Joint Variable Partitioning and Bank Selection Instruction Optimization on Embedded Systems with Multiple Memory Banks

Liu Tiantian, Minming Li, Chun Jason Xue
City University of Hong Kong
2010.1.19

香港城市大學
City University of Hong Kong

Outline of this talk

- Background & Related work
- What the problem is? Variable partitioning & BSL insertion
- How to solve the problem?
  - For speed
  - For space
- Experimental results

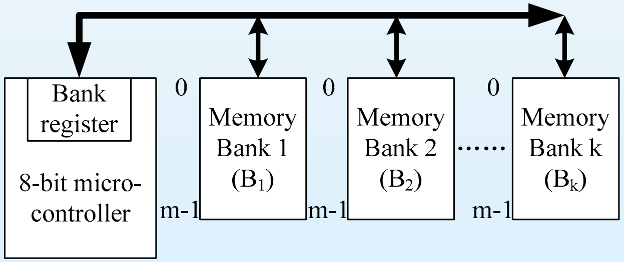
Background

8-bit microcontrollers

- About 55% of all CPUs sold in the world are 8-bit microcontrollers
- Can access limited memory with few buses and smaller address registers

Partitioned memory architecture

- Zilog Z80 and MOS6502 series:
  - 16 bit address registers can only address a maximum of 64 KB memory
  - => support more than 64 KB memory by partitioning it into banks



Bank Switching:

- Only can access one bank at a time, ***bank register***.
- ***Bank selection instructions (BSLs)***: instructions needed to be inserted into the original programs to modify the bank register to point to the wanted bank.
- Their insertions increase both ***code size and runtime overhead***.

What we focus on?

- to minimize the size and time overhead introduced by BSLs.
  - Speed overhead (means runtime increase) minimization
  - Space overhead (means code size increase) minimization
- Variable partitioning
  - Current techniques aim at achieving the maximum instruction level parallelism for architectures which allow multiple banks to be accessed simultaneously
- BSL insertion
  - Current compilers provide limited support to generate bank switching code optimally.
- The two process are related to each other and affect the overhead.

Related Work

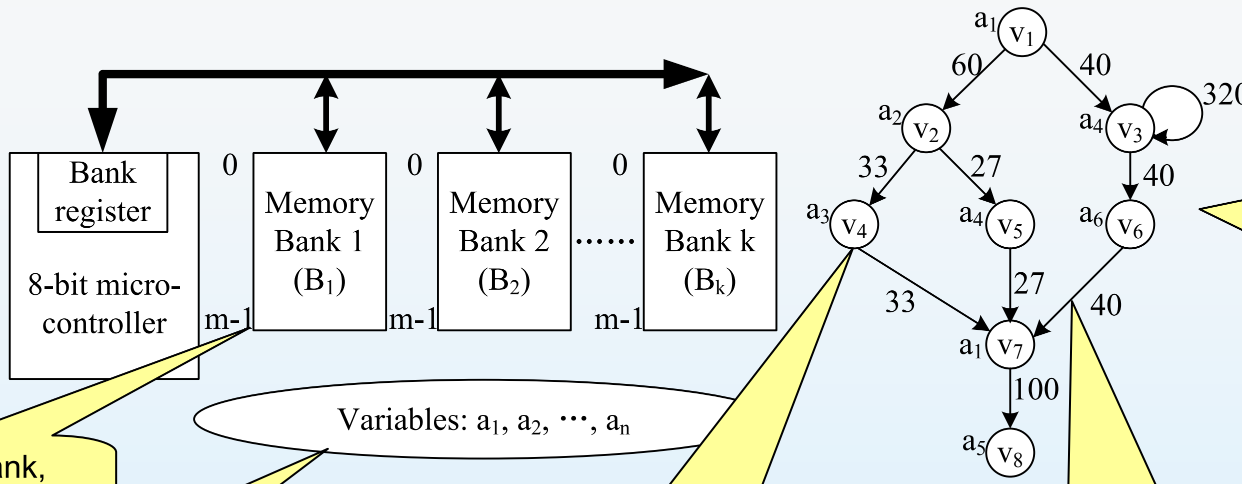
- BSL minimization
  - Bernhard Scholz, Bernd Burgstaller, Jingling Xue, "Minimal Placement of Bank Selection Instructions for Partitioned Memory Architectures," *ACM Transactions on Embedded Computing Systems (TECS), Vol. 7, No. 2: 1-32, 2008*.
  - Bernhard Scholz, Bernd Burgstaller, Jingling Xue, "Minimizing Bank Selection Instructions for Partitioned Memory Architectures," *CASES06: 201-211, 2006*.
    - assume the variables have already been assigned to memory banks
    - present an optimization technique that minimizes the overhead of BSLs.
  - Yuan Mengting, Wu Guoqing, Yu Chao, "Optimizing Bank Selection Instructions by Using Shared Memory," *ICESS2008: 447-450, 2008*.
    - assume the variables have already been partitioned into banks.
    - consider using the architecture with a shared memory bank
- Variable partitioning: most aim at achieving the maximum instruction level parallelism for architectures which allow multiple banks to be accessed simultaneously.
  - transform the variable partitioning problem to an Interference Graph (IG) or Variable Independence Graph (VIG) to find the parallelism between variables
  - combined with instruction scheduling problem for multi-core DSP architectures
  - optimizing energy consumption: try to determine when to power some banks down

