiler must analyze the P4 program
  - given an abstract model of the hardware:
    - limits of memory space, width, types,
    - # processing stages
    - max. level of concurrency at each stage, ...
  - finds the best encoding such that:
    - all constraints are met
  - 'best': min. # stages, min power, etc.

- We call this the Pipeline Embedding Problem
Pipeline Embedding

- Stage for Pipeline Embedding set by [NSDI’15]:

- Proposed:
  - Abstract model for Pipeline Embedding
  - ILP + heuristic algorithms

- Issues:
  - ILP: possibly exponential runtime (runs for hours for a moderate-sized pipeline)
  - heuristics: no proven guarantees of ‘goodness’

- Unfolding the algorithmic landscape of Pipeline Embedding was required
Models of programs and pipelines

- **Control-flow dependencies of a P4 program**
  - represented by a directed acyclic graph (DAG)
  - called Table Dependency Graph (TDG)
  - vertices: logical match-action tables (MATs)
  - arcs: dependencies between the MATs (match, action, etc.)

- **Packet processing pipeline**:
  - modeled as a directed path
  - nodes $s_1, s_2, \ldots$ represent the pipeline stages
  - arcs $(s_i, s_{i+1})$ encode succession
  - For simplicity:
    - the switch has infinitely many stages,
    - objective: minimize the # stages in the embedding.
Hardware constraints: Simplified models

Full hardware model: very complex → simplifications → gained some insight → some constraints put back → reanalysed

<table>
<thead>
<tr>
<th>Model name</th>
<th>INF-CAP</th>
<th>1D1R</th>
<th>1D1R-hsplit</th>
<th>2D1R</th>
<th>2D2R</th>
<th>2D2R-T/S</th>
<th>2D2R-PISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the</td>
<td>(mapping concurrency</td>
<td>1D capacity/</td>
<td>hsplit (table</td>
<td>2D capacity/</td>
<td>2 kinds of</td>
<td>limited number of</td>
<td>crossbar</td>
</tr>
<tr>
<td>previous model</td>
<td>due to dependency)</td>
<td>demands</td>
<td>entries split</td>
<td>demands</td>
<td>resources per</td>
<td>tables per stage</td>
<td>constraints, word packing,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>between stages</td>
<td></td>
<td>stage</td>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>

- INF-CAP: a directed path of stages, each with infinite capacities (no arc of TDG mapped to just one stage)
- 2D2R-PISA: a full-blown PISA model (RMT described in [NSDI’15])
Results
## Results - Complexity

<table>
<thead>
<tr>
<th>Model name</th>
<th>INF-CAP (mapping concurrency due to dependency)</th>
<th>1D1R (1D capacity/demands)</th>
<th>1D1R-(h\text{split}) (table entries split between stages)</th>
<th>2D1R (2D capacity/demands)</th>
<th>2D2R (2 kinds of resources per stage)</th>
<th>2D2R-T/S (limited number of tables per stage)</th>
<th>2D2R-PISA (crossbar constraints, word packing, etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the previous model</td>
<td>P</td>
<td>NPC</td>
<td>NPC</td>
<td>NPC</td>
<td>NPC</td>
<td>NPC strongly</td>
<td>NPC-NP-hard</td>
</tr>
</tbody>
</table>

- NP complete even with simple capacity constraints (1D1R)
- Hint of proof: some NP-hard problems are apparently special cases
More bad news: Inapproximability

<table>
<thead>
<tr>
<th>Model name</th>
<th>INF-CAP</th>
<th>1D1R</th>
<th>1D1R-hsplit</th>
<th>2D1R</th>
<th>2D2R</th>
<th>2D2R-T/S</th>
<th>2D2R-PISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the previous model</td>
<td>(mapping concurrency due to dependency)</td>
<td>1D capacity/demands</td>
<td>hsplit (table entries split between stages)</td>
<td>2D capacity/demands</td>
<td>2 kinds of resources per stage</td>
<td>limited number of tables per stage</td>
<td>crossbar constraints, word packing, etc.</td>
</tr>
<tr>
<td>Bad news: (unless P=NP,) Inapproximable better than . . .</td>
<td>OPT</td>
<td>3/2*OPT</td>
<td>5/4*OPT</td>
<td>5/4*OPT</td>
<td>5/4*OPT</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

- No Polynomial Time Approximation Scheme (PTAS) exists (no poly. alg. with arbitrary multiplicative error)
- Bird’s view of proofs:
  - showing a problem instance family s.t:
    - we can embed each instance in k stages exactly if a related NP-hard problem has a solution
    - otherwise we need (k+1) stages \( \rightarrow \text{inapprox. better than}(k+1)/k \times \text{OPT} \)

E.g. for 1D1R (oversimplified):
- no TDG arcs
- \( \Sigma (\text{TDG node sizes}) = 2^* \) (stage size)
- We can embed in k=2 stages exactly if the PARTITION has a solution over the table sizes
- ...that is NP-hard.
Good news: constant(!)-approximability in quasi-linear time

<table>
<thead>
<tr>
<th>Model name</th>
<th>INF-CAP</th>
<th>1D1R</th>
<th>1D1R-hsplit</th>
<th>2D1R</th>
<th>2D2R</th>
<th>2D2R-T/S</th>
<th>2D2R-PISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the previous model</td>
<td>(mapping concurrency due to dependency)</td>
<td>1D capacity/demands</td>
<td>hsplit (table entries split between stages)</td>
<td>2D capacity/demands</td>
<td>2 kinds of resources per stage</td>
<td>limited number of tables per stage</td>
<td>crossbar constraints, word packing, etc.</td>
</tr>
<tr>
<td>Good news: Constant-approximable in...</td>
<td>OPT</td>
<td>3*OPT</td>
<td>2*OPT</td>
<td>3*OPT</td>
<td>(5 to 8)*OPT (°)</td>
<td>(6 to 9)*OPT (°)</td>
<td>?</td>
</tr>
</tbody>
</table>

- Approximation idea: (First Fit by Level and Size)
  - group the TDG nodes by their level
    - node $v$ on level $i$ if the longest directed path from the root $R$ to $v$ has a length of $i$
  - nodes in each level can be mapped in the same stage
  - for each level: ~bin packing (without dependency constraints)
    - ...or combining bin packings

![Diagram showing TDG nodes and levels]

Levels: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$
Conclusion & Future Work

Take-away: Pipeline Embedding is

- NP-hard 😞
- inapproximable in poly. time 😞
  (unless P=NP)
- constant-approximable 😓

We will investigate:

- Optimality gaps of Chipmunk, Domino, & others?
- How to write better P4 programs?
Thank you for your attention
Q&A