Lecture 16: Embedded Compiler Optimisations

Michael O'Boyle
Embedded Software

Overview

- Introduction
- Address generating units
  - Single offset assignment problem
- Instruction level parallelism
  - Going beyond list scheduling
- Vectorisation for multiimedia instructions
- Avoiding branch delay
  - Using predicated instructions
- Function inlining
  - Trade off betweern code size and performance
- Summary

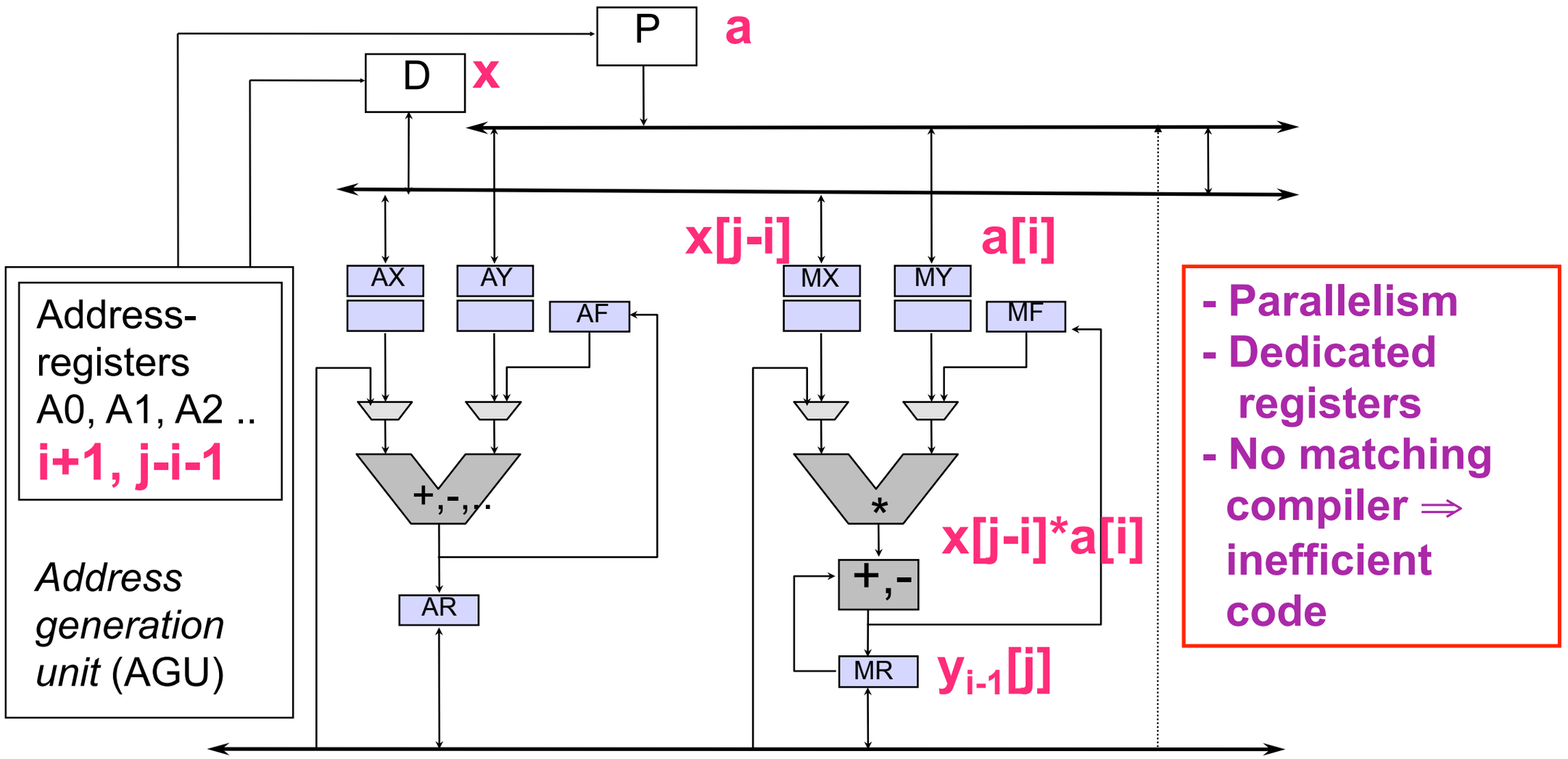
Introduction

- Previously looked at large scale mapping issues
    - Look at smaller scale intra-processor utilisation
- Traditionally embedded processors programmed in assembler
    - As programmer cost rises, more emphasis on tools
- Compilers are a key component in exploiting embedded sysytems
