Peeling Algorithm for Custom Instruction Identification

Kang Zhao, Jinian Bian

EDA Lab, Dept. Computer Science & Technology

Tsinghua University, Beijing 100084, China

Dec 8, 2010

Contents

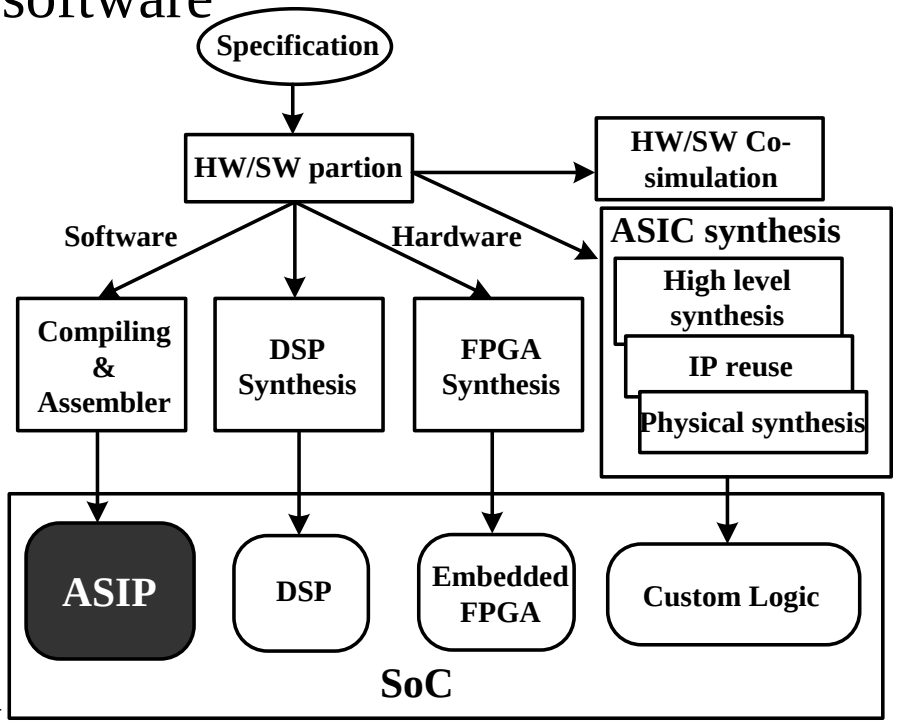
- Introduction
- Problem Formulation
- Algorithm
- Experiments
- Conclusion

Contents

- Introduction
- Problem Formulation
- Algorithm
- Experiments
- Conclusion

Introduction

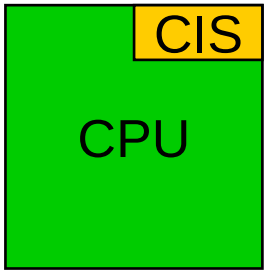
- SoC: system-on-a-chip
  - Integration of hardware and software
  - Need special technique and methodology
- ASIP
  - Application specific instruction-set processor
  - A special part in the SoC system design
  - Provide a tradeoff between efficiency and flexibility



