Automating Instruction-Set Extension for Embedded Processor Customisation

Laura Pozzi

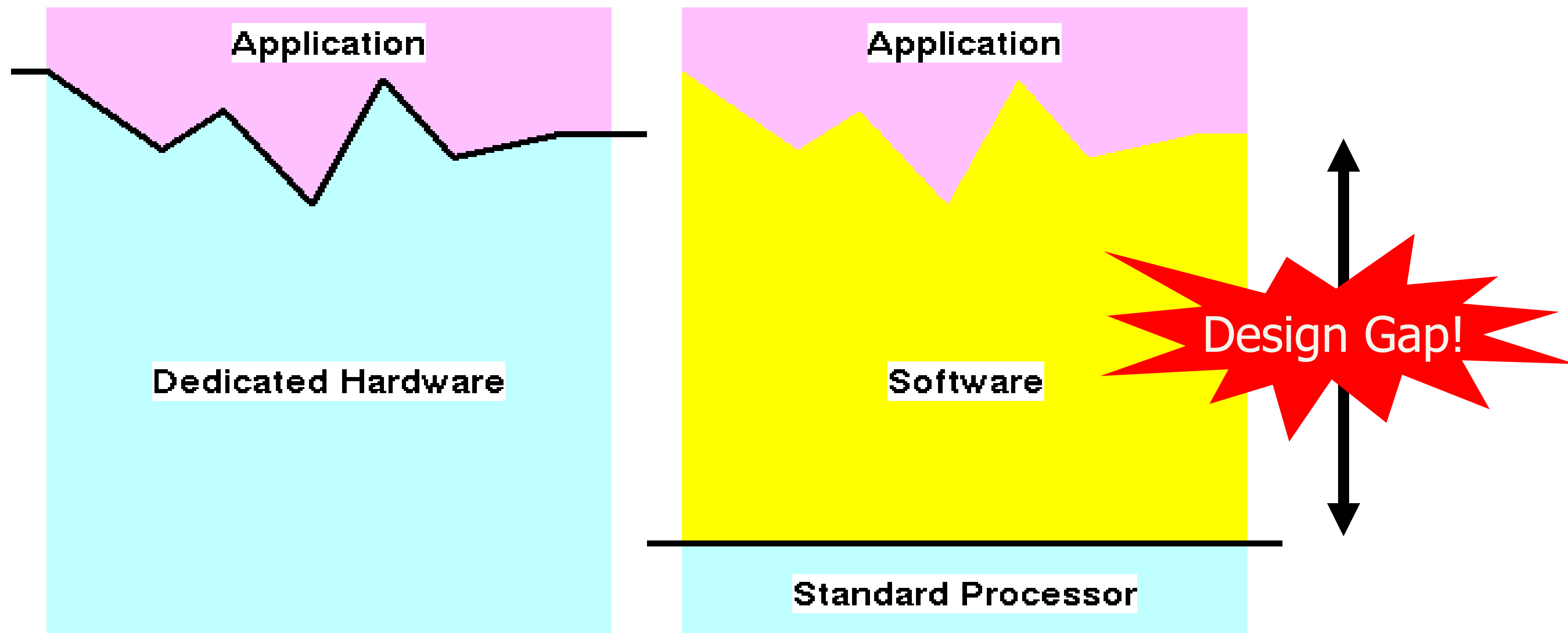
With:

K. Atasu, P. Biswas, A. Peymandoust, D. Jain, P. Ienne, N. Dutt, G. De Micheli

Processor Architecture Laboratory (LAP)
& Centre for Advanced Digital Systems (CSDA)

Swiss Federal Institute of Technology Lausanne (EPFL)

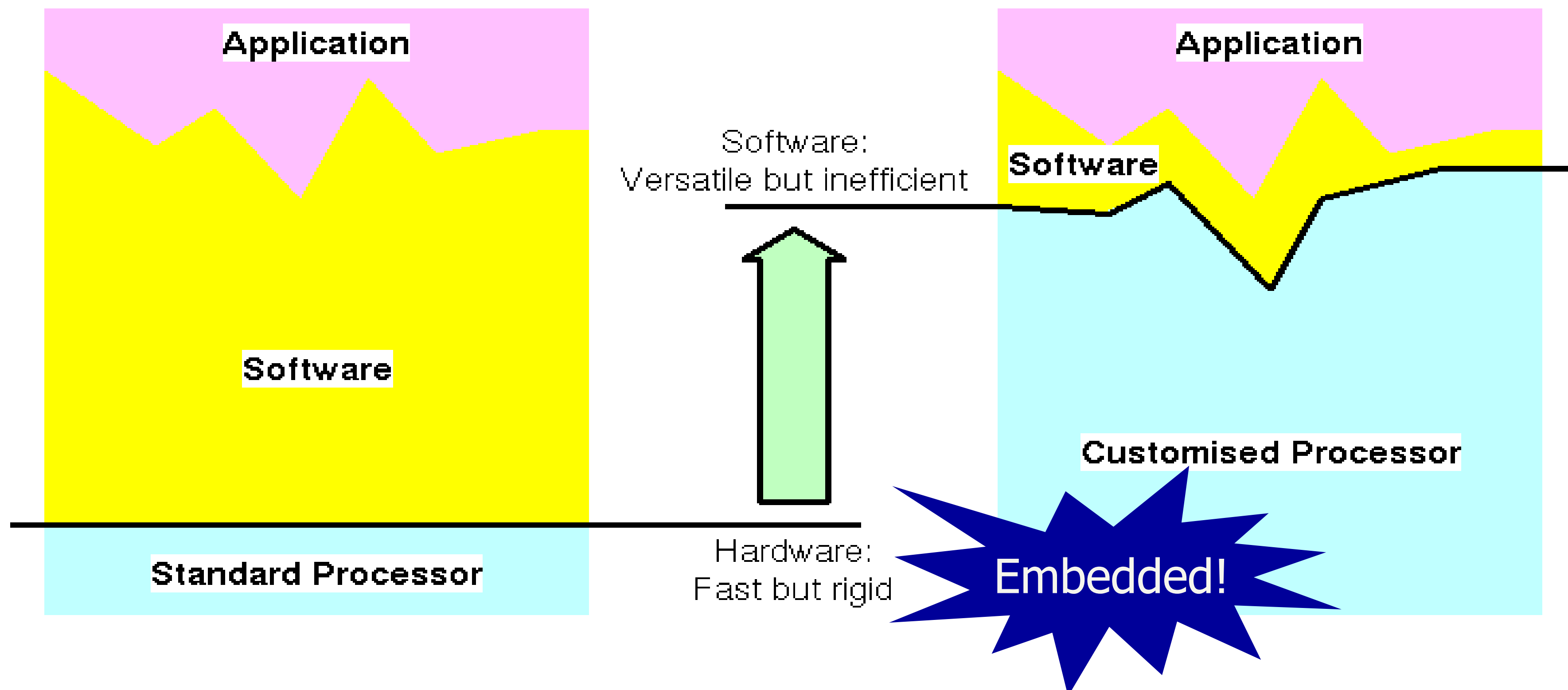
Automatic Processor Specialisation: How Application and HW Meet Today



+++	Highest performance
---	Long time to market
--	No field changes

-	Lowest performance
++	Best time to market
++	Any field changes

Automatic Processor Specialisation: How HW Should Meet Applications



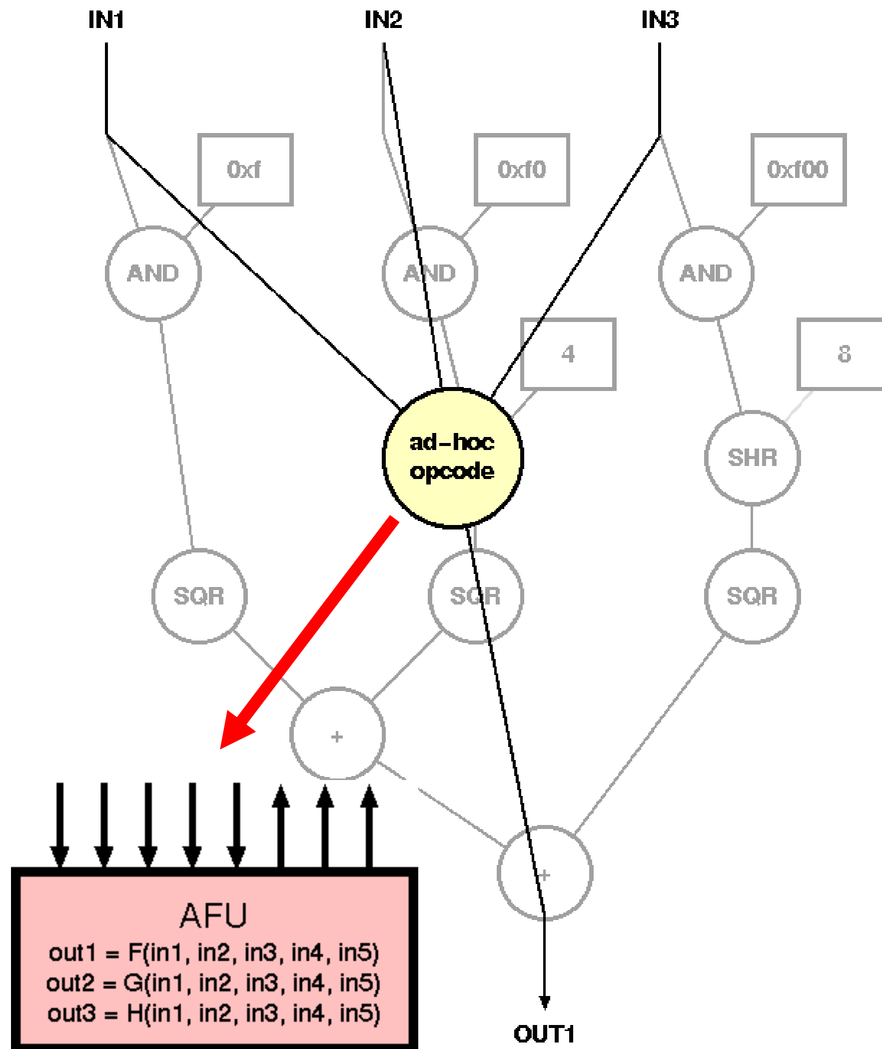
- – Lowest performance
- ++ Best time to market
- ++ Any field changes