- Different challenges and opportunities compared to general purpose
    - Architectures more complex and pre-specialised
    - Code size energy as well as speed
    - More work to do as less hardware support
- Compile time can take longer as amortised over number of uses

Address generation unit in parallel

Code : $y_i[j] = y_{i-1}[j] + x[j-i]*a[i]$ forall i, j

Example: Data path ADSP210x



Compiling for AGus

- In effect does restricted register indirect addressing in parallel

  LD r31, Mem[r1]
  ADD r1, r1, 1
  LD r31, Mem[r1]

- Is replaced by

  LD r31, Mem[a1]; (a1++)

- Where the a1++ may be explicit or implicit
- Very useful for loops and array refs

  Eliminates an instruction per load - code size

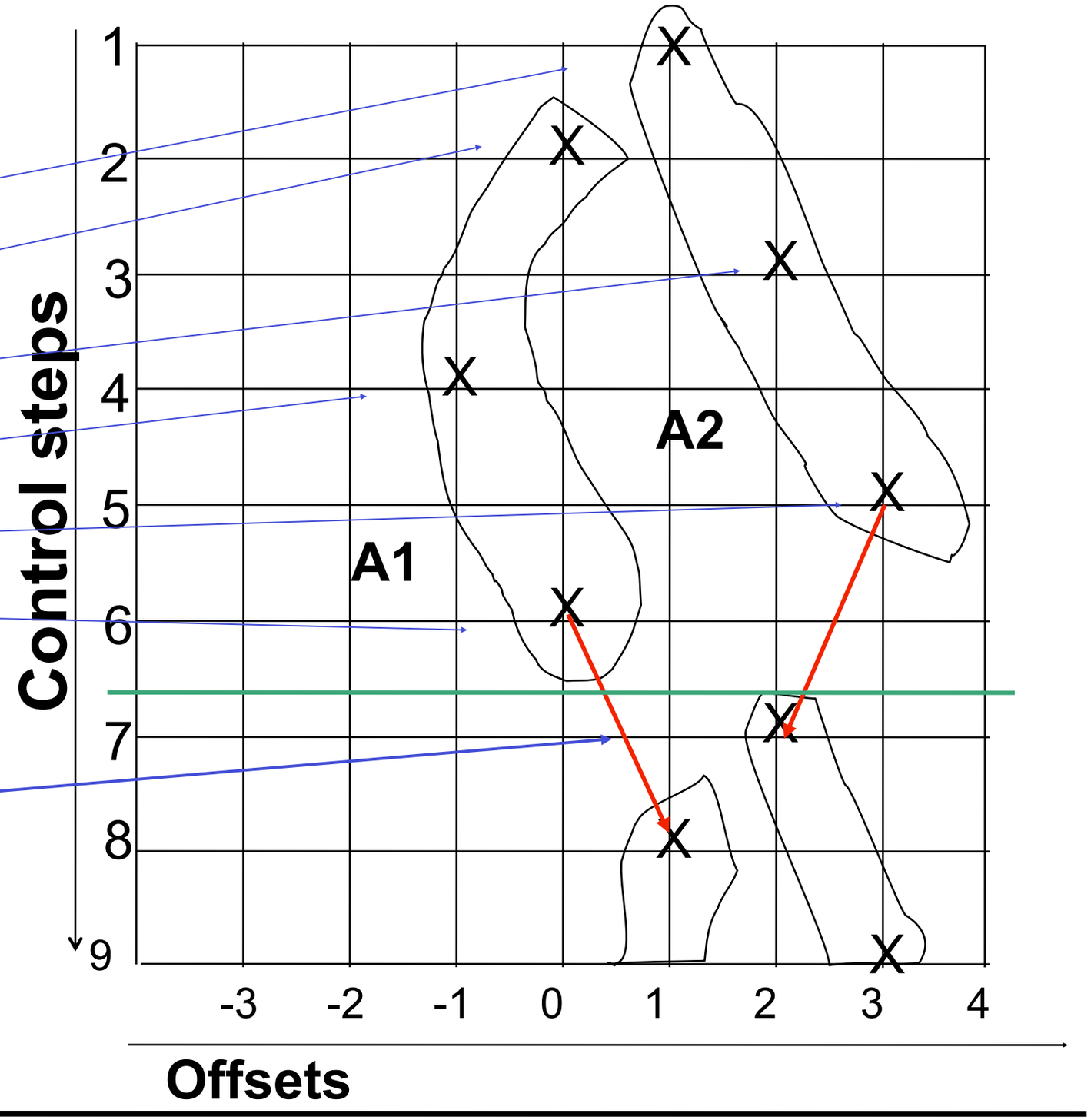
  However a challenge on how to utilise agus