ASIP in the Embedded system design

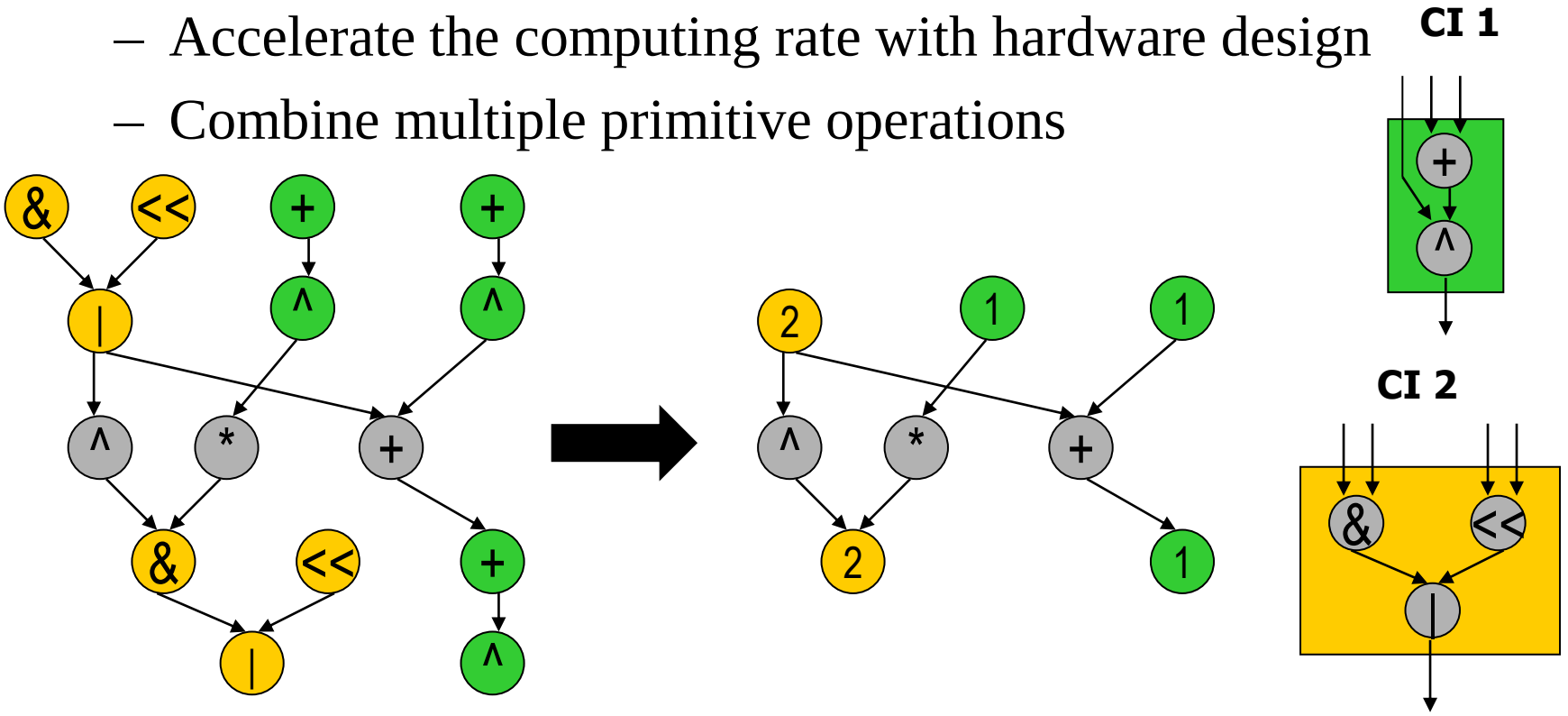
Multimedia Background

- Cell phones, digital cameras, etc. spread everywhere
  - High performance & low power consumption
- ASIP = General core + application specific instructions
  - Instruction-set extensible processor
  - CIS: custom instruction set
  - Custom instructions in ASIP can be viewed as hardware for special purpose to acceleration the processor



Acceleration through Custom Instruction

- Custom Instructions (CI)
  - Motivation: Increase the performance of processors
  - Accelerate the computing rate with hardware design
  - Combine multiple primitive operations



