Compiling Packet Programs to dRMT Switches: Theory and Algorithms

at EuroP4 2022, Rome

Balázs Vass$^{1,2}$, Ádám Fraknói$^3$, Erika Bérczi-Kovács$^{4,3,5}$, Gábor Rétvári$^1$

$^1$Budapest University of Technology and Economics (BME), Budapest, Hungary
$^2$ELKH-BME Information Systems Research Group
$^3$Eötvös Loránd University (ELTE), Budapest, Hungary
$^4$Alfréd Rényi Institute of Mathematics, Budapest, Hungary
$^5$MTA-ELTE Egerváry Research Group on Combinatorial Optimization
Motivation
Motivation I.

- We witness more and more complex
  - P4 programs
  - programmable switch ASICs
- Nowadays: RMT architectures deployed (Intel Tofino)
- Drawbacks of RMT:
  - table memory: local to pipeline stage $\rightarrow$ memory not used
    - by one stage cannot be reclaimed by another
  - sequentially executes matches followed by actions as packets traverse pipeline stages.
- solution: distributed RMT (dRMT)
  [SIGCOMM ‘17](with concrete HW design & cost analysis):
  - moves table memories out of pipeline stages and into a centralized pool that is accessible through a crossbar.
  - replaces RMT’s pipeline stages with a cluster of processors that can execute match and action operations in any order
Motivation II.

- Mapping a P4 program to hardware is critical in compilation
  - P4 program represented as a DAG of match / action nodes + dependencies (ODG, operation dependency graph)
  - abstract model of hardware resources
- Prior work on ODG embedding [SIGCOMM ’17]:
  - only cyclic embeddings - the same scheme repeated to every packet to reduce compilation complexity
    - aim: minimize \( P := \# \) processors to achieve line rate
  - algorithmic issues:
    - ILP: no time guarantees
    - heuristics: no approximation guarantees
- Question: complexity of the problem, efficient algorithms
Problem formulation
# Simplified pipeline models

<table>
<thead>
<tr>
<th>Model name:</th>
<th>BASIC</th>
<th>IPC1</th>
<th>WIDTH</th>
<th>WIDTH-IPC1</th>
<th>WIDTH-IPC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the basic constraints</td>
<td>(basic model)</td>
<td>Max. 1 packet per processor per cycle (IPC= 1)</td>
<td>arbitrary table widths</td>
<td>arbitrary table widths + IPC= 1</td>
<td>arbitrary table widths + IPC= 2 (≤2 pkt./proc./cycle)</td>
</tr>
</tbody>
</table>

- **BASIC:**
  - P4 program as ODG $D = (V, E), V = V_A \cup V_M$
    - match, action nodes and inter-dependencies
    - $\Delta M$ and $\Delta A$: # proc. cycles to wait after a match/action starts
    - each processor in each cycle can initiate up to $M$ parallel table searches
    - ... and modify up to $A$ action fields in parallel
- **IPC1:** each processor in each cycle can only start matches up to $\text{IPC}=1$ packets. Same for actions
- **WIDTH:** each match / action node has a width (measured in positive integers)
Theoretical Results
Results - Complexity

- The relaxed model is solvable in polynomial time
- Introducing width or Inter Packet Concurrency makes it NP-hard

<table>
<thead>
<tr>
<th>Model name:</th>
<th>BASIC</th>
<th>IPC1</th>
<th>WIDTH</th>
<th>WIDTH-IPC1</th>
<th>WIDTH-IPC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the basic constraints</td>
<td>(basic model)</td>
<td>Max. 1 packet per processor per cycle (IPC= 1)</td>
<td>arbitrary table widths</td>
<td>arbitrary table widths + IPC= 1</td>
<td>arbitrary table widths + IPC= 2 (\leq 2) pkt./proc./cycle</td>
</tr>
<tr>
<td>Complexity</td>
<td>(\mathcal{P})</td>
<td>(\mathcal{NP})-hard</td>
<td>(\mathcal{NP})-hard</td>
<td>(\mathcal{NP})-hard</td>
<td>?</td>
</tr>
</tbody>
</table>

- Hint of proof for BASIC is polynomial:
  - \(\text{max (\#match nodes, \#action nodes) / memory width: lower bound on } \mathcal{P}\)
  - this is enough, “almost greedy” embedding in \(O(|E| + |V| |P|)\).
- NP-hardnesses:
  - reductions to CLIQUE and EQUAL CARDINALITY PARTITION
### (In-) approximability

