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]

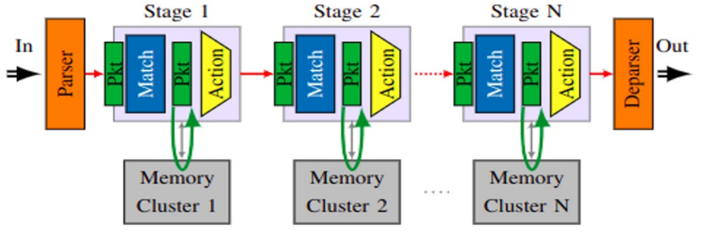
[1]Budapest University of Technology and Economics (BME), Budapest, Hungary
[2]ELKH-BME Information Systems Research Group
[3]Eötvös Loránd University (ELTE), Budapest, Hungary
[4]Alfréd Rényi Institute of Mathematics, Budapest, Hungary
[5]MTA-ELTE Egerváry Research Group on Combinatorial Optimization

Új Nemzeti Kiválóság Program
SINCE 1992 HSN Lab
M Ű E G Y E T E M 1 7 8 2
Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal

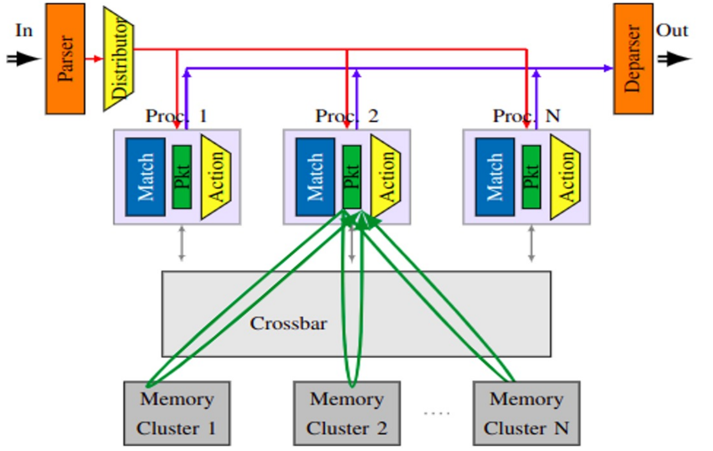
Motivation

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 –> 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



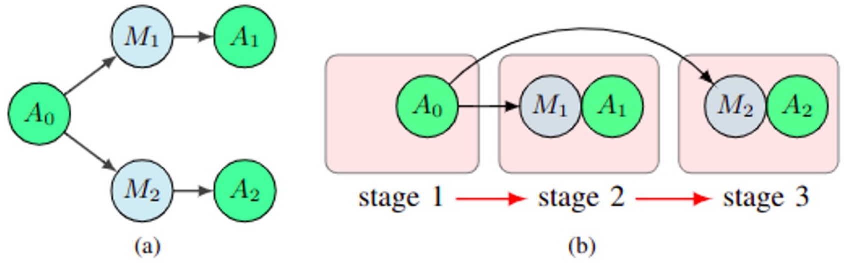
(a) RMT

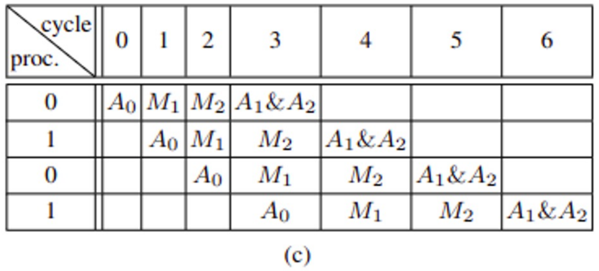


(b) dRMT

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





cycle / proc.	0	1	2	3	4	5	6
0	A_0	M_1	M_2	$A_1\&A_2$			
1		A_0	M_1	M_2	$A_1\&A_2$		
0			A_0	M_1	M_2	$A_1\&A_2$	
1				A_0	M_1	M_2	$A_1\&A_2$

(c)

Fig. 2: The ODG representation of a toy program (a), where A_i and M_i stand for action and match nodes/operations. Supposing a processor can initiate ≤ 1 match per clock cycle, (b) illustrates a straightforward RMT-embedding, (c) encodes an optimal dRMT-embedding of the program, where $P = 2$.