How to Custom Instructions

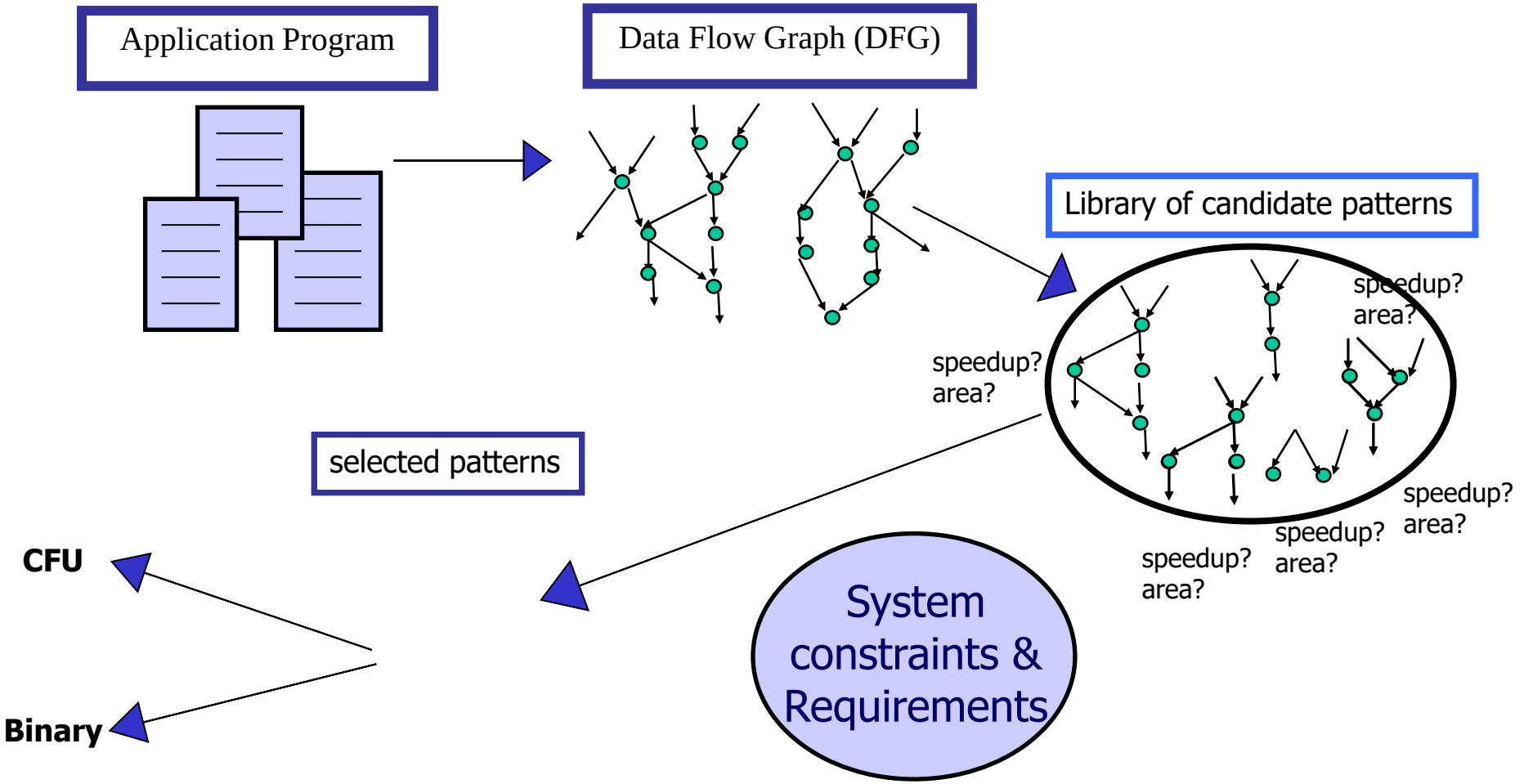
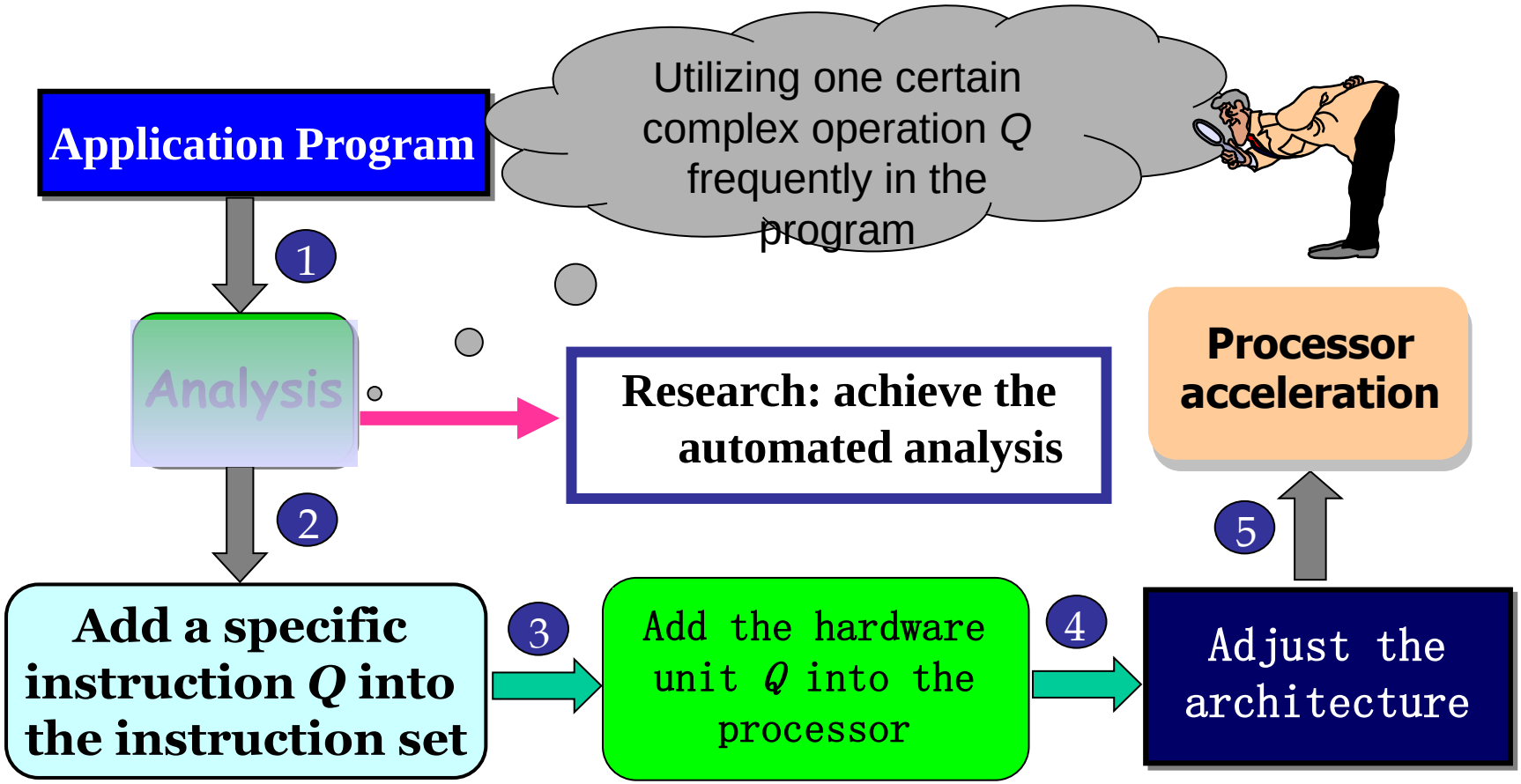


