A Polynomial-Time Custom Instruction Identification Algorithm Based on Dynamic Programming

Junwhan Ahn, Imyong Lee, and Kiyoung Choi

Design Automation Laboratory
Seoul National University

ASP-DAC 2011

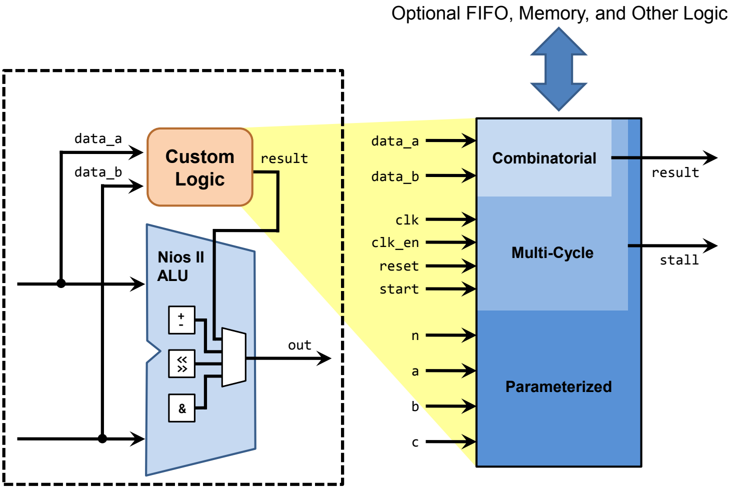
Outline

Configurable Processor

Ready-made general-purpose processor with user-defined instructions for specific applications

- More flexible and easier to design than ASICs
- Higher performance and lower power consumption than GPP

Configurable Processor



Custom Instruction Logic of Nios II Processor

Configurable Processor

Ready-made general-purpose processor with user-defined instructions for specific applications

- More flexible and easier to design than ASICs
- Higher performance and lower power consumption than GPP

Problem: How can we decide what to implement?

Custom Instruction Identification

Question

What is the best custom instruction for a specific application under the given architectural constraints?

Custom Instruction Identification

Question

What is the best custom instruction for a specific application under the given architectural constraints?

Best

- Performance improvement

Custom Instruction Identification

Question

What is the best custom instruction for a specific application under the given architectural constraints?

Best

- Performance improvement

Architectural constraints

- The number of inputs and outputs
- Types of operations that can be used in the custom functional unit

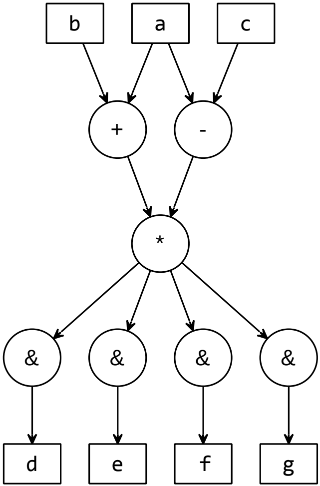
Custom Instruction Identification

```
b = x + 1;
c = x - 1;
for (a = 0; a < 20; ++a) {
  t = a + b;
  u = a - c;
  d = (t * u) & 0x000f;
  e = (t * u) & 0x00f0;
  f = (t * u) & 0x0f00;
  g = (t * u) & 0xf000;
}
y = d + e + f + g;
```