- ++ Very high performance
- \+ Good time to market
- \+ Any field changes

Outline

- Introduction to Instruction Set Extension (ISE)
- Our proposal for solving the problem automatically
  - Set of algorithms
  - Different flavours of the ISE problem
- Open research issues
- Conclusions

Instruction Set Extensions



- Collapse a subset of the Direct Acyclic Graph nodes into a single Functional Unit (AFU)
  - Exploit cheaply the parallelism within the basic block
  - Simplify operations with constant operands
  - Optimise sequences of instructions (logic, arithmetic, etc.)
  - Exploit limited precision

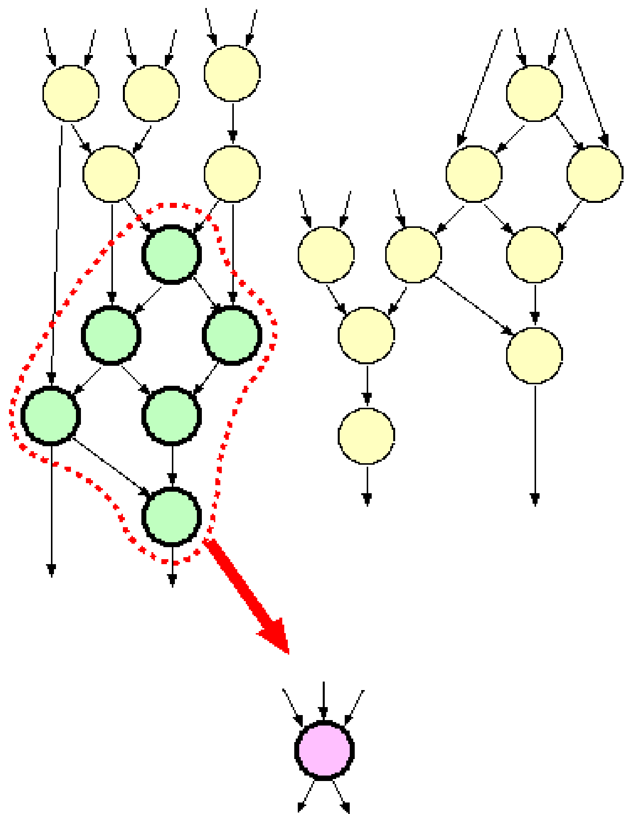
Automatic Instruction-Set Extensions

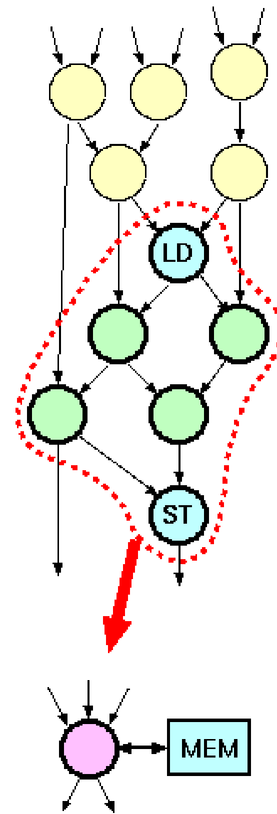
Automatic Identification of Instruction-Set Extensions

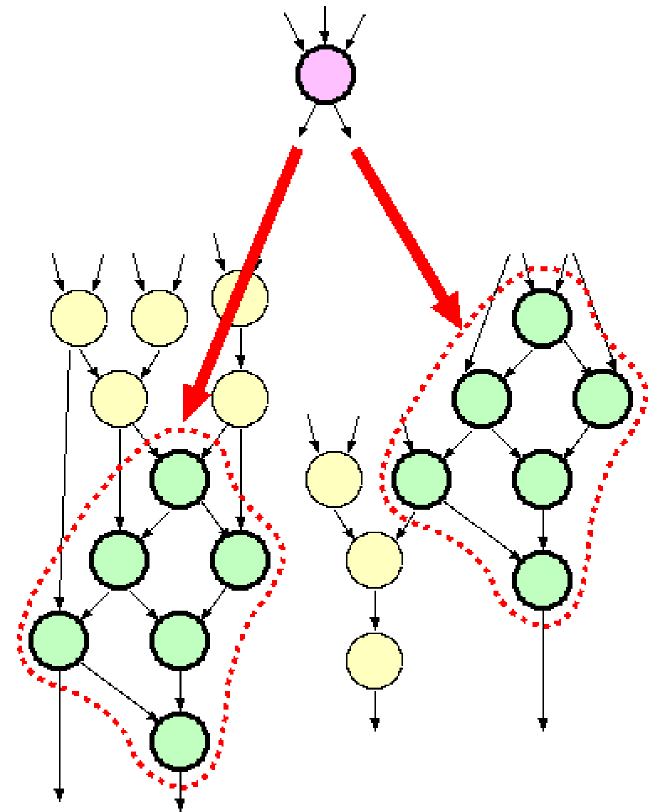
Atasu, Pozzi, Ienne – DAC 2003 *(Best Paper)*