Illustration Example



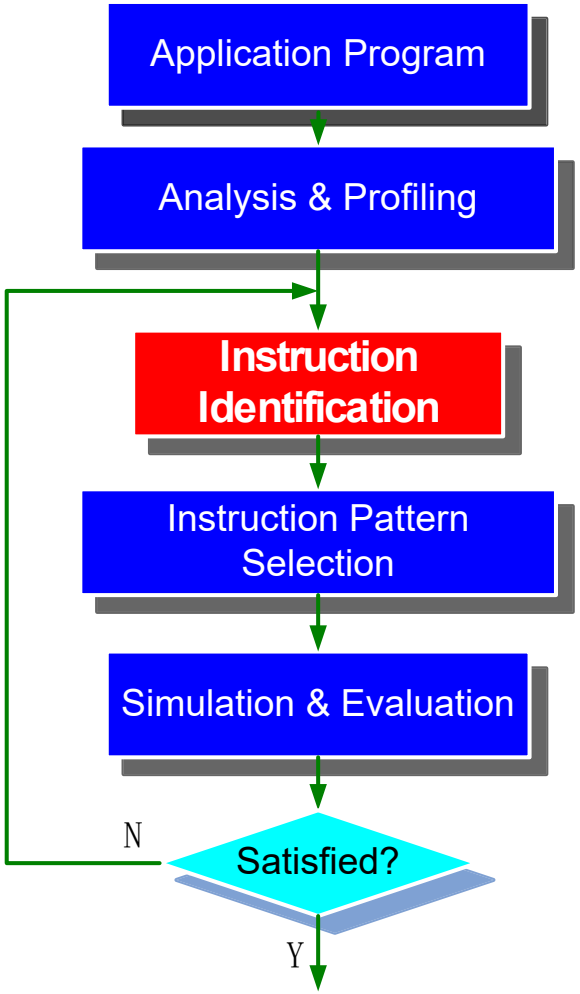
An illustration example of the purpose of our research