- There is no variable partitioning techniques for BSL overhead minimization which is important for embedded systems with multiple memory banks.

Outline of this talk

- Background & Related work
- What the problem is? Variable partitioning & BSL insertion
- How to solve the problem?
  - For speed
  - For space
- Experimental results

What the problem is? Variable partitioning



a) Multi-bank architecture

(b) Application m G

B_i: a memory bank, size of m

a_i: a variable, size of 1
$B(a_i)$: the memory bank storing a_i

v_i: a node, is a basic data accessing block
$A(v_i)$: the variable accessed by v_i

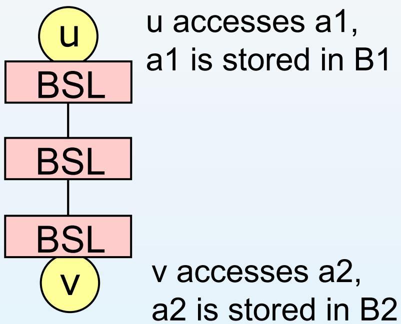
$e(u, v)$: an edge, the control flow of code
$w(e(u, v))$: the execution frequency of $e(u, v)$

DFG: Data Flow Graph, to model an embedded application

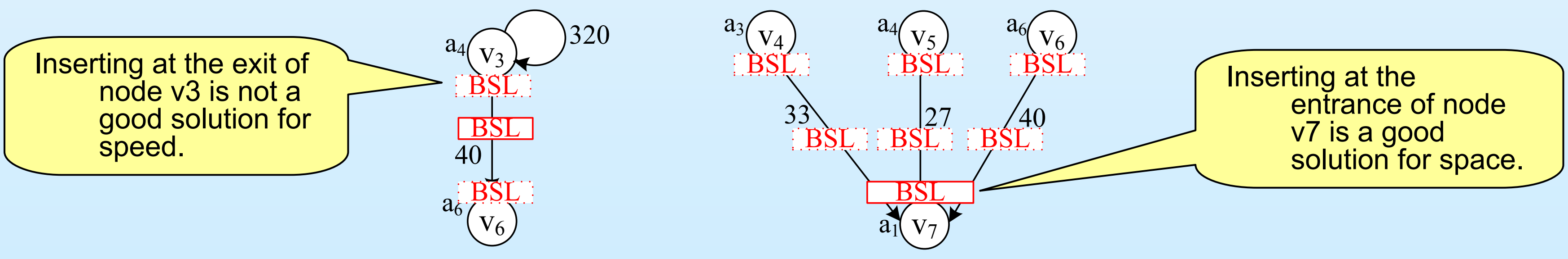
- In the system, we have in total:
  - k banks: B_1 ,… B_k , each of size m
  - n variables: v_1 ,… v_n , each of size 1
  - $k*m>=n$
- The *n* variables need to be partitioned into the *k* banks

What the problem is? BSL Insertion

- When a node *u* and one of its children *v* access variables that are in different banks, a BSL needs to be inserted.
- There are three possible positions to insert this BSL:
  - at the exit of node u;
  - at edge e(u, v);
  - at the entrance of node v.



- The three positions have **different impacts on speed and space** overheads.