Problem formulation

Simplified pipeline models

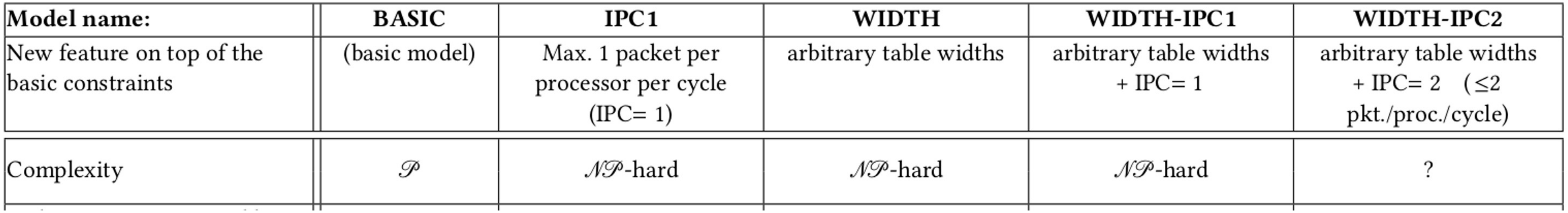
Model name:	BASIC	IPC1	WIDTH	WIDTH-IPC1	WIDTH-IPC2
New feature on top of the basic constraints	(basic model)	Max. 1 packet per processor per cycle (IPC= 1)	arbitrary table widths	arbitrary table widths + IPC= 1	arbitrary table widths + IPC= 2 (≤2 pkt./proc./cycle)

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

Theoretical Results

Results - Complexity

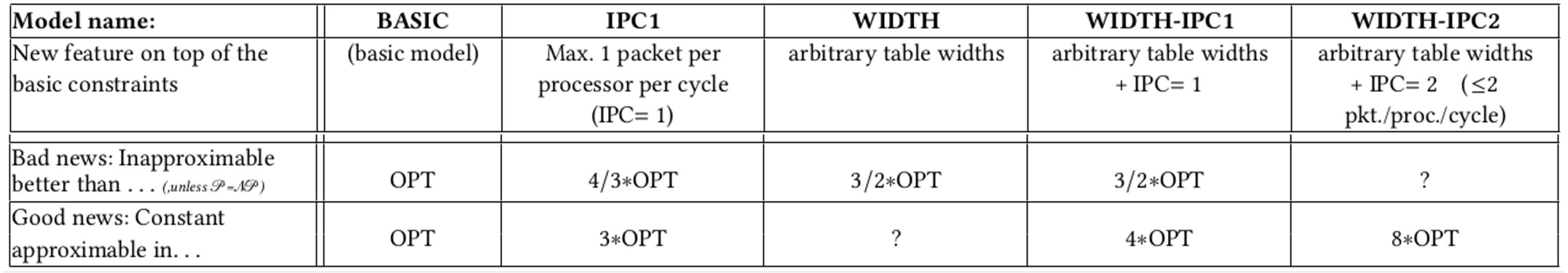
- The relaxed model is solvable in polynomial time
- Introducing width or Inter Packet Concurrency makes it NP-hard



Model name:	**BASIC**	**IPC1**	**WIDTH**	**WIDTH-IPC1**	**WIDTH-IPC2**
New feature on top of the basic constraints	(basic model)	Max. 1 packet per processor per cycle (IPC= 1)	arbitrary table widths	arbitrary table widths + IPC= 1	arbitrary table widths + IPC= 2 (≤2 pkt./proc./cycle)
Complexity	$\mathscr{P}$	$\mathscr{NP}$-hard	$\mathscr{NP}$-hard	$\mathscr{NP}$-hard	?