Contents

- Introduction
- Problem Formulation
- Algorithm
- Experiments
- Conclusion

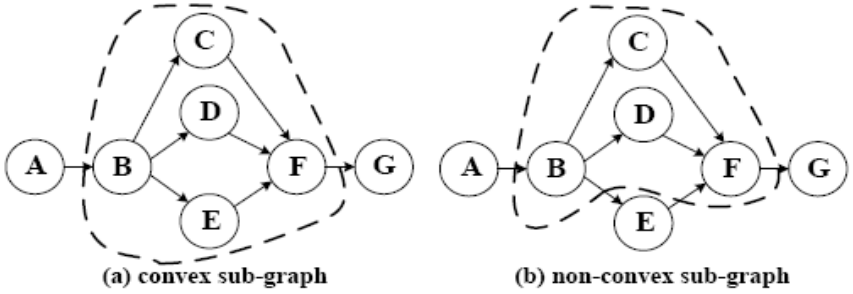
Instruction Identification

- Instruction Identification Problem
  - Input
    - Data Flow Graph (DFG)
    - Profiling results
  - Output
    - Candidate custom instructions

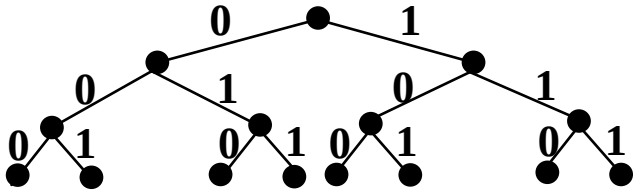


Problem Formulation

- Problem:
  - Given a directed acyclic graph $G(V, E)$, find all the feasible sub-graphs that satisfy the following two conditions:
    - The subgraph is connected
    - Only the convex sub-graph is feasible
  - Design space: exponential



Convex subgraph example



Exponential design space

Contents

- Introduction
- Problem Formulation
- Algorithm
- Experiments
- Conclusion

Strategy

- Large graph → small graph
  - Since the maximum graph must be convex, we may begin from the larger one
  - Obtain smaller subgraphs through partitioning on the large graphs
- Reduce the exploration space
  - Partition based on each node will bring repeated enumerations
  - We will use the peeling node instead of each node, to avoid the repeated enumerations