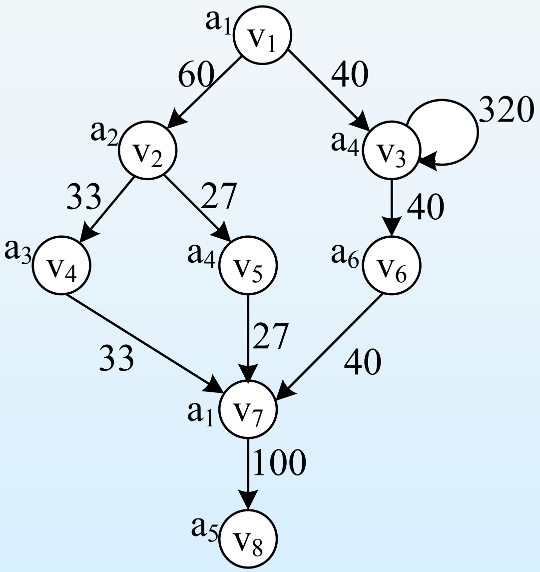


(a) For Speed (b) For Space

Motivational Example

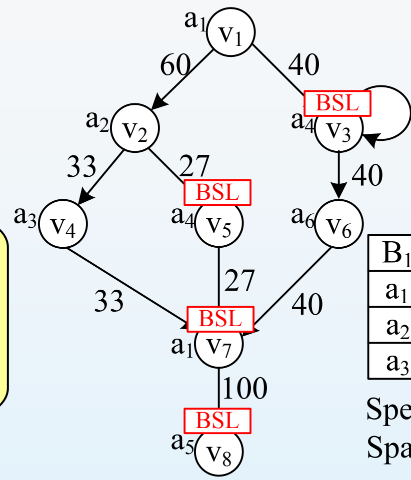
- Assume we have a dual-bank architecture and each bank has a size of three.

- The variable partitioning techniques under comparison are:
  - Equally partitioning according to reference order
  - Partitioning for parallelism [8]
  - The proposed algorithms for speed
  - The proposed algorithms for space



Motivational Example-Results

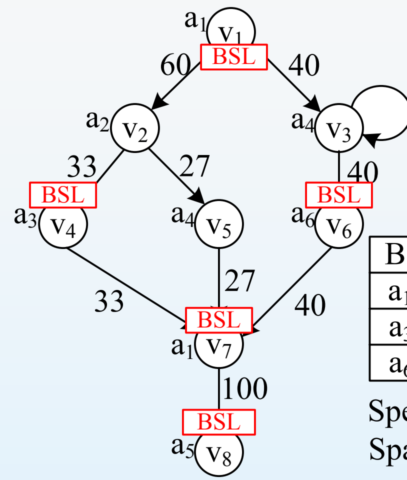
Even for such a small example, we can achieve significant improvements!



B_1	B_2
a_1	a_4
a_2	a_5
a_3	a_6

Speed: 234
Space: 4

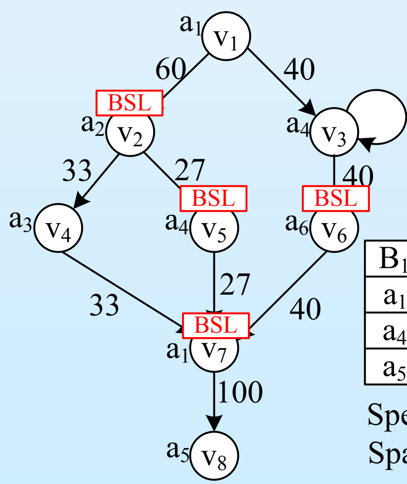
(a) Partition equally according to reference order.



B_1	B_2
a_1	a_2
a_3	a_4
a_6	a_5

Speed: 300
Space: 5

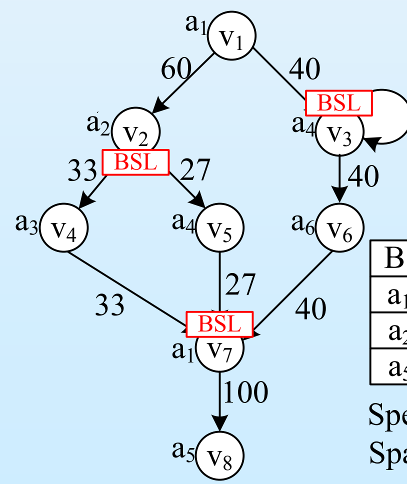
(b) Partition using method in [8].



B_1	B_2
a_1	a_2
a_4	a_3
a_5	a_6

Speed: 200
Space: 4

(c) Partition using algorithm for speed.



B_1	B_2
a_1	a_3
a_2	a_4
a_5	a_6

Speed: 200
Space: 3

(d) Partition using algorithm for space.