<table>
<thead>
<tr>
<th>Model name:</th>
<th>BASIC</th>
<th>IPC1</th>
<th>WIDTH</th>
<th>WIDTH-IPC1</th>
<th>WIDTH-IPC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature on top of the</td>
<td>(basic model)</td>
<td>Max. 1 packet per</td>
<td>arbitrary table widths</td>
<td>arbitrary table widths</td>
<td>arbitrary table widths</td>
</tr>
<tr>
<td>basic constraints</td>
<td></td>
<td>processor per cycle</td>
<td></td>
<td>+ IPC= 1</td>
<td>+ IPC= 2 (≤ 2 pkt./proc./cycle)</td>
</tr>
<tr>
<td>Bad news: Inapproximable</td>
<td>OPT</td>
<td>4/3*OPT</td>
<td>3/2*OPT</td>
<td>3/2*OPT</td>
<td>?</td>
</tr>
<tr>
<td>better than . . . (unless P=NP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good news: Constant approximable</td>
<td>OPT</td>
<td>3*OPT</td>
<td>?</td>
<td>4*OPT</td>
<td>8*OPT</td>
</tr>
</tbody>
</table>

- Inapproximabilities: straightforward from the NP-hardness reductions
- Constant approximations:
  - There is a 4-approximating alg. for WIDTH-IPC1 (runs in \( O(|V| \log |V| + |E|) \))
  - it becomes 3-approximating for IPC1
  - ...and trivially 8-approx for WIDTH-IPC2
Our greedy

- Intuitive idea:
  - A variation of the First Fit Decreasing algorithm
    - take an arbitrary (random) topological order of the nodes
    - nodes with all predecessors embedded may be chosen to be embedded
  - Each bin can host either match or action nodes
  - We make each bin to use at least half of the width available
  - This uses at most 2x2 more bins than the optimum only taking in account the widths
  - This can be extended to a proper scheduling

```
Algorithm 1: WIDTH-IPC1 Our Greedy

Input: ODG \( D = (V, E) \); \( W : V \rightarrow \mathbb{N}^{+}; \bar{M}, \bar{A} \)
Output: \( PS : V \rightarrow \mathbb{N}^{+} \)
begin
    \( i := 1; V' := V \)
    while \( V' \neq \emptyset \) do
        \( a := \) list of action nodes with 0 indegrees, descending order of width
        \( m := \) list of match nodes with 0 indegrees, descending order of width
        \( w_a := \) sum of widths in \( a \)
        \( w_m := \) sum of widths in \( m \)
        current_usage := 0
        if \( w_m \geq \frac{1}{2}\bar{M} \) and \( w_a \geq \frac{1}{2}\bar{A} \) then
            Go to line 12 or 19
        if \( w_a \geq \frac{1}{2}\bar{A} \) and \( w_m < \frac{1}{2}\bar{M} \) then
            Go to line 19
        while \( m[0]+current\_usage \leq \bar{M} \) do
            current_usage += \( m[0] \)
            PS[\( m[0] \)] := \( i \)
            \( V' := V' \setminus \{m[0]\} \)
            \( m := m - m[0] \)
        \( i := i + 1 \)
        if \( w_m \geq \frac{1}{2}\bar{M} \) then
            continue
        while \( a[0]+current\_usage \leq \bar{A} \) do
            current_usage += \( a[0] \)
            PS[\( a[0] \)] := \( i \)
            \( V' := V' \setminus \{a[0]\} \)
            \( a := a - a[0] \)
        \( i := i + 1 \)
    return PS
```

B. Vass, Á. Fraknói, Erika Bérczi-Kovács, G. Révtári
Compiling Packet Programs to dRMT Switches: Theory and Algorithms
EuroP4 2022
Evaluation
Evaluation

- Graphs Egress, Ingress, Combined: derived from Switch.p4 (taken from the dRMT paper)
- Our greedy:
  - faster than the old rnd_sieve
  - yields at least as high throughput as rnd_sieve

![Table 2: Best P values computed by different algorithms](image1)

![Figure 3: Throughput provided by different heuristics as percentage of the best ILP solution](image2)
Conclusion & Future Work

- Algorithmic issues of P4 program embedding to dRMT tackled
- A practically useful constant-approximation algorithm introduced
- Lessons learned could be used in future HW design

- Sharp bounds, better algorithms for the different pipeline models, etc.
Thank you for your attention!

Q&A