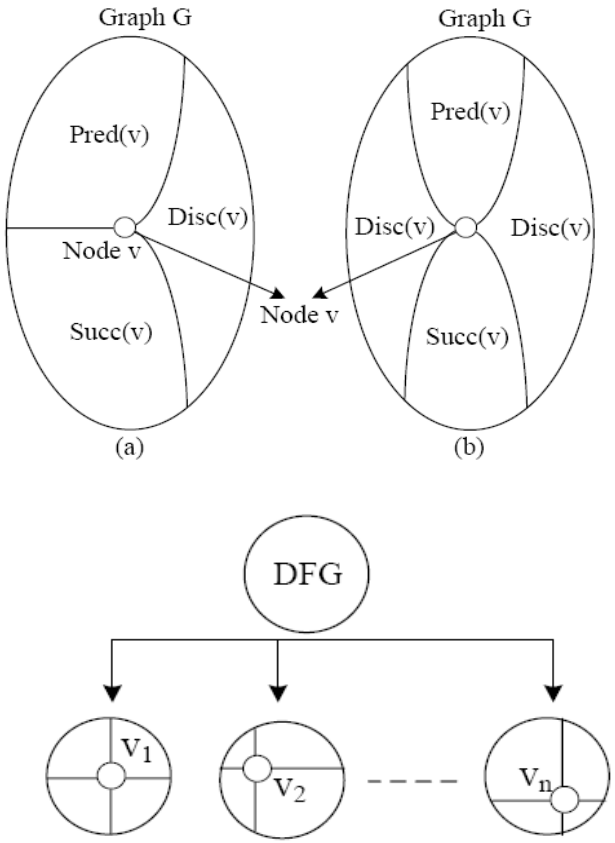


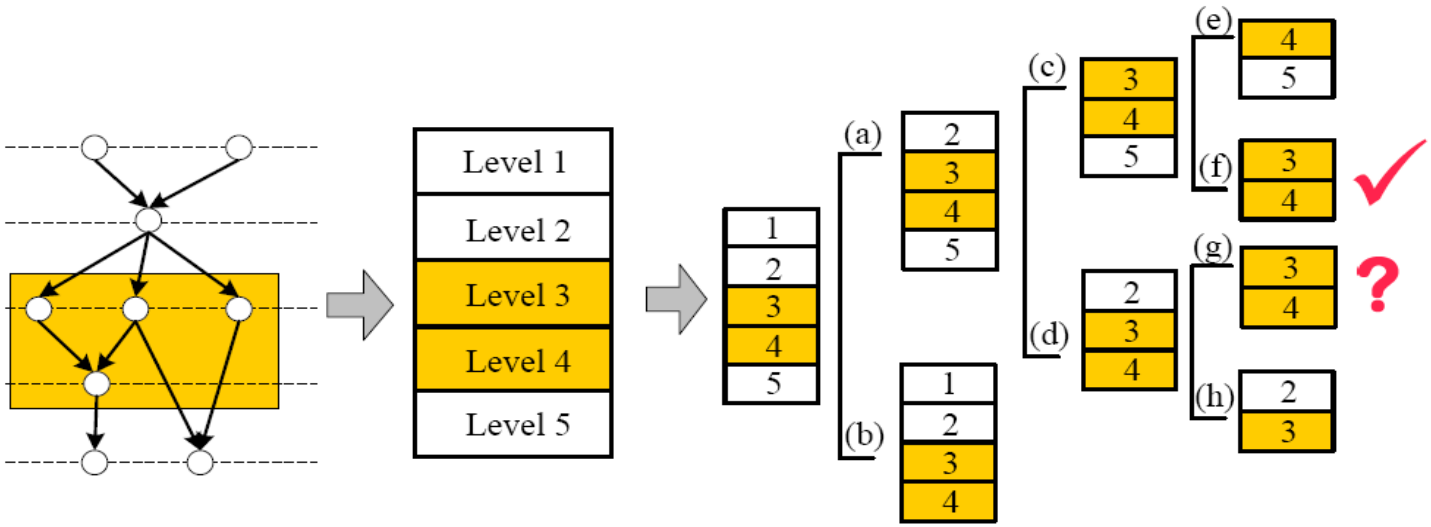
Illustration for the partitioning based on each node

Peeling node

- **Definition:** the source and sink nodes will be called as peeling nodes
  - The partitioning through deleting each peeling nodes
  - We could get the same results with the partitioning based on each node
- **Dele(G, u):** u is the peeling node of the graph G. The partition will be:
  - $$Dele(G,u) = G - \{u\}$$
    $$= \begin{cases} Succ(G,u) \cup Disc(G,u) & if\ u \in source(G) \\ Pred(G,u) \cup Disc(G,u) & if\ u \in sink(G) \end{cases}$$