Biswas, Pozzi, Ienne, Dutt, et al. – DAC 2004

Biswas, Pozzi, Ienne, Dutt, et al. – DATE 2005



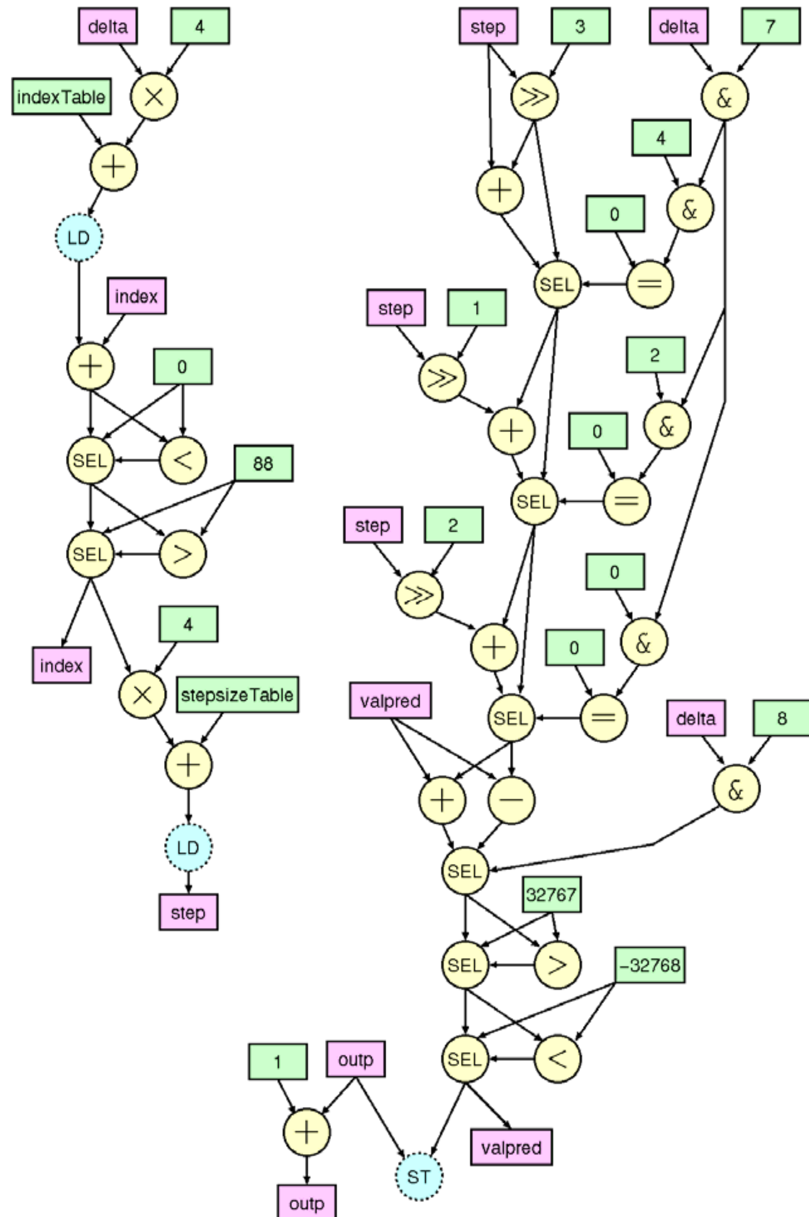




Symbolic Algebra for Instruction Selection

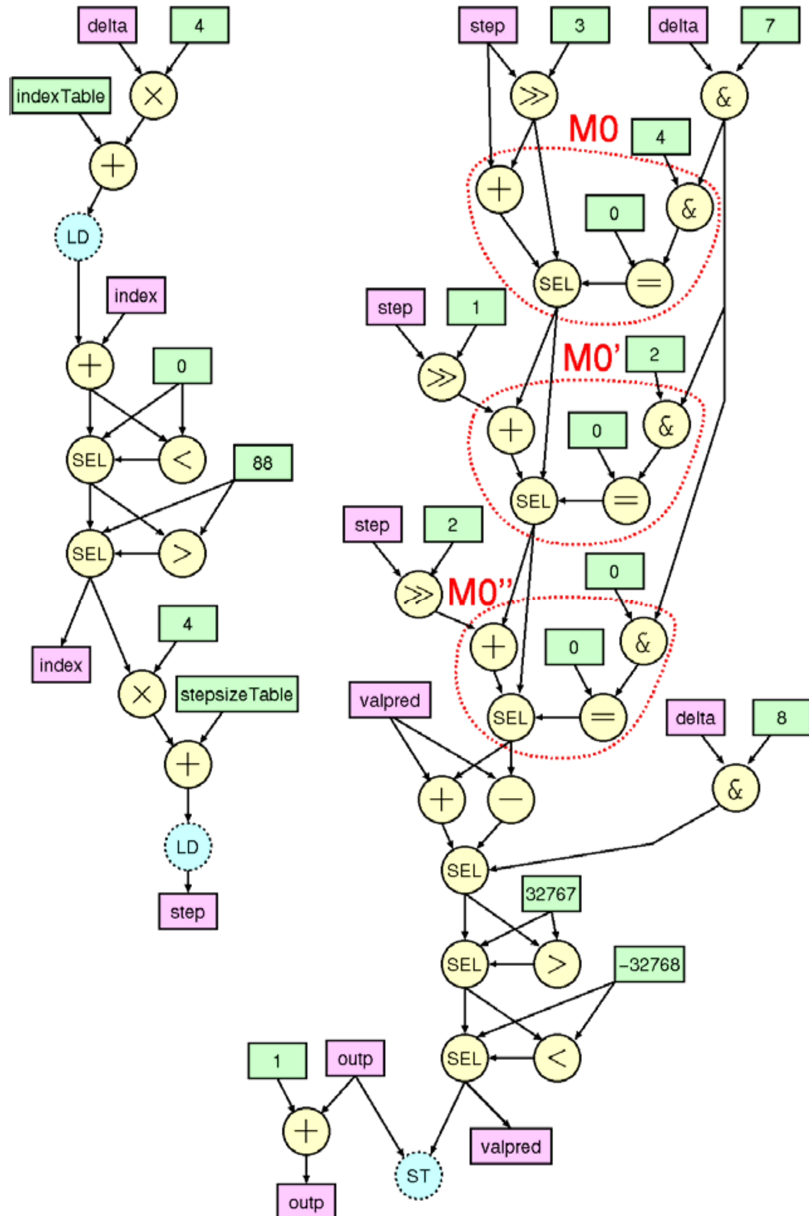
Peymandoust, Pozzi, Ienne, De Micheli – ASAP 2003

Motivational Example and Goal



- **Goal**: Find subgraphs
  - having a user defined maximum number of inputs and outputs,
  - including disconnected components, and
  - that maximize the overall speedup
  - that are close to manual solutions

Existing Solutions Miss Potential Speed-ups



- Typical approach (I)

Find frequent patterns

  - Typically rather small
  - Might have too many inputs or outputs for register file
  - Reuse is not a good heuristic for high speedup

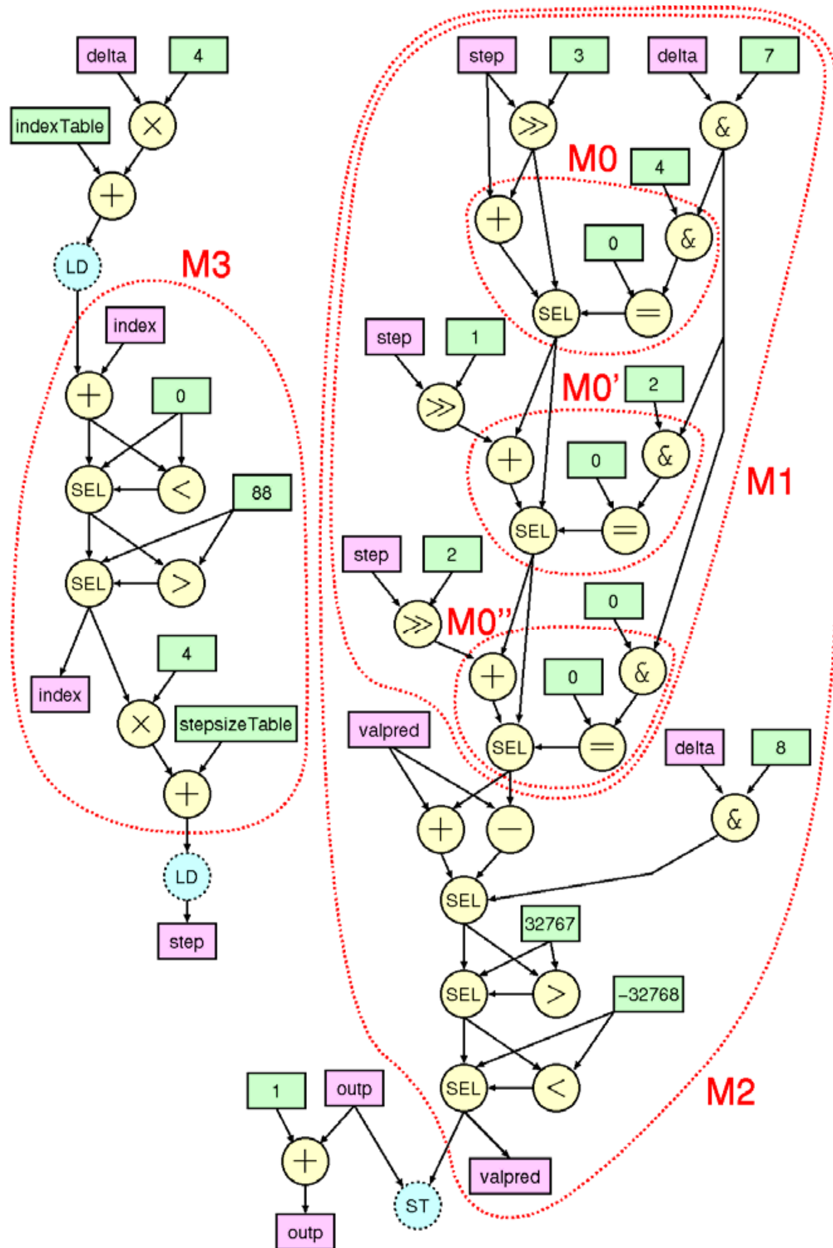
  E.g., ChoiJun99, KastnerOct02, ArnoldApr01

- Typical approach (II)

Grow clusters until I/O violation occurs

  - Limits possibilities, only connected
  - Usually only single output

Our Goal



- M1 for 2-1
- M2 for 3-1
- M2+M3 for 4-3

Problem Statement

- $G_i(V, E)$ are the graphs representing the DAGs of the algorithm basic blocks
- S is a subgraph of G
- $M(S)$ represents the gain achievable by implementing the subgraph S as a special instruction

---

- **Problem:** Find the N_{instr} subgraphs S_j of any G_i such that

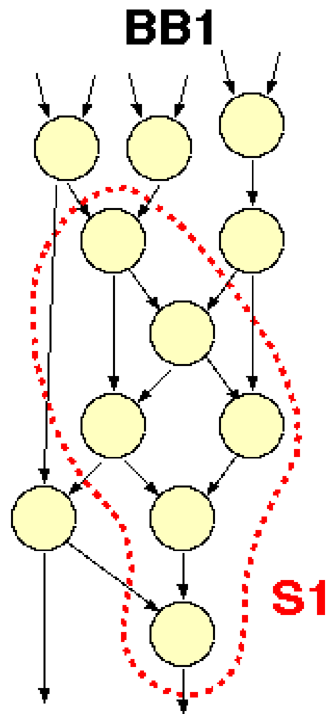
$$\sum_j M(S_j) \rightarrow \max$$

- Under the following constraints for each subgraphs S_j
  - Number of inputs of $S_j \leq N_{in}$
  - Number of outputs of $S_j \leq N_{out}$
  - S_j is convex