Handling array references in loops

Example:

```
for (i=2; i<=N; i++)
{ .. B[i+1]     /*A2++   */
  .. B[i]       /*A1--   */
  .. B[i+2]     /*A2++   */
  .. B[i-1]     /*A1++   */
  .. B[i+3]     /*A2--   */
  .. B[i] }     /*A1++   */
```

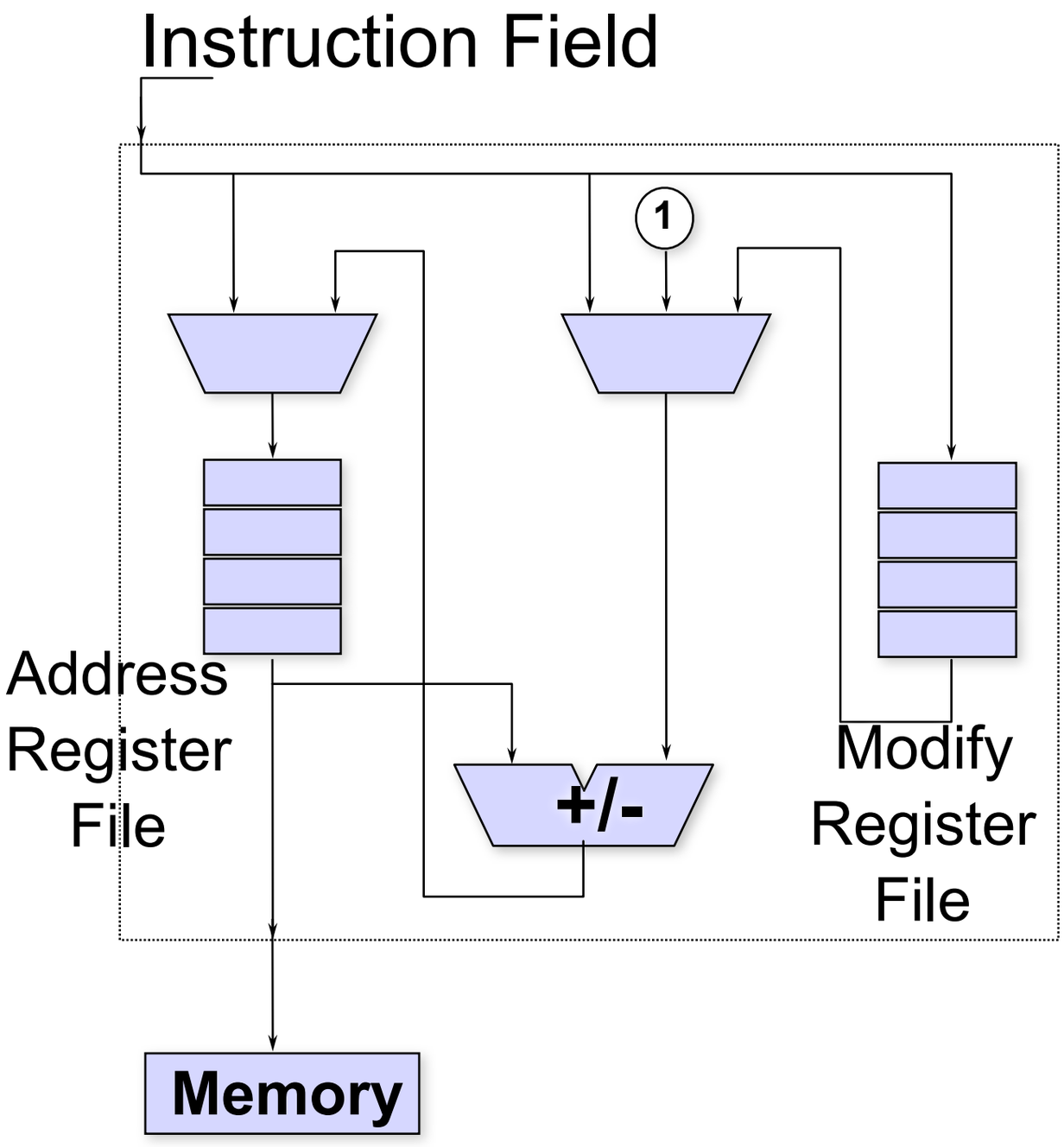
Cost for crossing loop boundaries considered.