Peeling process

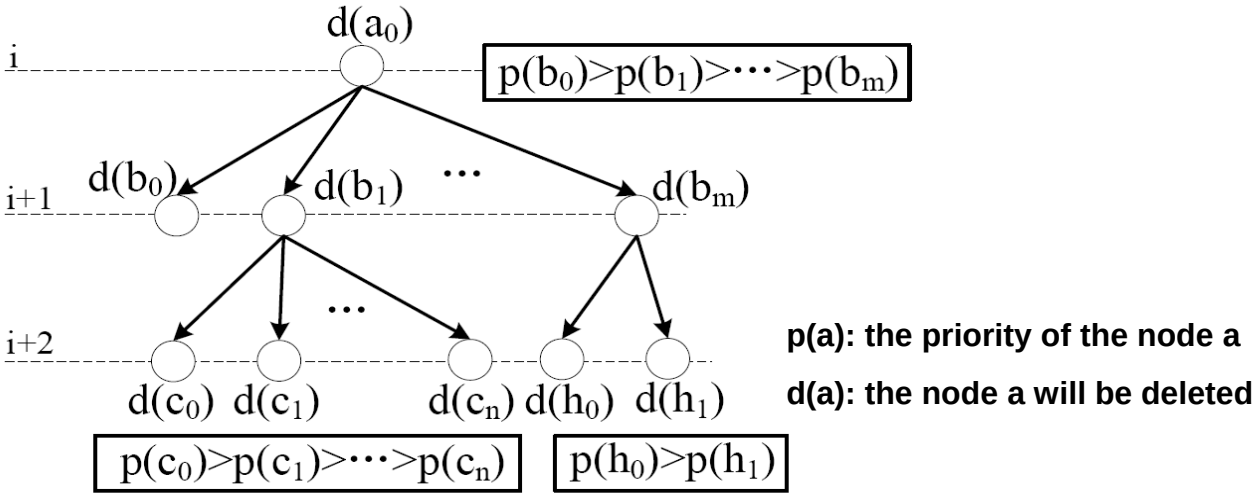
- **Theorem:** The peeling process can get the same results with the partitioning based on each node
  - However, there will be repeated enumerations



Example for the partition based on peeling nodes

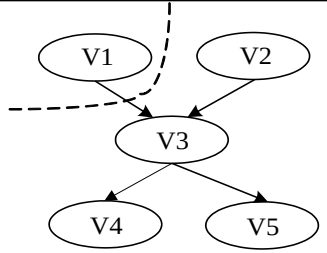
Pruning

- **Priority:** Let *u* and *v* be two peeling nodes for the pattern *G*. If *Dele(G,u)* is enumerated earlier than *Dele(G, v)*, the priority of u will be higher than *v*. This order is named as the local priority.
- **Theorem:** The local priority has three characters: 1) the partial order; 2) the local effect; 3) the genetic.