Identification Algorithms

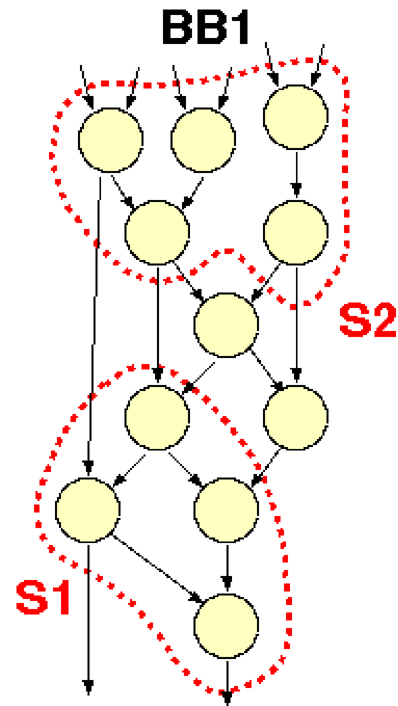
1

Single Subgraph
Single Basic Block



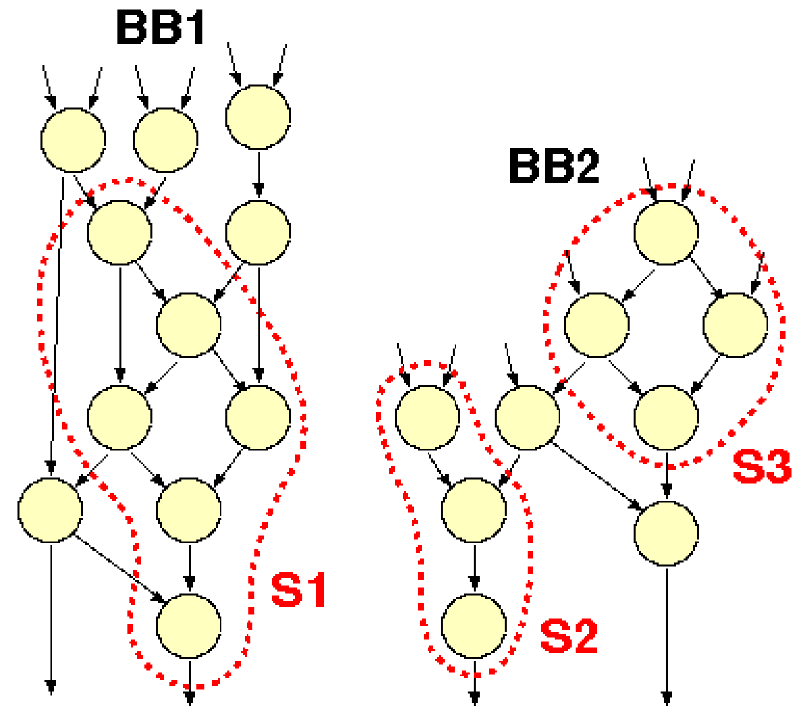
2

Multiple Subgraphs (e.g., 2)
Single Basic Block



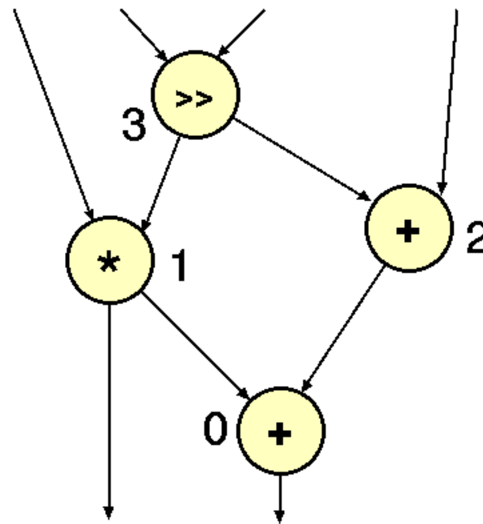
3/4

Multiple Subgraphs (e.g., 3)
Multiple Basic Blocks



Single Subgraph within a Single Basic Block

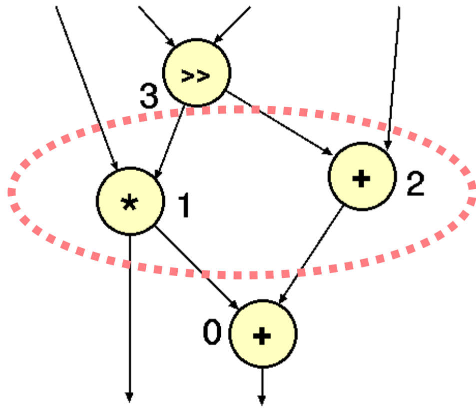
- A graph with N nodes has 2^N subgraphs
- Potential solutions are in fact rather sparse



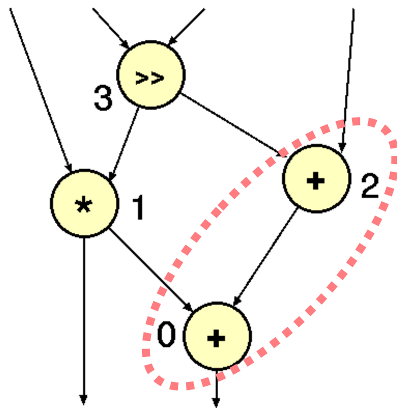
How to avoid exploring unnecessarily the whole design space?

Search Space Pruning

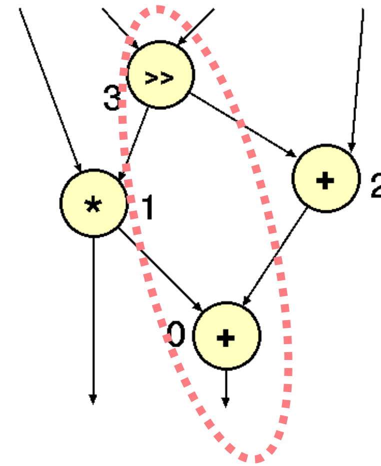
- Based on a violation of the **output port** constraint



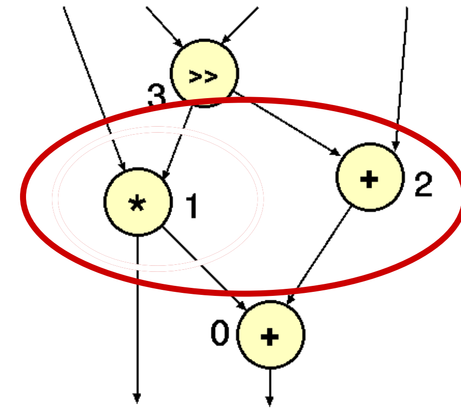
- Based on a violation of the **input port** constraint