CI Identification Flow

1. Select a kernel
2. Convert it into a DFG
3. Find the best CI

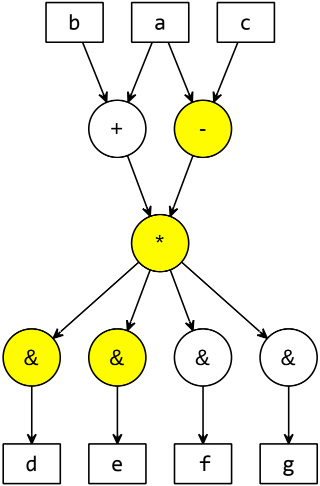
Custom Instruction Identification



CI Identification Flow

1. Select a kernel
2. Convert it into a DFG
3. Find the best CI

Custom Instruction Identification



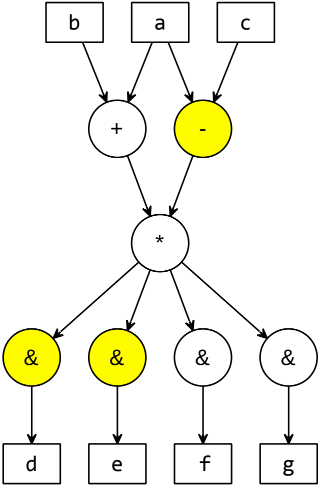
CI Identification Flow

1. Select a kernel
2. Convert it into a DFG
3. Find the best CI

Basic properties of a cut:

- Inputs and outputs
- Convexity

Custom Instruction Identification



CI Identification Flow

1. Select a kernel
2. Convert it into a DFG
3. Find the best CI

Basic properties of a cut:

- Inputs and outputs
- Convexity

Custom Instruction Identification

Formal definition of the problem is as follows:

Single Custom Instruction Identification

Given a graph G, find a convex cut S that maximizes $\mathrm{M}(S)$ under the constraints $|\mathrm{IN}(S)| \leq N_{\mathrm{in}}$ and $|\mathrm{OUT}(S)| \leq N_{\mathrm{out}}$.

- G: a DAG which denotes the data flow of a basic block
- S: a subgraph of G
- $\mathrm{IN}(S)$, $\mathrm{OUT}(S)$: inputs and outputs of a cut S
- $\mathrm{M}(S)$: user-defined function for evaluating a cut

Previous Works

Optimal solution with exponential time complexity

- Branch and bound with pruning (K.Atasu, DAC'03)
- Integer linear programming (K.Atasu, CODES+ISSS'05)

Previous Works

Optimal solution with exponential time complexity

- Branch and bound with pruning (K.Atasu, DAC'03)
- Integer linear programming (K.Atasu, CODES+ISSS'05)

Problems

- Too much time is needed for large dataflow graphs.

Previous Works

Nonoptimal solution with lower time complexity

- Genetic algorithm (L.Pozzi, TCAD'06)
- Partitioning-based selection (L.Pozzi, TCAD'06)
- Cone union algorithm (P.Yu, DAC'04)

Previous Works

Nonoptimal solution with lower time complexity

- Genetic algorithm (L.Pozzi, TCAD'06)
- Partitioning-based selection (L.Pozzi, TCAD'06)
- Cone union algorithm (P.Yu, DAC'04)

Problems

- The solution may not be optimal.

Previous Works

Nonoptimal solution with lower time complexity

- Genetic algorithm (L.Pozzi, TCAD'06)
- Partitioning-based selection (L.Pozzi, TCAD'06)
- Cone union algorithm (P.Yu, DAC'04)

Problems

- The solution may not be optimal.
- Only connected cuts can be selected where disconnected cuts give better performance improvement in general.

Our Approach

Top-down dynamic programming with single cut identification algorithm (branch and bound with pruning)

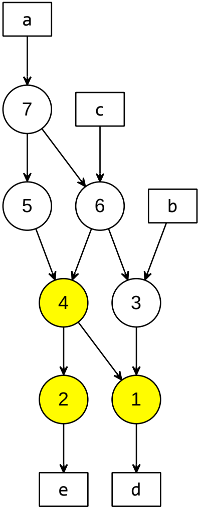
- Solutions of subproblems are stored in a memoization table
- The stored solution is used instead of enumeration if avaliable

But, how can we guarantee that constraints are still satisfied even if we use stored solutions?

Our Approach

Problem

How can we know that the cut is no longer convex if we add the vertex 7 into the cut?



Our Approach

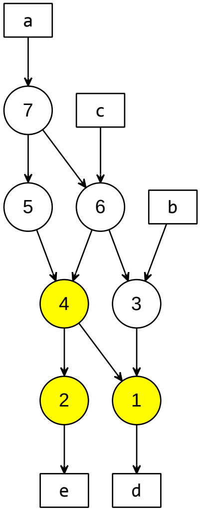
Problem

How can we know that the cut is no longer convex if we add the vertex 7 into the cut?