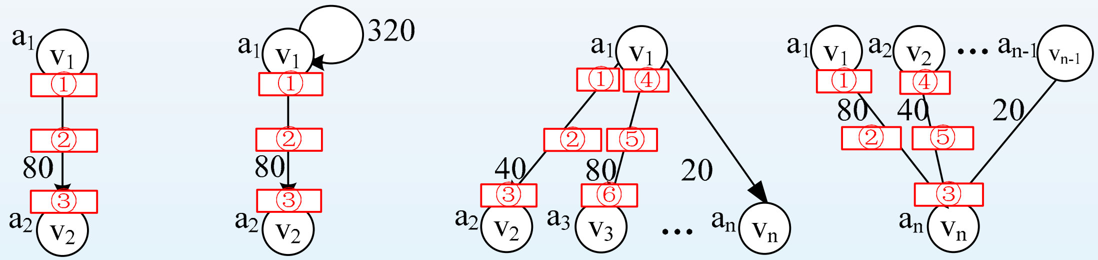
The proposed algorithm for space also offers an optimal solution for speed of 200 BSL-cycles

Outline of this talk

- Background & Related work
- What the problem is? Variable partitioning & BSL insertion
- How to solve the problem?
  - For speed
  - For space
- Experimental results

For Speed-patterns

- Need to **consider the execution cycles of inserted BSLs**.
  - One insertion of BSL may lead to many executions cycles in runtime.
- Four basic kinds of parent-child patterns



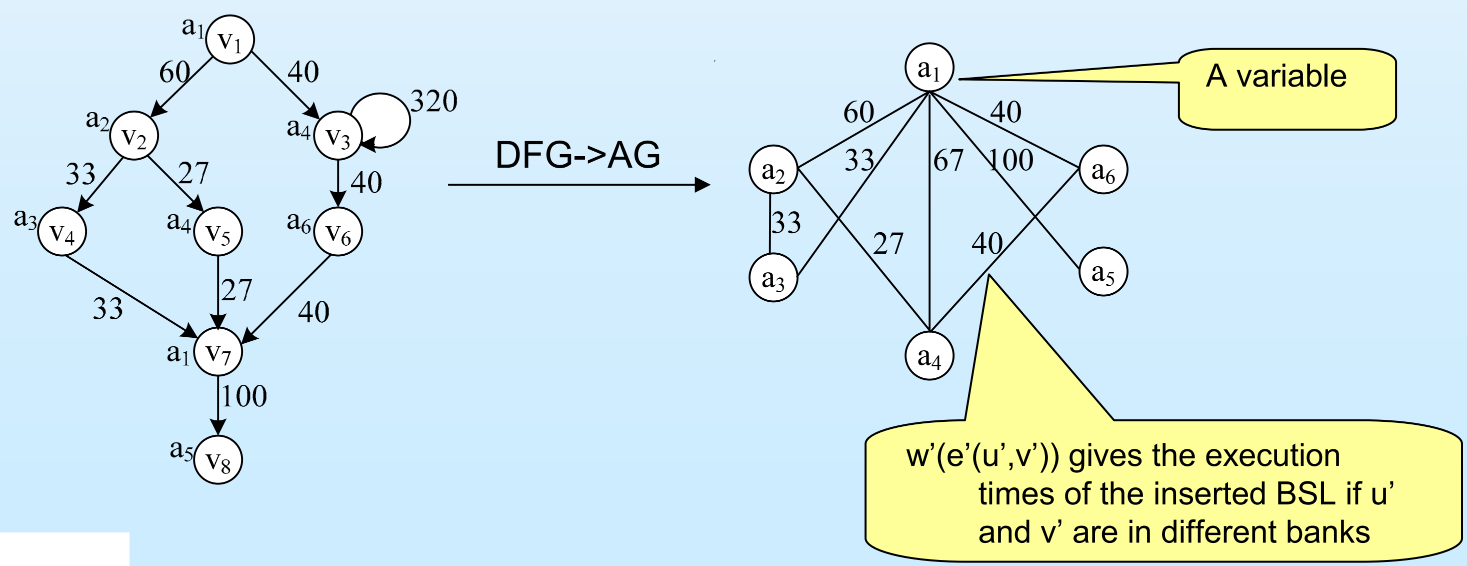
(a) One to one (b) One to one with self-loops (c) One to many (d) Many to one

  - Pattern (a). One to one: inserting one BSL at three positions are the same.
  - Pattern (b). One to one with self-loops: inserting one BSL at position ② or ③ is better than ①. **Never inserting BSLs at nodes with self-loop**.
  - Pattern (c). One to many: No matter which positions BSLs are inserted at, they will be executed in total $\sum_{i\in[2,n]} w(e(v_1, v_i))$ BSL-cycles.
  - Pattern (d). Many to one: No matter which positions BSLs are inserted at, they will be executed in total $\sum_{i\in[1,n-1]} w(e(v_i, v_n))$ BSL-cycles.
- Conclusion: after partitioning, always **inserting BSLs at edges when needed**.