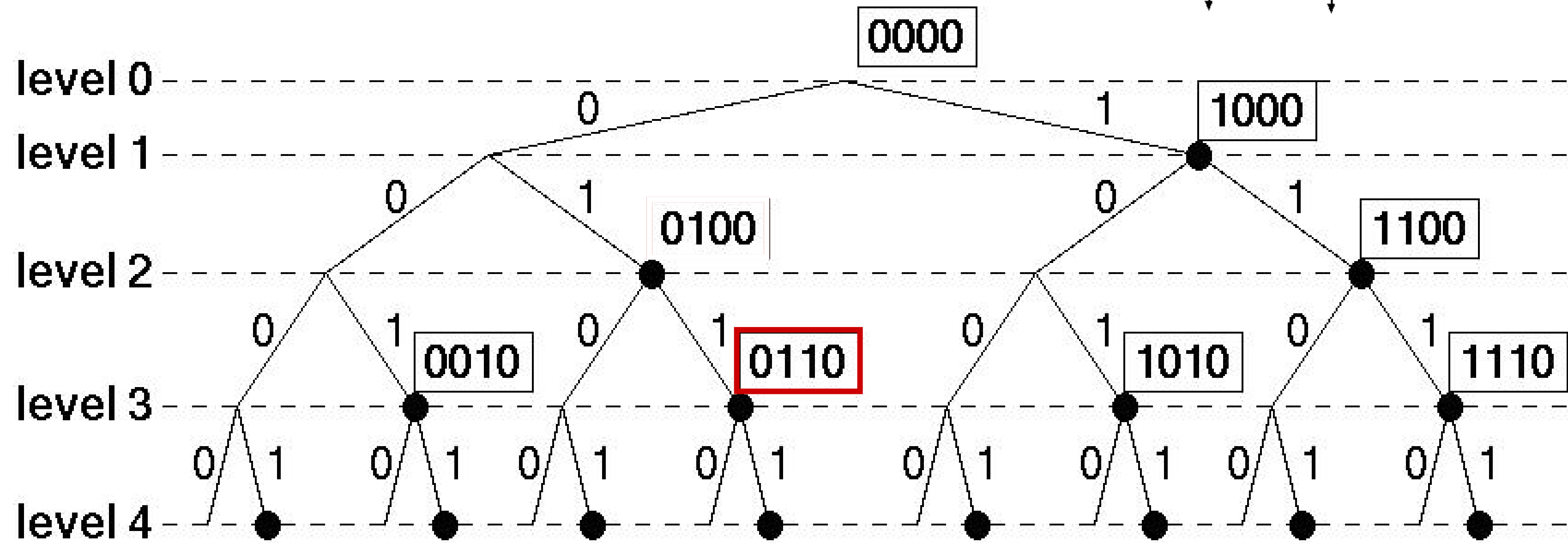


- Based on a violation of the **convexity** constraint



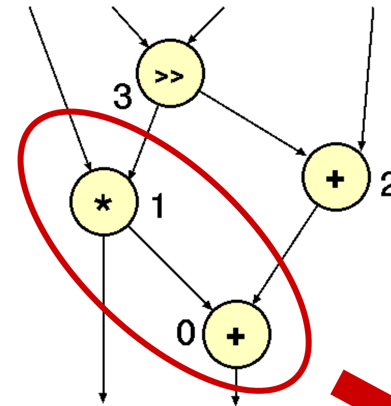
Search tree construction

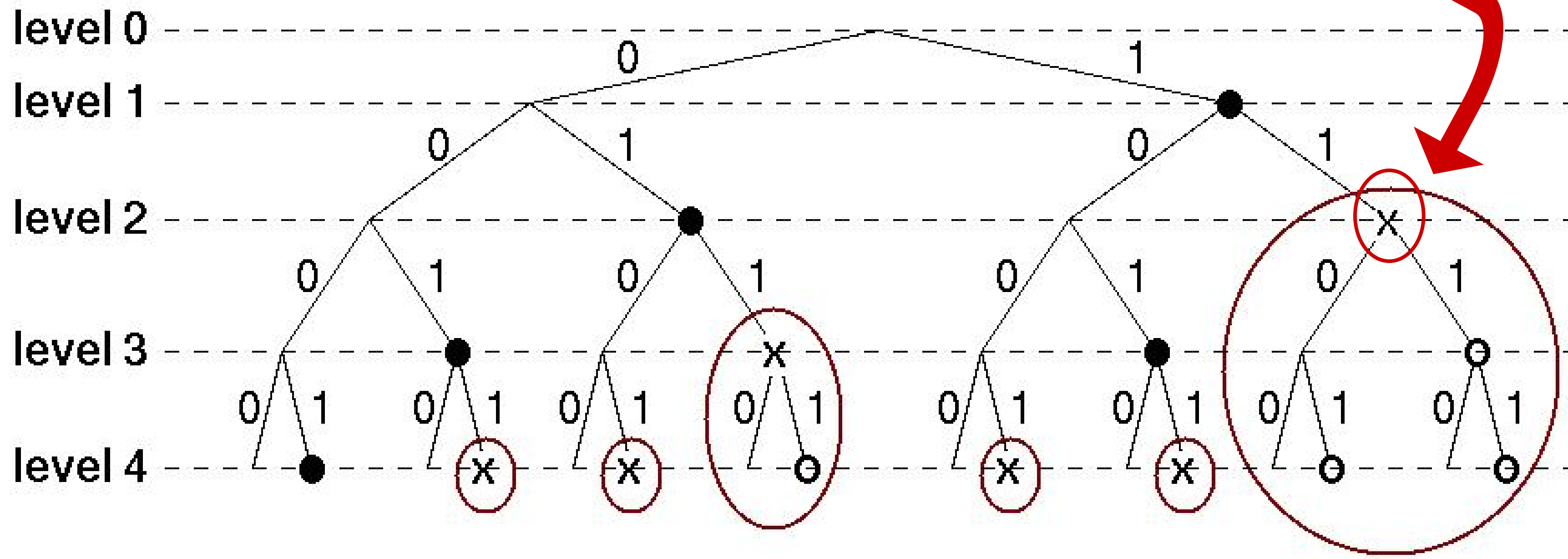




Search tree pruning

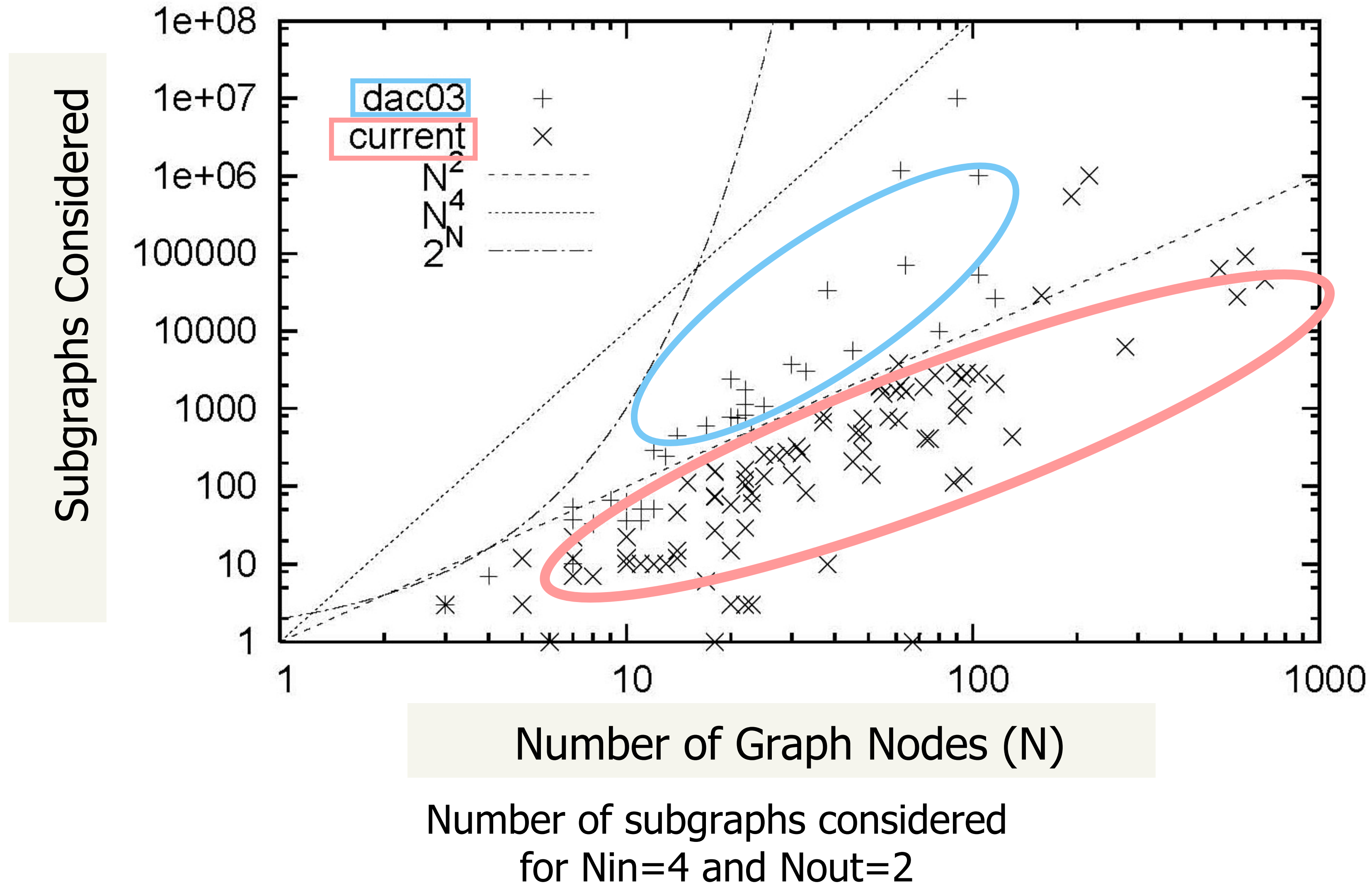
Topologically sorted graph!