Solution

Convexity constraint is violated when we add a vertex that has a path to an input which

1. is a follower vertex of S, or
2. has at least one includable ancestor that is a follower vertex of S.



Our Approach

Problem

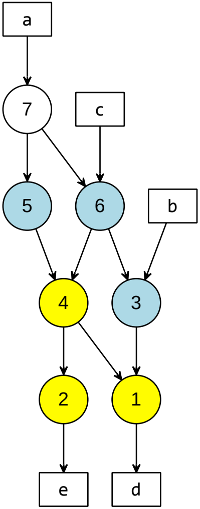
How can we know that the cut is no longer convex if we add the vertex 7 into the cut?

Solution

Convexity constraint is violated when we add a vertex that has a path to an input which

1. is a follower vertex of S, or
2. has at least one includable ancestor that is a follower vertex of S.

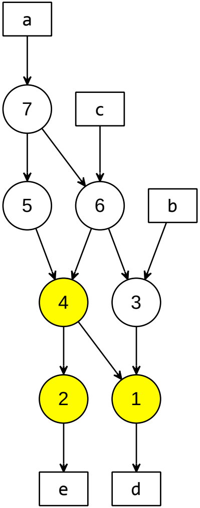
We call such inputs as **watcher inputs**.



Our Approach

Problem

How can we calculate $|\mathrm{IN}(S)|$ of the cut $S' = S \cup \{v\}$ with properties of S?



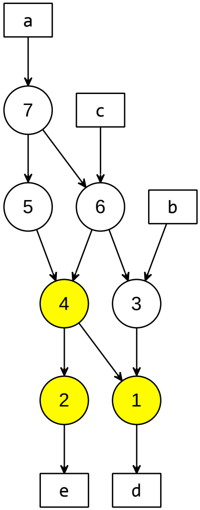
Our Approach

Problem

How can we calculate $|\mathrm{IN}(S)|$ of the cut $S' = S \cup \{v\}$ with properties of S?

Solution

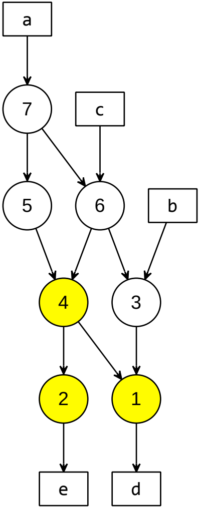
$$|\mathrm{IN}(S)| = |\mathrm{IN}_{\mathrm{w}}(S)| + |\mathrm{IN}_{\mathrm{nw}}(S)|$$



Our Approach

Problem

How can we calculate $|\text{OUT}(S)|$ of the cut $S' = S \cup \{v\}$ with properties of S?



Our Approach

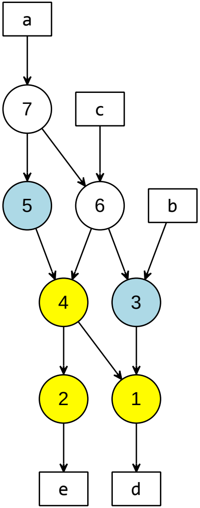
Problem

How can we calculate $|\mathrm{OUT}(S)|$ of the cut $S' = S \cup \{v\}$ with properties of S?

Solution

v is an output if and only if it has at least one outside successor. We call an input with no outside successor as a **dedicated input**. Therefore,

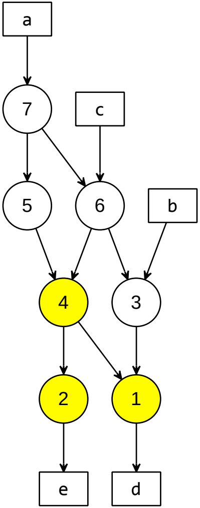
$$|\mathrm{OUT}(S')| = \begin{cases} |\mathrm{OUT}(S)| + 1 & v \notin \mathrm{IN_d}(S) \\ |\mathrm{OUT}(S)| & \text{otherwise} \end{cases}$$



Our Approach

Problem

How can we calculate dedicated inputs of the cut $S' = S \cup \{v\}$ with properties of S?