For Speed-Algorithm(1)

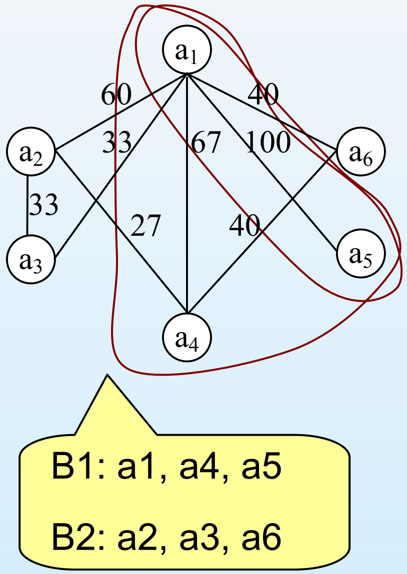
- An **Accessing Graph (AG)** G'(V', E)' is constructed to present the adjacent accessing relationships between variables.
  - an undirected graph
  - v': a node represents a unique variable
  - e'(u', v'): an edge represents the adjacent accessing relationship between the two variables u'and v'.
    - w'(e'(u', v')) $= \sum w(e(u,v))$, where

$$e(u,v) \in E_{DFG} \;\&\; ((A(u) = u' \;\&\; A(v) = v') \;||\; (A(u) = v' \;\&\; A(v) = u'))$$



For Speed-Algorithm(1)

- The problem becomes: we want to partition the n nodes into k parts so that the size of each part does not exceed m and the total weight of the cross edges between parts are minimized.
  - k-balanced partitioning problem
  - proved to be an NP-Hard problem
  - some theoretical works about this problem



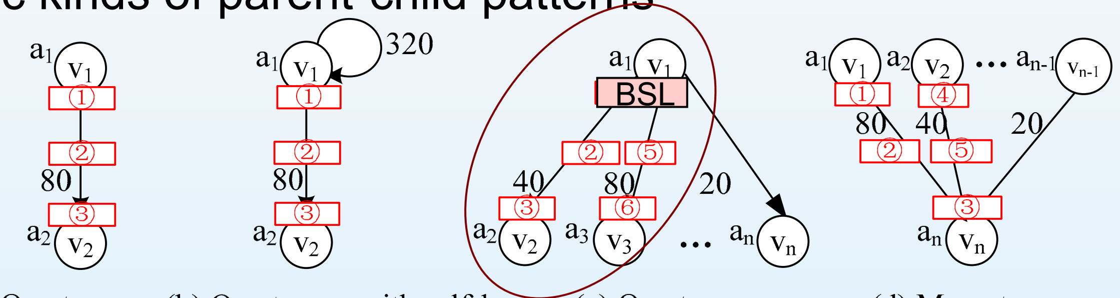
- A polynomial heuristic algorithm: greedy
  - Combine two variables whose combination can maximize the speed saving in each iteration.
  - The complexity is $O(n^2 \log n)$.

Outline of this talk

- Background & Related work
- What the problem is? Variable partitioning & BSL insertion
- How to solve the problem?
  - For speed
  - For space
- Experimental results

For Space-patterns

- Only care about the total number of BSLs inserted into the original code
  - upper bound : |V|-1 (inserting at entrance when needed except root).
- Four basic kinds of parent-child patterns



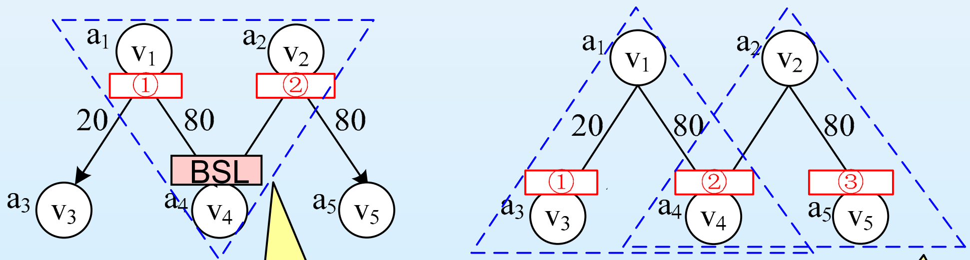
(a) One to one (b) One to one with self-loops (c) One to many (d) Many to one

  - Pattern (a): inserting one BSL at the three positions are the same.
  - Pattern (b): inserting one BSL at the three positions are the same.
  - Pattern (c): the number of BSLs is at most the number of children. Inserting at the exit of the branching parent may save some BSLs.
  - Pattern (d): at most one BSL needs to be inserted, and inserting at the entrance of the merging child-node will always be an optimal solution.