Local priority defined to avoid repeated enumeration

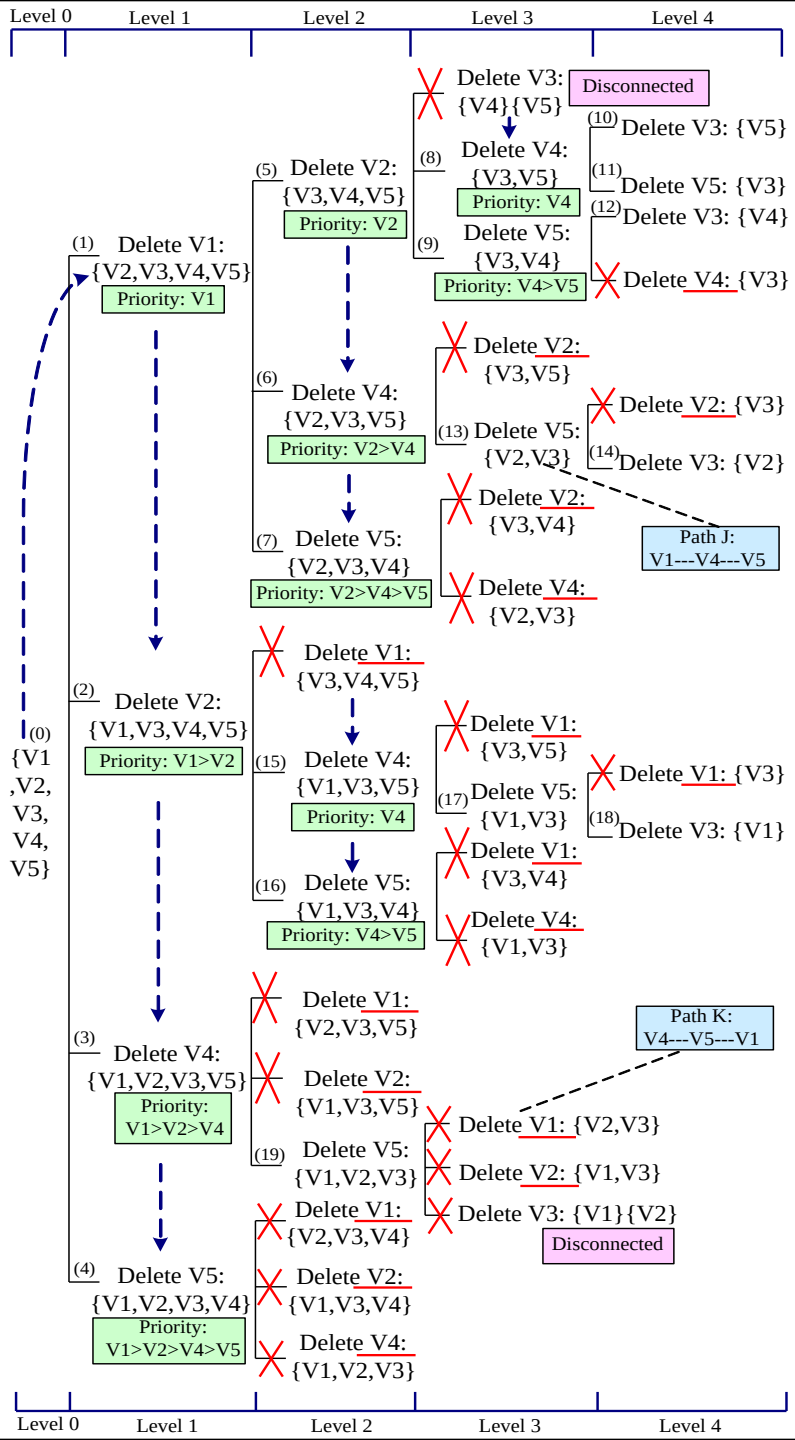
Algorithm



- (1) Get the maximal valid subgraph
- (2) Delete each peeling node
- (3) After deleting each peeling node, calculate their own priority
- (4) If the priority is violated, the partition process on this branch will stop
- (5) The iteration will stop when the minimum valid subgraph appears

partition(G, X)

```
1.  if(G = ∅) return;
2.  P ⇐ source(G) ∪ sink(G);
3.  for(each node u ∈ P)
4.    if(violate_priority(u, X)) or disconnected(G, u)
5.      P ⇐ P − {u};
6.    else
7.      patterns.add(Dele(G, u));
8.  for(each node u ∈ P)
9.    X′ ⇐ change_priority(P, u, X);
10.   partition(Dele(G, u), X′);
```



Contents

- Introduction
- Problem Formulation
- Algorithm
- Experiments
- Conclusion

Experimental Setup

- Setup
  - Program with C++ on a Linux machine
  - Intel Xeon 3GHz CPU and 4GB memory
  - OS: Red Hat Enterprise Linux AS release 3
  - Gcc 2.90 with option -03
- Benchmarks
  - Some benchmarks from MediaBench
  - DFGs come from the DOT files represented in MediaBench
- Evaluation
  - Execution time: C++ function *clock*() and the macro CLOCKS_PER_SEC

Results

- Comparison
  - The identification algorithm in IEEE Trans.CAD 2007, referred as “CH”
  - Our proposed method is referred as “PE”
- Experimental results
  - The enumeration results are shown in “pattern” column (total number of valid patterns)
  - PE is faster than CH; however, the speedup effect is not super. The reason is that they both used the predecessor and the successor to satisfy the convexity constraint.

ID	MESA Functions	Node	Edge	pattern	Identification time (s)		Speedup (×)
					CH	PE	PE/CH
1	Horner Bezier	18	16	34	0.010	0.000	11.11
2	Feedback Points	53	50	114	0.030	0.000	33.33
3	Invert Matrix	333	354	9786	122.870	1.980	62.06
4	Smooth Triangle	197	196	329	0.840	0.010	84.00
5	Matrix Multiplication	109	116	625	0.590	0.040	14.75

Contents

- Introduction
- Problem Formulation
- Algorithm
- Experiments
- Conclusion

Conclusion

- Contributions
  - We propose a peeling algorithm for the custom instruction identification, and this algorithm proposes a local priority for an exhaustive pruning process
  - The experiments indicate that the peeling algorithm can speedup the identification and can identify the custom instructions quickly.

Thanks