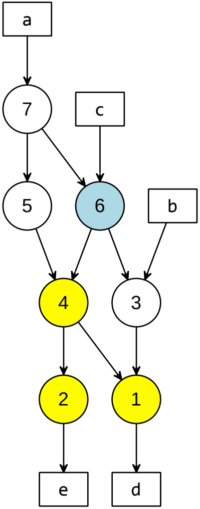
Our Approach

Problem

How can we calculate dedicated inputs of the cut $S' = S \cup \{v\}$ with properties of S?

Solution

When the last successor of an input is added to the cut, the input becomes dedicated unless it already has at least one outside successor (**permanently undedicated inputs**).



Our Approach

In short, I/O and convexity constraints can be checked with following properties when we inserting a follower vertex v to the convex cut S. (a **representative tuple**)

$$t(S) = (\mathrm{IN_w}(S), |\mathrm{IN_{nw}}(S)|, \mathrm{IN_{pu}}(S), \mathrm{IN_d}(S), |\mathrm{OUT}(S)|, v_{\mathrm{last}}(S))$$

Moreover, we found a function F for the following relation:

$$t(S \cup \{v\}) = F(t(S))$$

Our Approach

Therefore, we can conclude the following.

Corollary

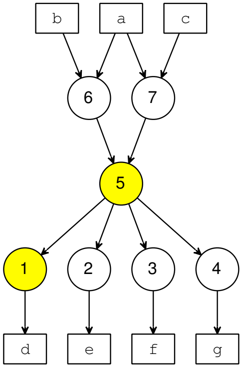
When constructing a bigger convex cut $S' = S \cup \{v\}$ *from a convex cut* S *by adding a vertex in the pre-determined traversal order, the constraint on the number of inputs, outputs, and convexity of* S' *can be fully determined by only the following properties:* $\mathrm{IN_w}(S)$, $|\mathrm{IN_{nw}}(S)|$, $\mathrm{IN_{pu}}(S)$, $\mathrm{IN_d}(S)$, $|\mathrm{OUT}(S)|$, *and* $v_{\mathrm{last}}(S)$.

Thus, we can safely use stored solutions for all cuts with same representative tuple.

Our Approach

For $S_1 = \{1, 5\}$,

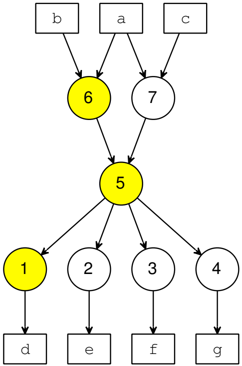
$$\mathrm{IN_w}(S_1) = \{6, 7\} \quad |\mathrm{IN_{nw}}(S_1)| = 0$$
$$\mathrm{IN_d}(S_1) = \{6, 7\} \quad \mathrm{IN_{pu}}(S_1) = \varnothing$$
$$|\mathrm{OUT}(S_1)| = 2 \quad v_{\mathrm{last}}(S_1) = 5$$



Our Approach

For $S_1 = \{1, 5\}$,

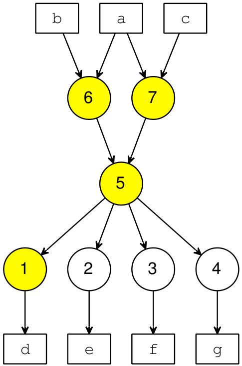
$$\mathrm{IN_w}(S_1) = \{6, 7\} \quad |\mathrm{IN_{nw}}(S_1)| = 0$$
$$\mathrm{IN_d}(S_1) = \{6, 7\} \quad \mathrm{IN_{pu}}(S_1) = \varnothing$$
$$|\mathrm{OUT}(S_1)| = 2 \quad v_{\mathrm{last}}(S_1) = 5$$