- Conclusion: for nodes with more than one incoming or outgoing edges, further reduction of BSLs can be achieved when considering the inserting positions and variable positions

For Space-difficulties

- Even though we know some strategies about BSL insertion on the basic patterns, there is no fixed best strategy like when optimizing for speed.
  - some **basic patterns could intervene** and create complex patterns
  - a better BSL insertion position also **depends on the variables' positions**



(a) (b)

If a3, a4 and a5 are in the same bank, this method is the best solution

But if you first consider this pattern (d), you may insert a BSL at the entrance of v4 at first

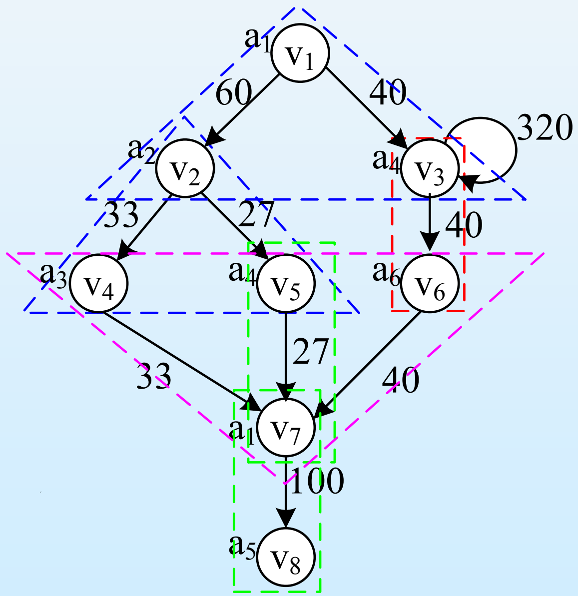
If a3, a4 and a5 are not in the same bank, always needs three BSLs.
This method is the one of the best solution

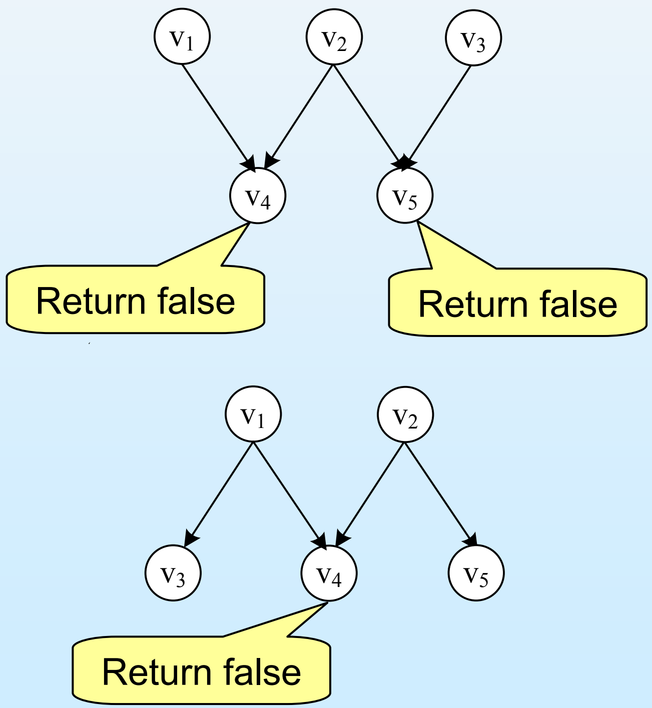
For Space-algorithm

- The space minimization problem is also NP-Hard.
- Even if the variables have already been partitioned into banks, the BSL inserting problem is still NP-Hard.
- We propose a heuristic algorithm:
  - Step0: Partition DFG into patterns;
  - Step1: Partition variables using AG;
  - Step2: Insert BSLs based on patterns.

For Space-algorithm: Step 0

- Step0. Partition DFG: Algorithm PCDFG to keep track of each basic pattern and partition a DFG into basic patterns.
  - if one child-node of v has in-degree(v) > 1 then return FALSE