Reference: A. Basu, R. Leupers, P. Marwedel: Array Index Allocation under Register Constraints, Int. Conf. on VLSI Design, Goa/India, 1999

Exploitation of parallel address computations

Generic address generation unit (AGU) model



Parameters:

k = # address registers

m = # modify registers

Cost metric for AGU operations:

Operation	cost
immediate AR load	1
immediate AR modify	1
auto-increment/ decrement	0
AR += MR	0

Offset assignment problem (OA)
- Effect of optimised memory layout -

Let's assume that we can modify the memory layout

- ☞offset assignment problem.

(k,m,r)-OA is the problem of generating a memory layout which minimizes the cost of addressing variables, with

- ☞k: number of address registers
- ☞m: number of modify registers
- ☞r: the offset range

The case (1,0,1) is called simple offset assignment (SOA), the case (k,0,1) is called general offset assignment (GOA).

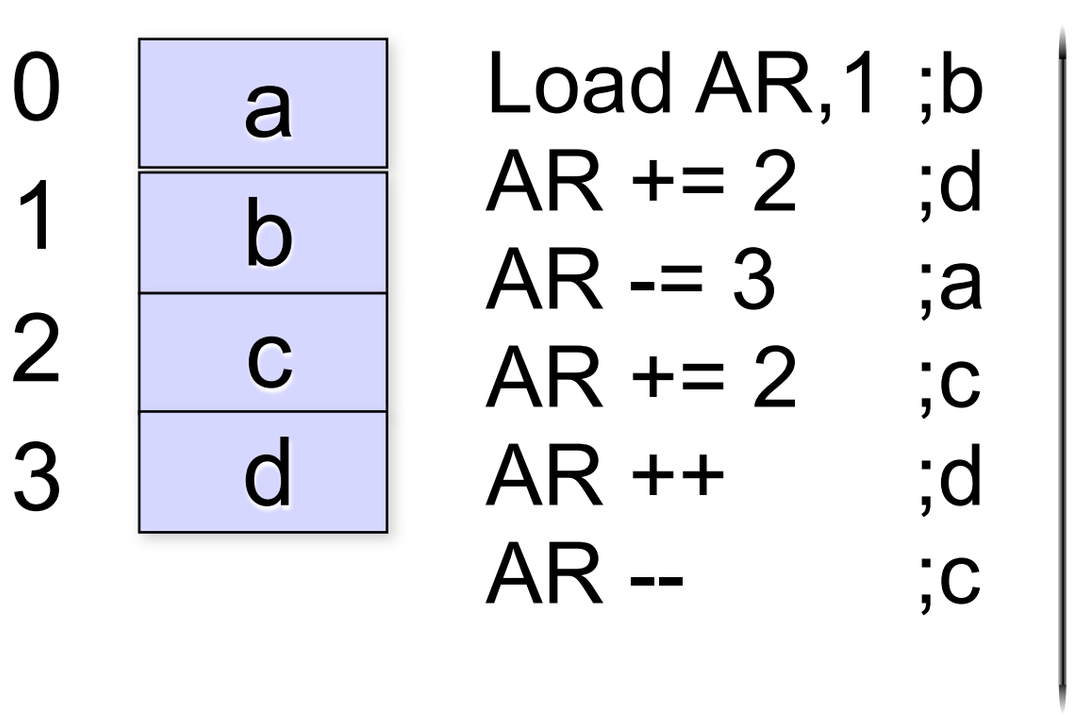
SOA example
- Effect of optimised memory layout -

Variables in a basic block:

$V = \{a, b, c, d\}$

Access sequence:

$S = (b, d, a, c, d, c)$



cost: 4

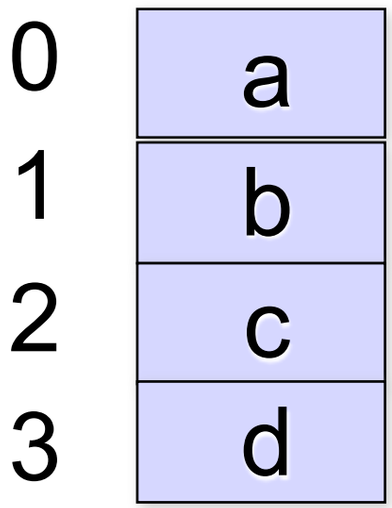
SOA example
- Effect of optimised memory layout -

Variables in a basic block:

$V = \{a, b, c, d\}$

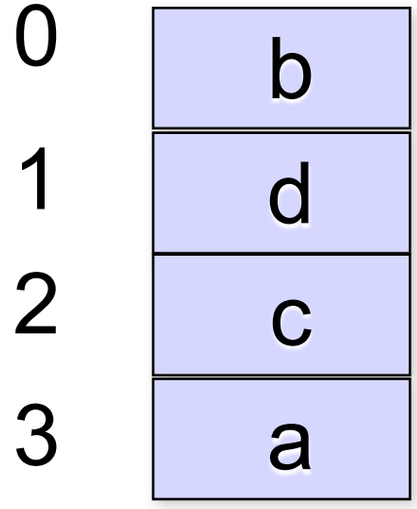
Access sequence:

$S = (b, d, a, c, d, c)$




```
Load AR,1  ;b
AR += 2    ;d
AR -= 3    ;a
AR += 2    ;c
AR ++      ;d
AR --      ;c
```

cost: 4




```
Load AR,0  ;b
AR ++      ;d
AR +=2     ;a
AR --      ;c
AR --      ;d
AR ++      ;c
```