Our Approach

For $S_1 = \{1, 5\}$,

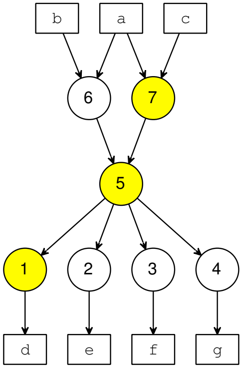
$$\mathrm{IN_w}(S_1) = \{6, 7\} \quad |\mathrm{IN_{nw}}(S_1)| = 0$$
$$\mathrm{IN_d}(S_1) = \{6, 7\} \quad \mathrm{IN_{pu}}(S_1) = \varnothing$$
$$|\mathrm{OUT}(S_1)| = 2 \quad v_{\mathrm{last}}(S_1) = 5$$



Our Approach

For $S_1 = \{1, 5\}$,

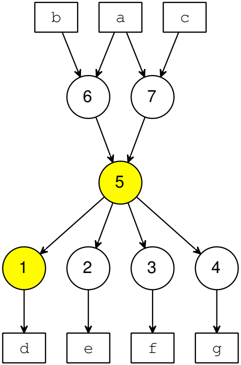
$$\mathrm{IN_w}(S_1) = \{6, 7\} \quad |\mathrm{IN_{nw}}(S_1)| = 0$$
$$\mathrm{IN_d}(S_1) = \{6, 7\} \quad \mathrm{IN_{pu}}(S_1) = \varnothing$$
$$|\mathrm{OUT}(S_1)| = 2 \quad v_{\mathrm{last}}(S_1) = 5$$



Our Approach

For $S_1 = \{1, 5\}$,

$$\mathrm{IN_w}(S_1) = \{6, 7\} \quad |\mathrm{IN_{nw}}(S_1)| = 0$$
$$\mathrm{IN_d}(S_1) = \{6, 7\} \quad \mathrm{IN_{pu}}(S_1) = \varnothing$$
$$|\mathrm{OUT}(S_1)| = 2 \quad v_{\mathrm{last}}(S_1) = 5$$



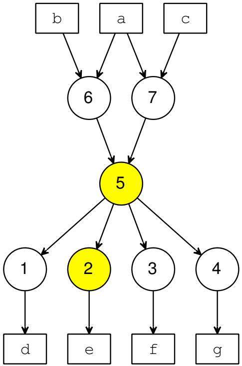
If the best cut is $S_1' = \{1, 5, 6, 7\}$, store $S_1' - S_1$ into the memoization table with a key k, where

$$k = (\mathrm{IN_w}(S_1), |\mathrm{IN_{nw}}(S_1)|, \mathrm{IN_d}(S_1), \mathrm{IN_{pu}}(S_1), |\mathrm{OUT}(S_1)|, v_{\mathrm{last}}(S_1))$$

Our Approach

For $S_2 = \{2, 5\}$,

$$\mathrm{IN_w}(S_1) = \{6, 7\} \quad |\mathrm{IN_{nw}}(S_1)| = 0$$
$$\mathrm{IN_d}(S_1) = \{6, 7\} \quad \mathrm{IN_{pu}}(S_1) = \varnothing$$
$$|\mathrm{OUT}(S_1)| = 2 \quad v_{\mathrm{last}}(S_1) = 5$$



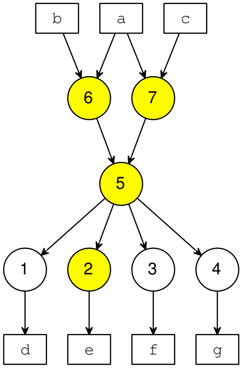
Our Approach

For $S_2 = \{2, 5\}$,

$$\begin{aligned} \text{IN}_w(S_1) &= \{6, 7\} & |\text{IN}_{nw}(S_1)| &= 0 \\ \text{IN}_d(S_1) &= \{6, 7\} & \text{IN}_{pu}(S_1) &= \varnothing \\ |\text{OUT}(S_1)| &= 2 & v_{last}(S_1) &= 5 \end{aligned}$$