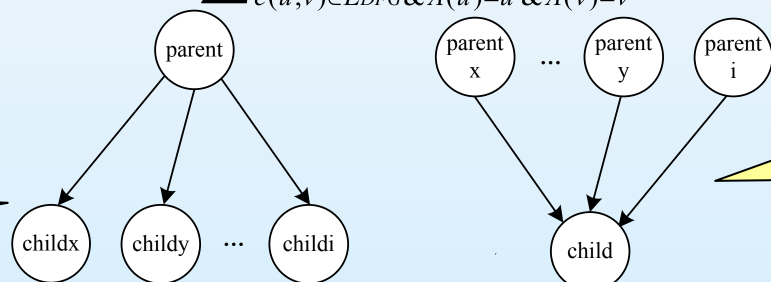


For Space-algorithm: Step 1

- Step1. Partition variables:
  - construct AG G'(V', E') for space
    - Calculate edges' weights w'(e'(u', v')):
      - 1. summate the number of edges whose vertices access u' and v':

$$w'(e'(u',v')) = \sum_{e(u,v)\in E_{DFG} \& A(u)=u' \& A(v)=v'} 1$$

We expect more children are in the same bank



It doesn't matter too much whether or not parents and child are in the same bank

      - 2. For pattern (c), enhance edges e'(A(childx), A(childy)): w'(e'(A(childx), A(childy))) + +;
      - 3. For pattern (d), slack edges e'(A(parentx), A(child)): w'(e'(A(parentx), A(child)) - -;
      - 4. Remove the edges with weight no bigger than zero
  - Use the proposed algorithm to partition variables

For Space-algorithm: Step 2

- Step2. Insert BSLs: based on patterns
  - For each basic pattern,

    give it an optimal insertion solution.
    - For pattern (a), (b) and (d):

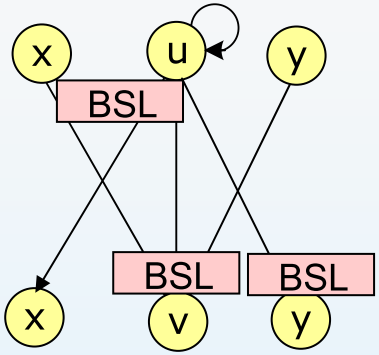
      insert one BSL at the entrance of the child-node if needed;
    - For pattern (c):

      while (variables accessed by some child-nodes are in the same bank) do

      Insert one BSL at the exit of the parent-node for these child-nodes if needed;

      end while

      Insert one BSL at the entrance of the child-node for each of the other child-nodes if needed;
  - For the other kinds of patterns, BSLs are always inserted at the entrance of the child-node.



Outline of this talk

- Background & Related work
- What the problem is? Variable partitioning & BSL insertion
- How to solve the problem?
  - For speed
  - For space
- Experimental results

香港城市大學
City University of Hong Kong

Experiments-benchmarks

Real Benchmarks				Constructed DFGs			
Name	Source	Nodes	Varis	Name	Nodes	Varis	Prop
SQUAR	SNU	19	7	BUS	150	20	basic patterns
FILTER	DSP	35	6	PACKET	200	15	few variables
EXP	MRTC	50	14	ROBUST	360	30	random
ROOT	SNU	65	18	PARALLEL	500	60	more parallel

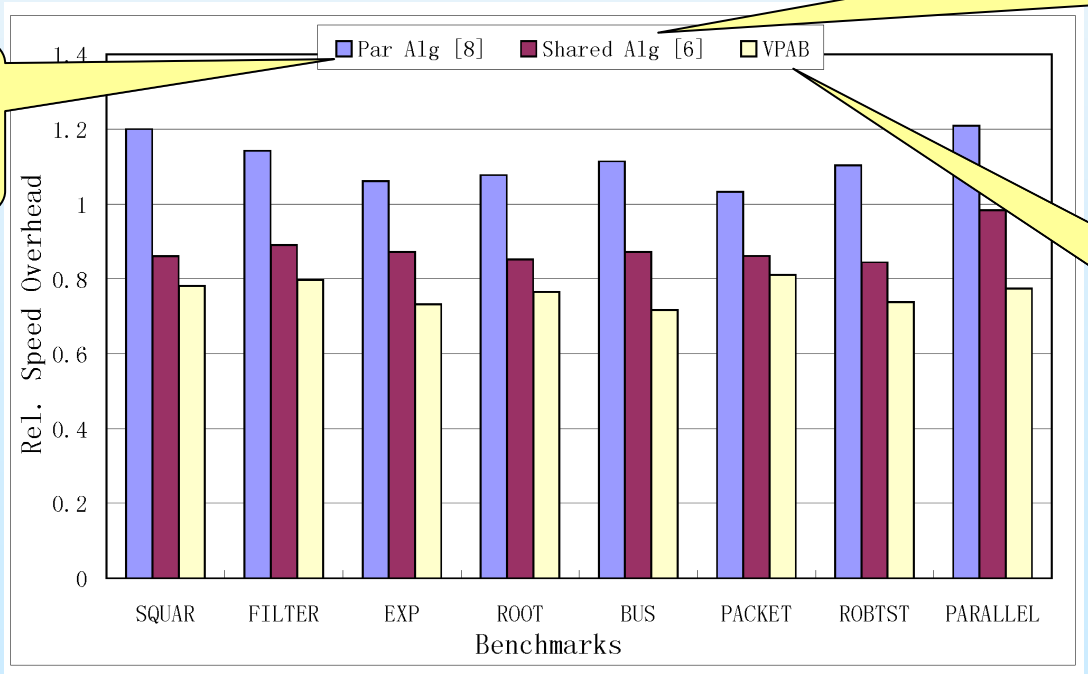
- A four-banks architecture is used for benchmarks with variables no more than 30
- An eight-banks architecture for the others.