cost: 2

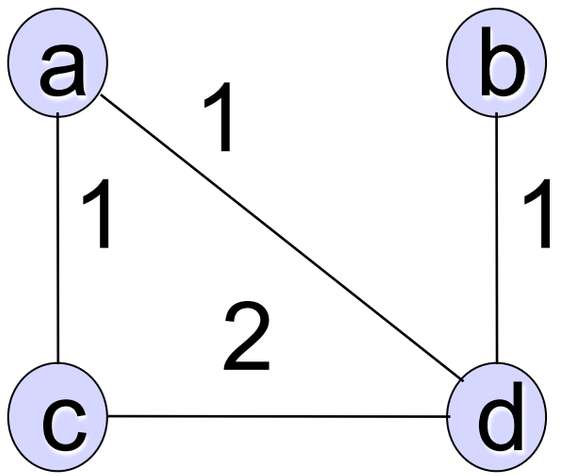
SOA example: Access sequence, access graph and Hamiltonian paths

access sequence: b d a c d c

[Bartley, 1992; Liao, 1995]

SOA example: Access sequence, access graph and Hamiltonian paths

access sequence: b d a c d c

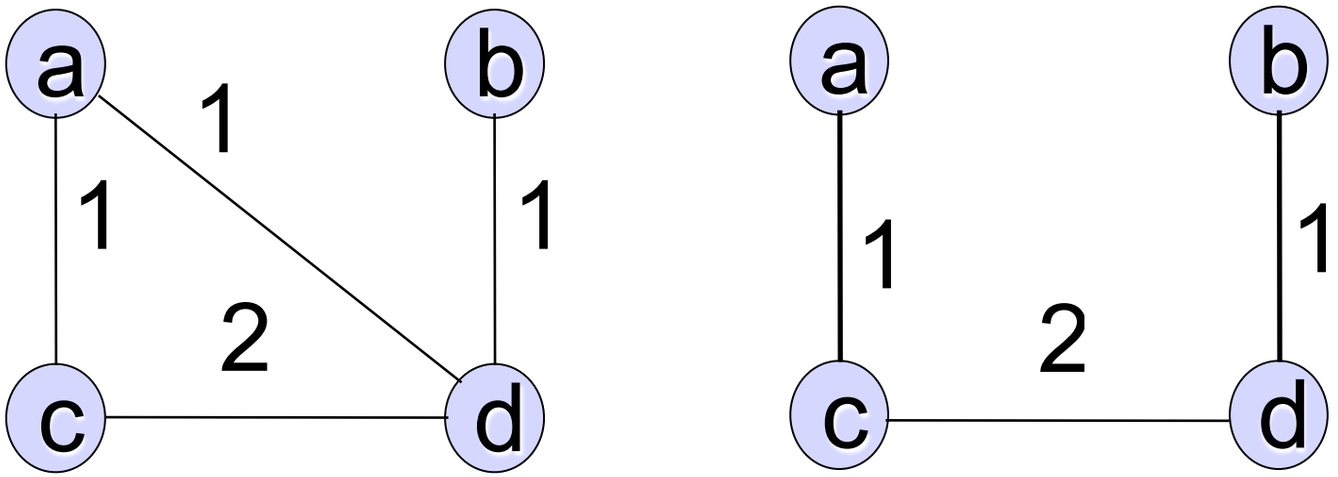


access graph

[Bartley, 1992; Liao, 1995]

SOA example: Access sequence, access graph and Hamiltonian paths

access sequence: b d a c d c



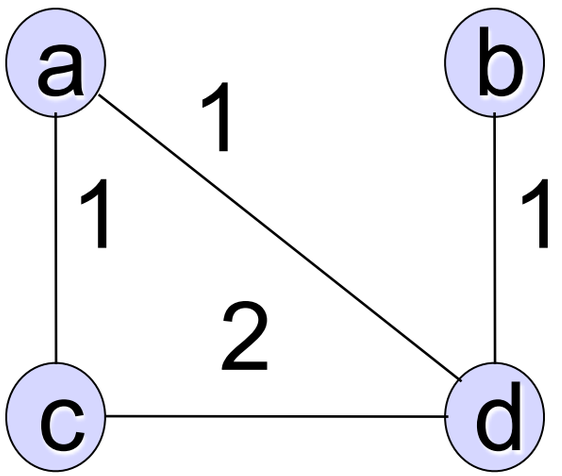
access graph

maximum weighted path= max. weighted Hamilton path covering (MWHC)