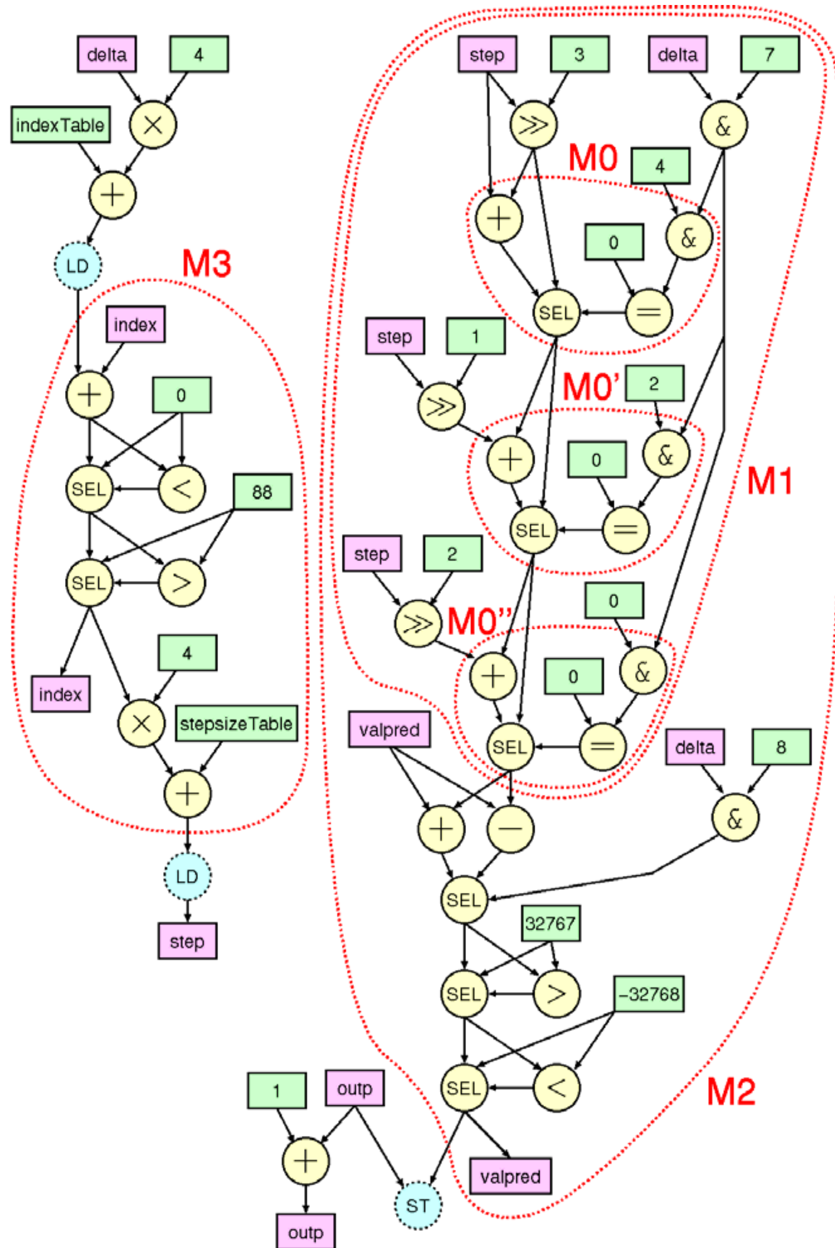
for Nout = 2

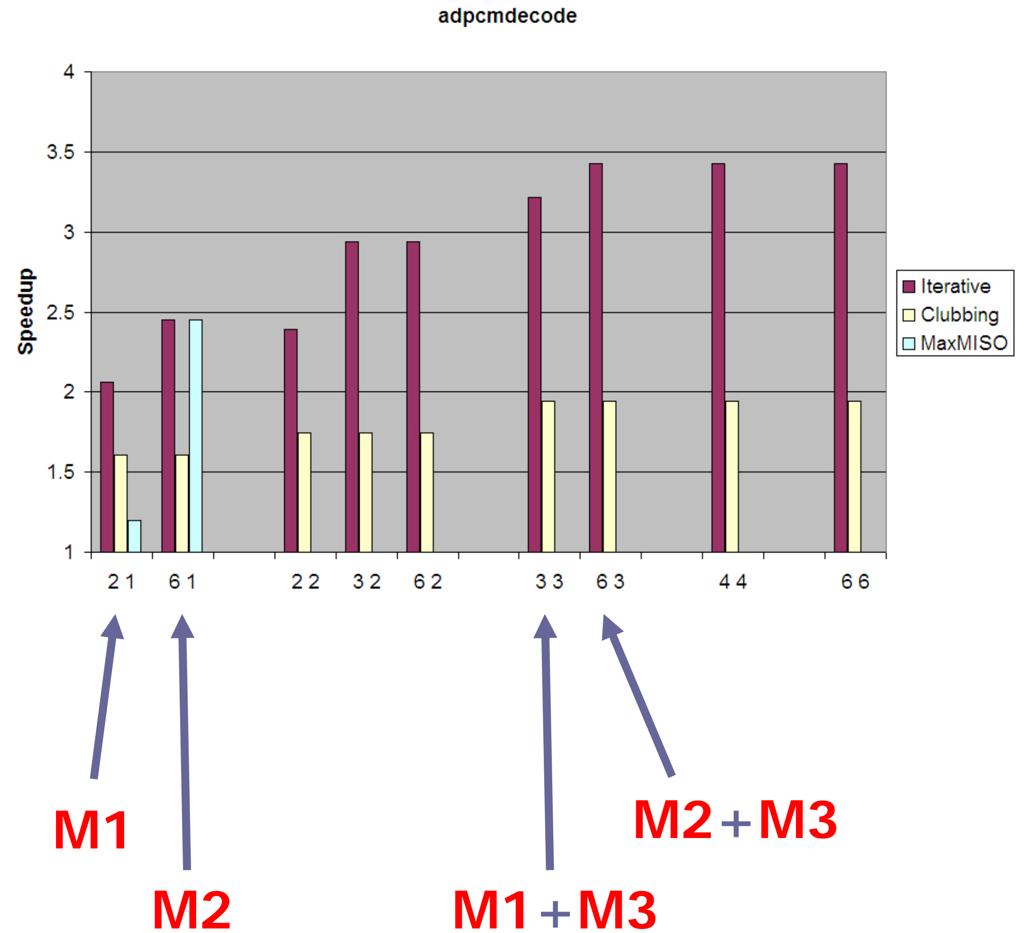
Current Algorithm Performance



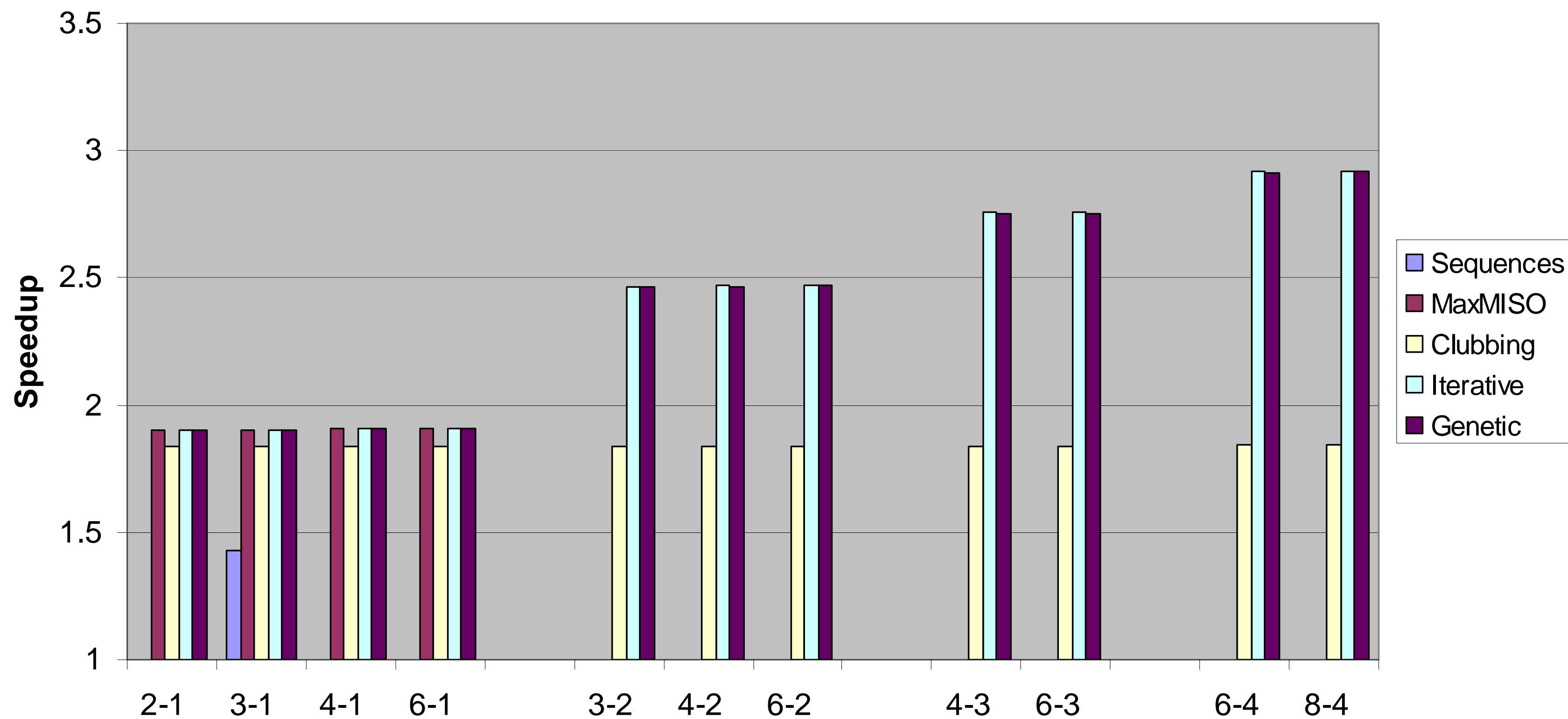
Number of subgraphs considered
for Nin=4 and Nout=2

Identification Results

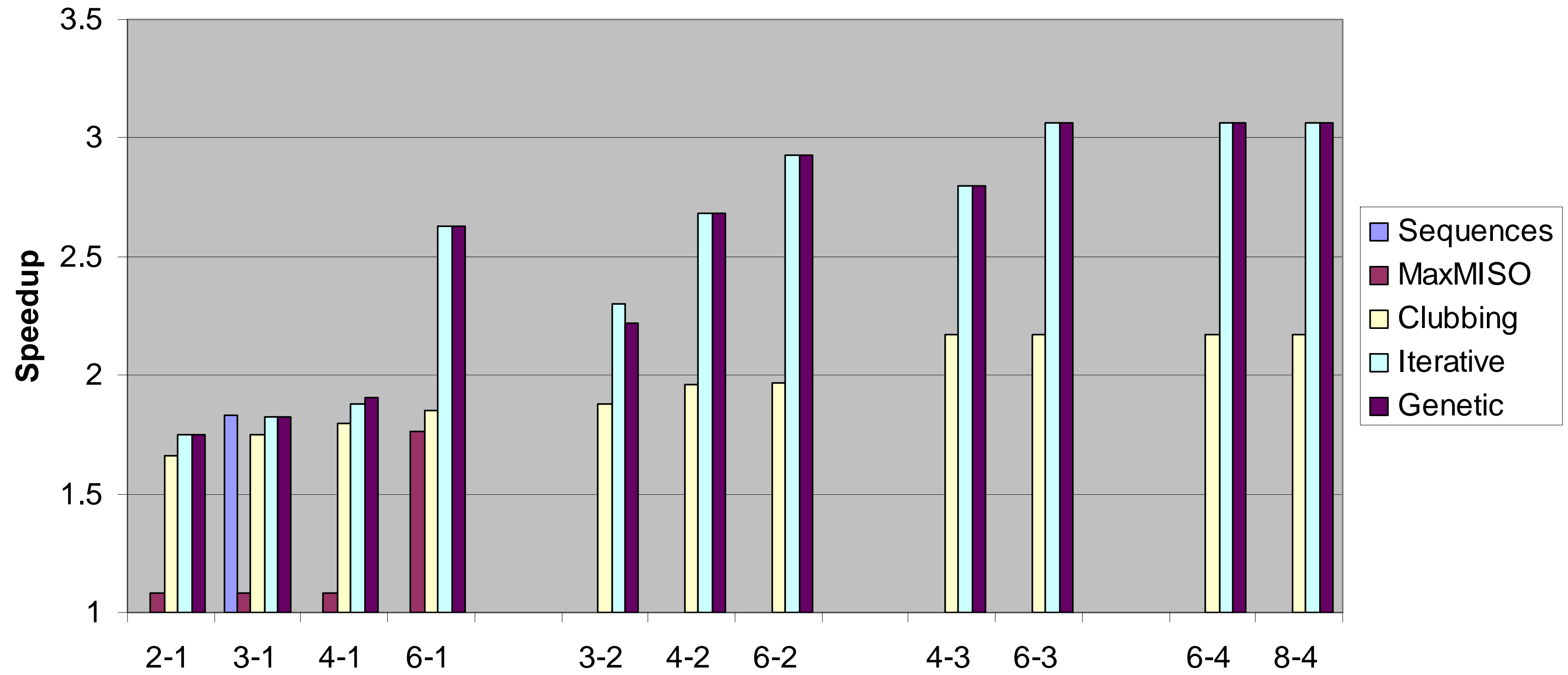




Results



Results

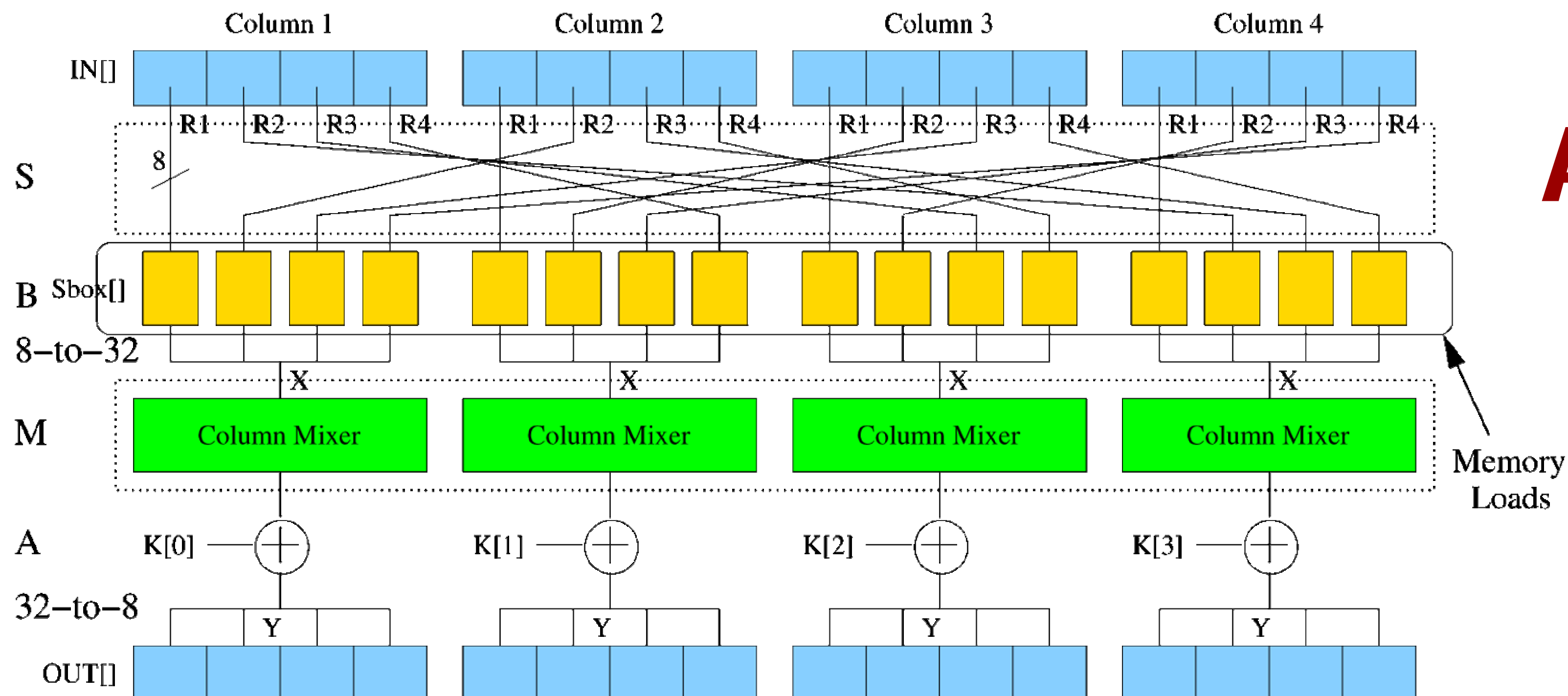


And what about Memory?