Experiments-results for speed

- The partitioning techniques under comparison are:
  - (a) **equally** partitioning according to reference order,
  - (b) a **parallel** partitioning method [8],
  - (c) equally partitioning with **shared-bank** adjustment [6],
  - (d) the proposed algorithm for speed.

This method achieves 2-16% overhead reduction

This method offers the worst solutions



Our method achieves 19-29%overhead reduction

香港城市大學 City University of Hong Kong

Experiments-results for space

- For space, different BSL placement methods are also comp… because they also affect the solutions.

BSL \ Part	Partitioning (a)	Partitioning (b)	Partitioning (c)	
Opt[2][3]	Basis	ParAlg[8] &OptBSL	SharedAlg[6] &OptBSL	
				VPA

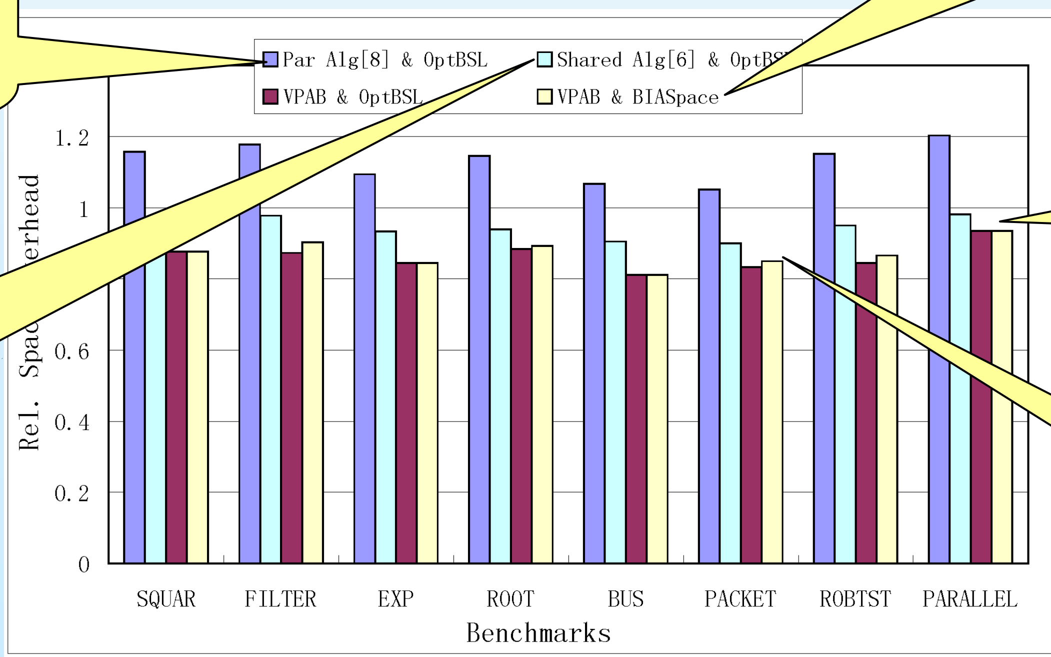
This method offers the worst solutions

This method achieves 2-10% overhead reduction

Even combined with the optimal BSL placement, the other partitioning algorithms still cannot obtain better results than the proposed algorithm

can obtain the best solutions for some cases

Only 2% worse than the optimal solutions.



Conclusion

- Architecture: 8-bit micro-controllers & partitioned banks & bank switching
- Objective: minimize the size and time overheads introduced by BSLs
- Through: optimizing the variable partitioning in different banks and elaborately placing BSLs in programs.
  - NP-Hard problems
  - Different positions to insert BSLs
  - Different patterns
  - Heuristic algorithms are proposed.
- Future work
  - Combining multiple objectives
  - Special cases: optimal solutions

Thanks!
Q & A