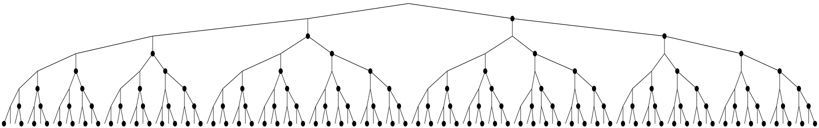
We can use the stored solution from S_1 due to the corollary. Therefore,

$$S_2' = S_2 \cup \{6, 7\} = \{2, 5, 6, 7\}$$



Our Approach

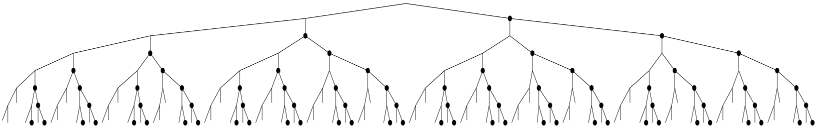
Branch-and-bound search can be visualized as a binary tree.



1. Total search space

Our Approach

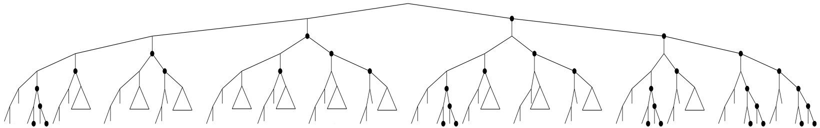
Branch-and-bound search can be visualized as a binary tree.



1. Total search space
2. Pruned by K. Atasu's algorithm (with $N_{\text{in}} = 8$, $N_{\text{out}} = 4$)

Our Approach

Branch-and-bound search can be visualized as a binary tree.



1. Total search space
2. Pruned by K. Atasu's algorithm (with $N_{\text{in}} = 8$, $N_{\text{out}} = 4$)
3. Used memoized solutions (our approach)

Our Approach

How about optimality?

Theorem

The single cut identification problem has optimal substructure if $\mathrm{M}(S \cup S') = \mathrm{M}(S) + \mathrm{M}(S')$.

Therefore, the algorithm is optimal for $\mathrm{M}(S) = \sum_{v \in S} s_v$.
However,

$$\mathrm{M}(S) = \sum_{v \in S} s_v - \lceil L \rceil$$

is used in general. In this case, the proposed algorithm may give an approximated solution.

Our Approach

The upper bound for the size of the memoization table:

$$O\left(|V|^{N_{\text{in}}} \cdot (N_{\text{in}}^{N_{\text{in}}})^2 \times N_{\text{in}} \times |V|^2 \times N_{\text{out}}\right)$$

The time complexity for processing each item:

$$O\left(N_{\text{in}} + |V| + 1\right)$$

Therefore, the overall time complexity of our algorithm is

$$O\left(|V|^{N_{\text{in}}+3} \cdot N_{\text{in}}^{3N_{\text{in}}} \cdot N_{\text{out}}\right)$$

which is polynomial to the number of vertices.

Our Approach

Therefore, the overall time complexity of our algorithm is

$$O\left(|V|^{N_{\text{in}}+3} \cdot N_{\text{in}}^{3N_{\text{in}}} \cdot N_{\text{out}}\right)$$

which is polynomial to the number of vertices.

Note that two previous optimal algorithms have $O\left(2^{|V|}\right)$ time complexity.

Experiments

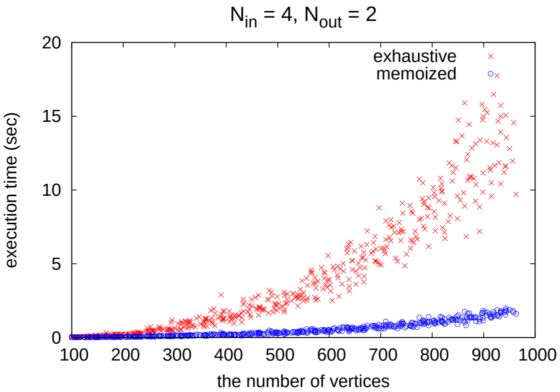
We compared the execution time of the proposed algorithm (**memoized**) to that of the K.Atasu's single cut identification algorithm (**exhaustive**).