- Large share of **system energy spent in memory**
- Many techniques known for improving caches
  - Drowsy caches, Cool caches, Zero-compression caches, Filter caches, Software-controlled caches, etc.
- Can processor specialisation help?
  - Reduce **Instruction Fetches** (← one of the reasons of lower additional power cost than performance gain)
  - And in **Data Memory**?

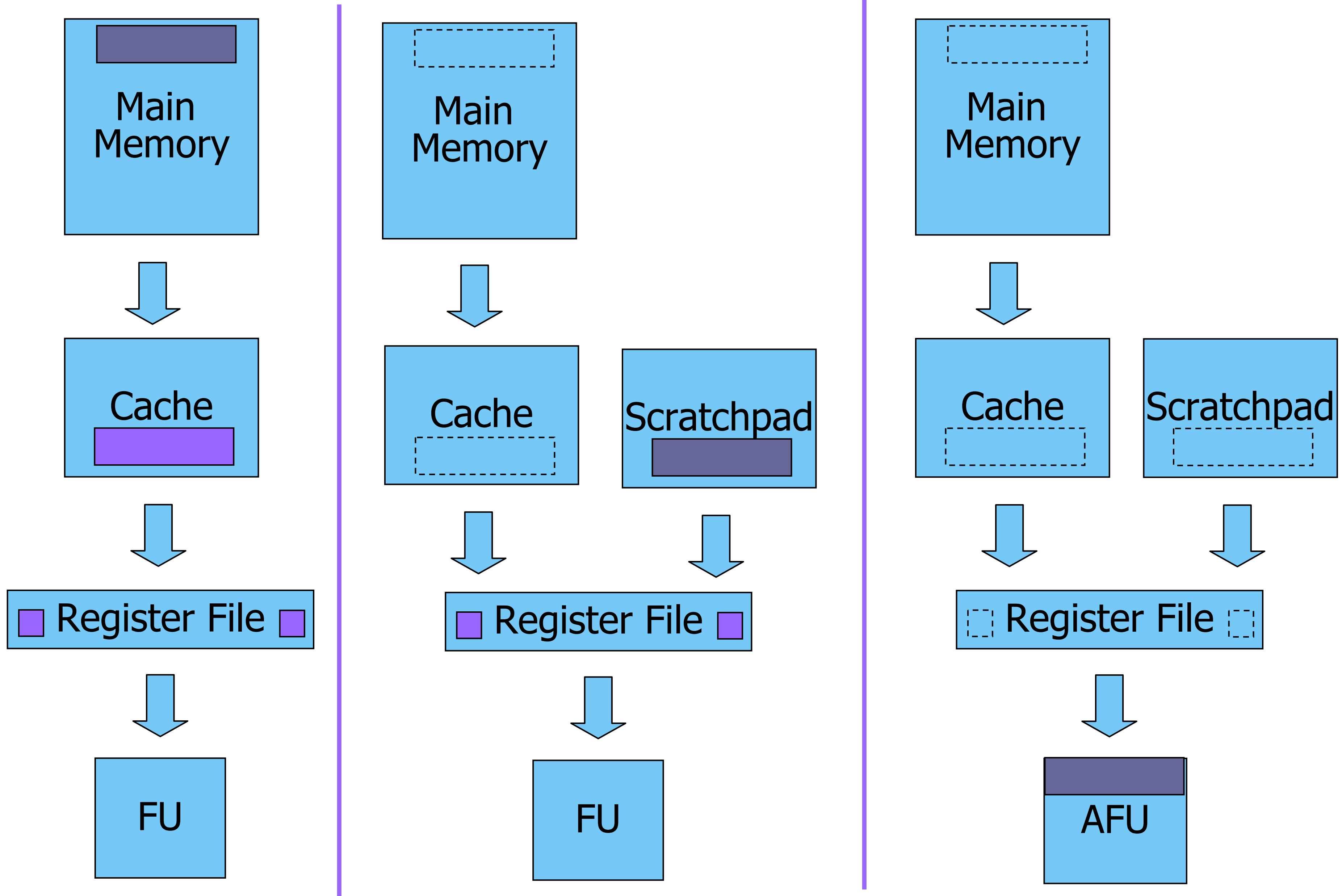
Introducing LD/ST in ISE

- Typical situation #1: **constant tables**
  - Encoding and quantisation tables
  - Cryptography substitution tables (S-Boxes)