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

(In-) approximability

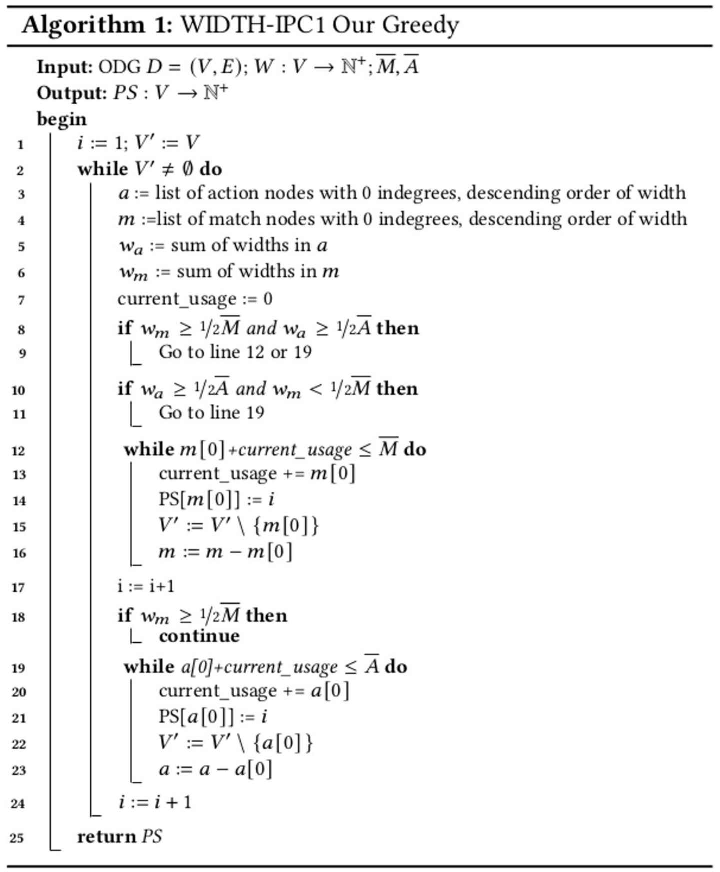


Model name:	BASIC	IPC1	WIDTH	WIDTH-IPC1	WIDTH-IPC2
New feature on top of the basic constraints	(basic model)	Max. 1 packet per processor per cycle (IPC= 1)	arbitrary table widths	arbitrary table widths + IPC= 1	arbitrary table widths + IPC= 2 (≤2 pkt./proc./cycle)
Bad news: Inapproximable better than . . . (,unless $\mathcal{P}=\mathcal{NP}$)	OPT	4/3∗OPT	3/2∗OPT	3/2∗OPT	?
Good news: Constant approximable in. . .	OPT	3∗OPT	?	4∗OPT	8∗OPT

- 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



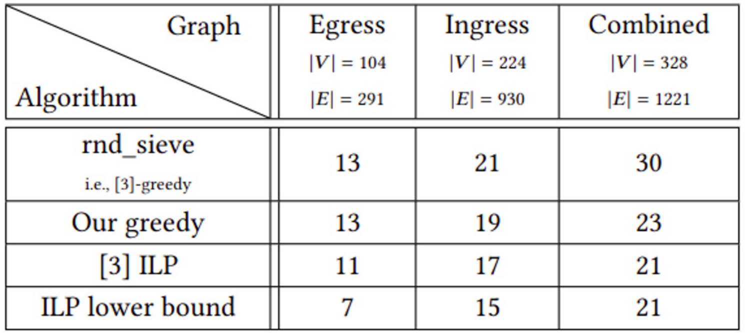
Algorithm 1: WIDTH-IPC1 Our Greedy

```
Input: ODG $D = (V, E)$; $W : V \to \mathbb{N}^+$; $\overline{M}, \overline{A}$
Output: $PS : V \to \mathbb{N}^+$
begin
1     $i := 1$; $V' := V$
2     while $V' \neq \emptyset$ do
3         $a$ := list of action nodes with 0 indegrees, descending order of width
4         $m$ :=list of match nodes with 0 indegrees, descending order of width
5         $w_a$ := sum of widths in $a$
6         $w_m$ := sum of widths in $m$
7         current_usage := 0
8         if $w_m \geq 1/2\overline{M}$ and $w_a \geq 1/2\overline{A}$ then
9             Go to line 12 or 19
10        if $w_a \geq 1/2\overline{A}$ and $w_m < 1/2\overline{M}$ then
11            Go to line 19
12        while $m[0]$+current_usage $\leq \overline{M}$ do
13            current_usage += $m[0]$
14            PS[$m[0]$] := $i$
15            $V' := V' \setminus \{m[0]\}$
16            $m := m - m[0]$
17        i := i+1
18        if $w_m \geq 1/2\overline{M}$ then
              continue
19        while $a[0]$+current_usage $\leq \overline{A}$ do
20            current_usage += $a[0]$
21            PS[$a[0]$] := $i$
22            $V' := V' \setminus \{a[0]\}$
23            $a := a - a[0]$
24        $i := i + 1$
25    return PS
```

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



Graph / Algorithm	Egress $\|V\| = 104$ $\|E\| = 291$	Ingress $\|V\| = 224$ $\|E\| = 930$	Combined $\|V\| = 328$ $\|E\| = 1221$
rnd_sieve i.e., [3]-greedy	13	21	30
Our greedy	13	19	23
[3] ILP	11	17	21
ILP lower bound	7	15	21

Table 2: Best P values computed by different algorithms

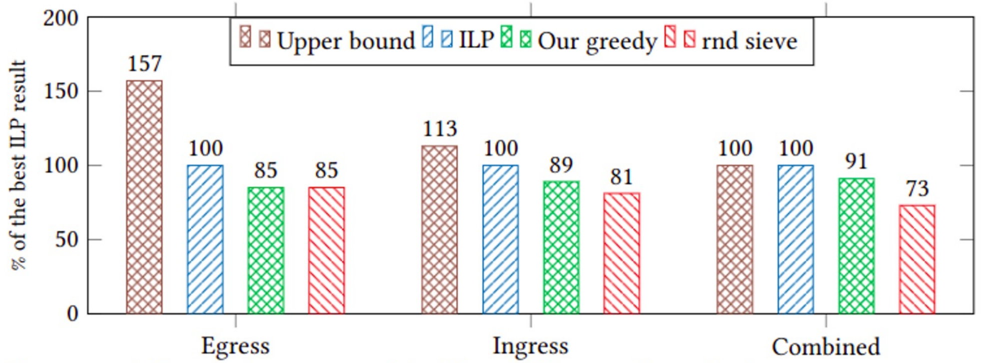


Figure 3: Throughput provided by different heuristics as precentage of the best ILP solution

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