Settings

- Merit function is defined as a difference between software and hardware latency.
- The size of the memoization table is limited to one million items. (consumes roughly 500MB memory)

Synthetic Graphs

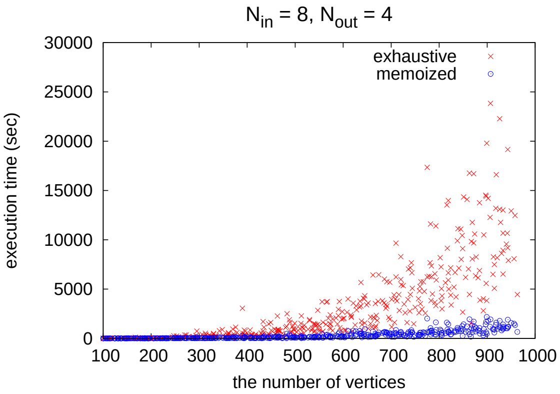
The following result is for randomly generated dataflow graphs with 87 to 963 vertices.



All solutions obtained by **memoized** were optimal.

Synthetic Graphs

The following result is for randomly generated dataflow graphs with 87 to 963 vertices.



All solutions obtained by **memoized** were optimal.

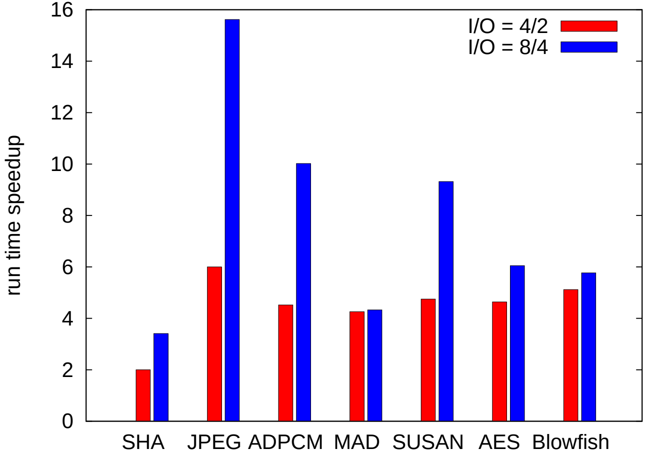
Real World Applications from MiBench

Name	$\|V\|$	I/O	**exhaustive**	**memoized**	Speedup	d_{SW} †	d_{CI} †
SHA	38	4/2 8/4	0.032s 3.696s	0.016s 1.083s	2.00 3.41	50	37 13
JPEG*	92	4/2 8/4	0.120s 69.67s	0.020s 4.460s	6.00 15.62	164	144 131
ADPCM*	133	4/2 8/4	0.411s 89.71s	0.091s 8.952s	4.52 10.02	220	193 152
MAD	137	4/2 8/4	1.125s 311.5s	0.264s 71.91s	4.26 4.33	337	326 304
SUSAN	197	4/2 8/4	0.261s 216.8s	0.055s 23.26s	4.75 9.32	525	515 503
AES	247	4/2 8/4	1.271s 183.0min	0.274s 30.24min	4.64 6.05	431	414 392
Blowfish	414	4/2 8/4	1.433s 71.24min	0.280s 12.35min	5.12 5.77	219	212 185

* disconnected graph

† execution cycle of the basic block without (d_{SW}) and with (d_{CI}) a custom instruction

Real World Applications from MiBench



Conclusion

We proposed a polynomial-time algorithm for custom instruction identification.

- The correctness of the algorithm is proved theoretically.
- The algorithm gives an optimal solution for generally used merit function with very high probability.
- The algorithm is significantly faster than the previous approach.