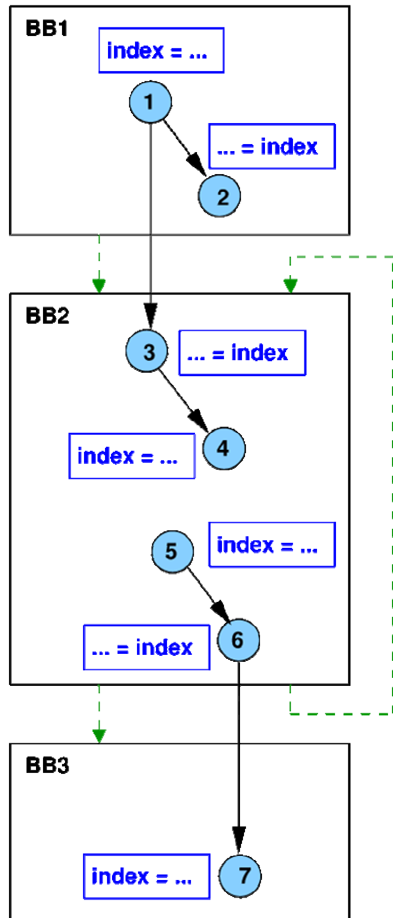
AES

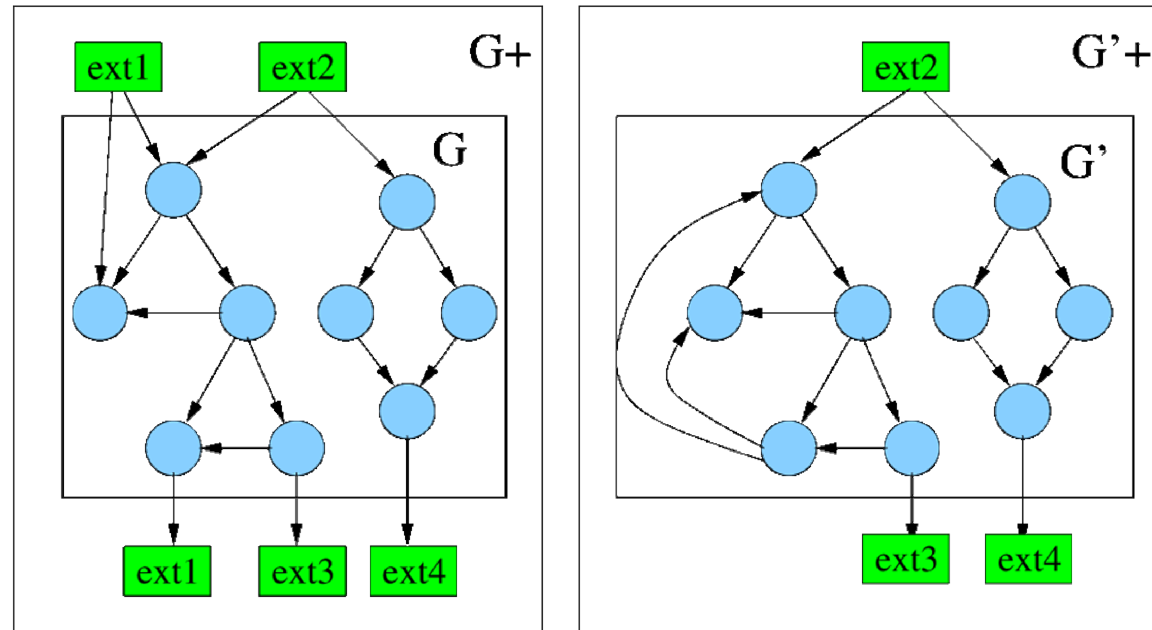
Move tables closer to the core



Introducing LD/ST in ISE

- Typical situation #2: **Loop-carried dependencies**

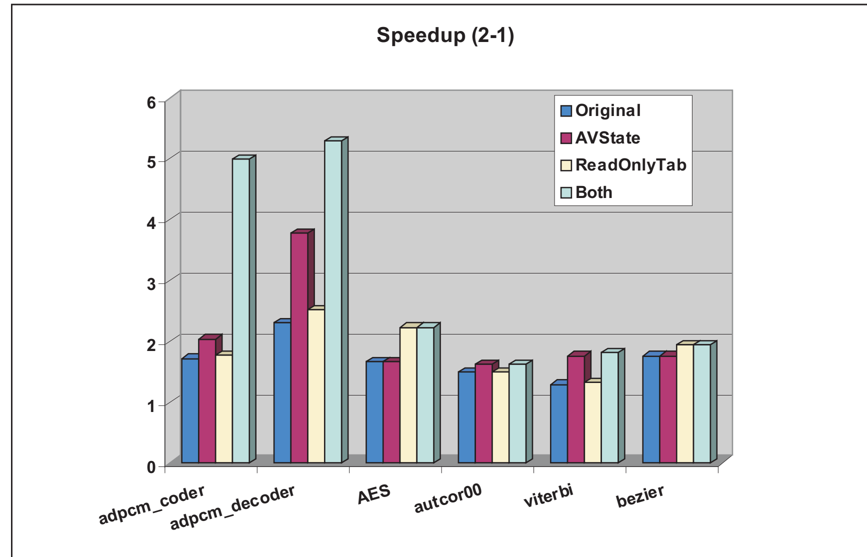


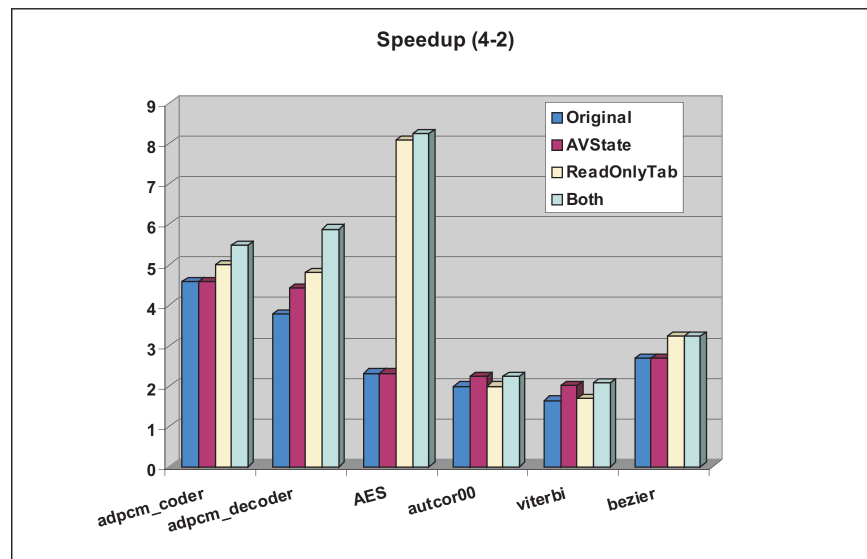


A **variation** of the previous Instruction-Set Extension identification

adpcm

Speedup Results





- Tangible advantage of including **Architecturally Visible State Registers** and **Read Only Tables** into ISEs
- Advantage is often **more than cumulative** for the two types of memory
- Advantage is in some cases **more pronounced for limited bandwidth register files** (e.g., 2-1 vs 4-2)

Energy Savings in the D-Cache

Benchmark	Accesses in Hardware Tables	Accesses in D-Cache	% Energy Saving
AES	16,408	21,859	**34**
adpcm_encoder	295,188	443,892	**32**
adpcm_decoder	295,188	591,412	**26**
bezier	48,004,004	61,208,005	**35**
viterbi	2,160,000	72,183,000	**2**

- Energy cost of accessing caches estimated for a **direct mapped 32kbyte cache** (lowest energy)
- Energy cost of accessing hardware tables estimated as **ASIC SRAMs**

Open Issues: Generic Memory within AFUs

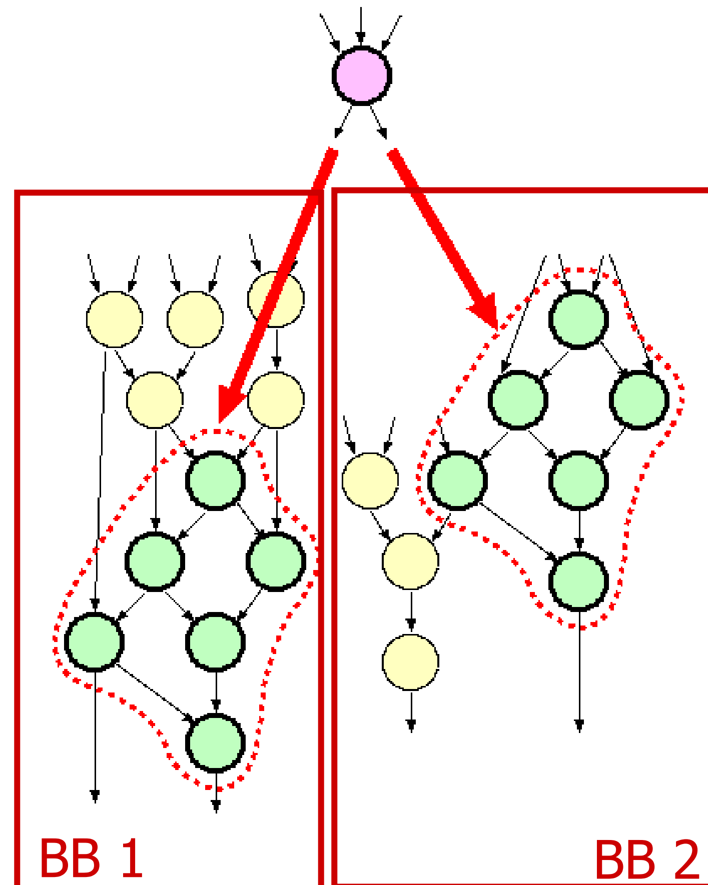
- Of course the read-only table inclusion in AFUs is a first step towards ‘generic memory within AFUs’
- Ideally, when there is *reuse* of memory space (e.g., vector elements) within a loop body, it would be advisable to read-in such vectors once in the AFU, and then write them out once, if needed
- Can use the theory of scratchpad memory allocation, which is a very similar problem

Open Issues: Region Formation

- We are efficiently finding ISEs within BBs.
- Sometimes BB enlargement is needed in order to match designer expectations
- Techniques we have found useful, after application study:
  - If-conversion
  - Loop unrolling
  - Function inlining
- The question is when and how much
- Probably typical heuristics:
  - Only ‘hot-enough’ spots
  - Only ‘small-enough’ loop-bodies / functions
    - (but ‘small-enough’ means hw area now, not sw instructions)

Open Issues: Recurrence

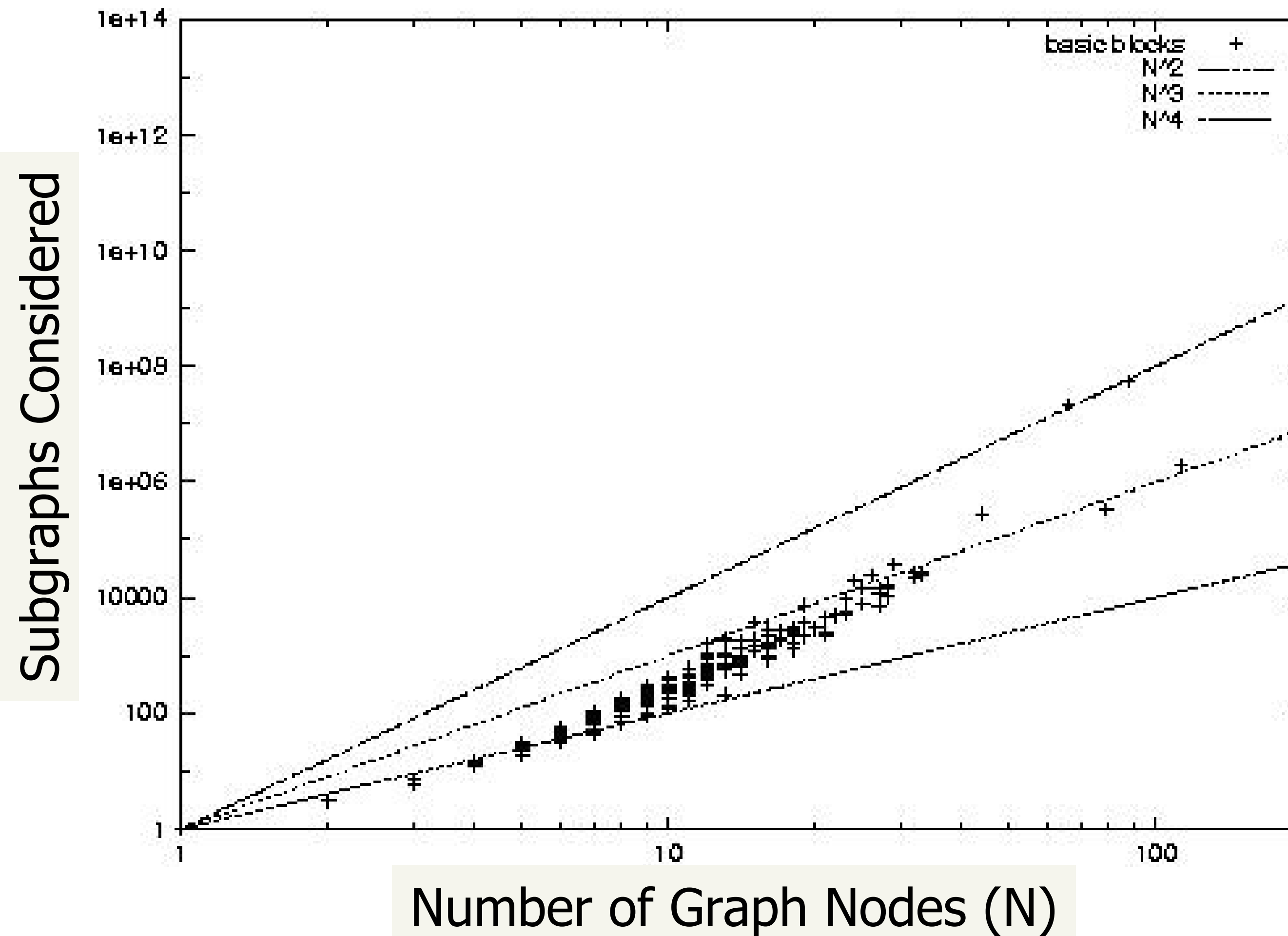
- Our problem formulations so far search for the *best* AFU in a basic block, without caring for recurrence
- What about recurrence?
- → subgraph isomorphism



Conclusions and current work

- DAC03 has good formulation and results, but is exponential in worst case; limited to 100 nodes
- Recent improvements to the above algorithm have brought additional pruning and made it deal with up to 1000-1200 nodes
- DAC04 proposes automated inclusion of read-only tables or loop carried dependent scalar variables in AFUs, plus it proposes a heuristics based on genetic algorithms
- DATE 05 proposes a heuristics based on the K-L partitioning algorithm, which is more efficient than a previous genetic formulation
- Current work is exploring
  - Region formation and generic memory inclusion, for furthering the reach of our ISE algorithms
  - AFU recurrence – area constraintk

Algorithm Performance



Number of subgraphs considered
using an output port constraint of two