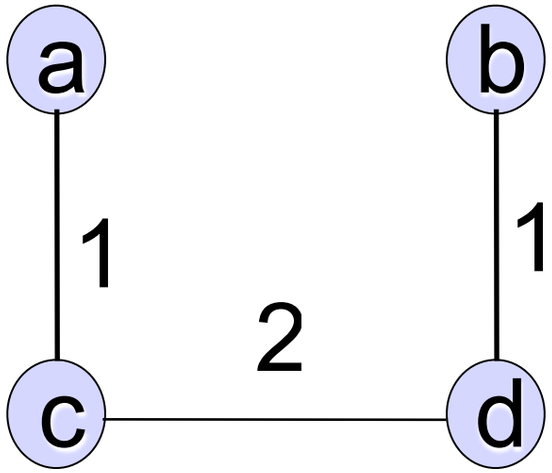
[Bartley, 1992; Liao, 1995]

SOA example: Access sequence, access graph and Hamiltonian paths

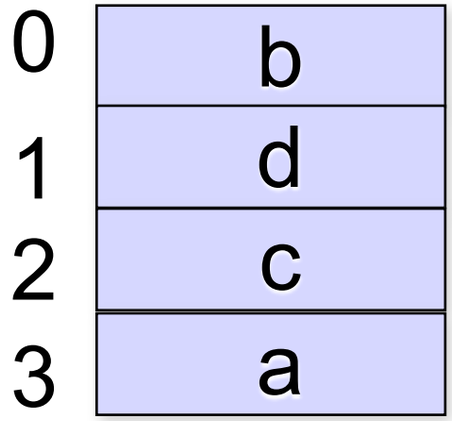
access sequence: b d a c d c



access graph



maximum weighted path= max. weighted Hamilton path covering (MWHC)

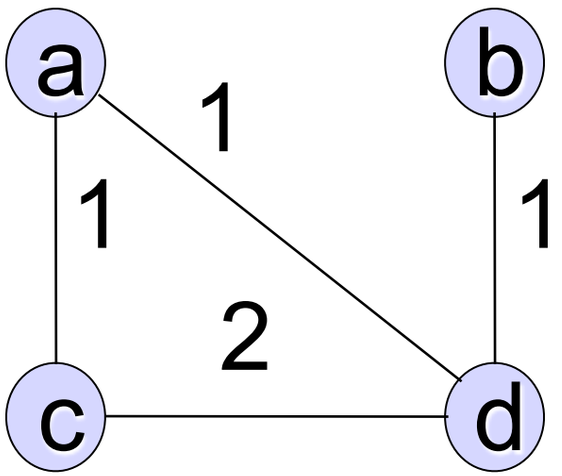


memory layout

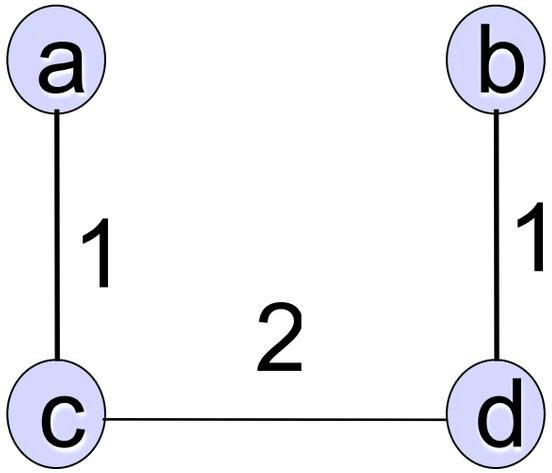
[Bartley, 1992; Liao, 1995]

SOA example: Access sequence, access graph and Hamiltonian paths

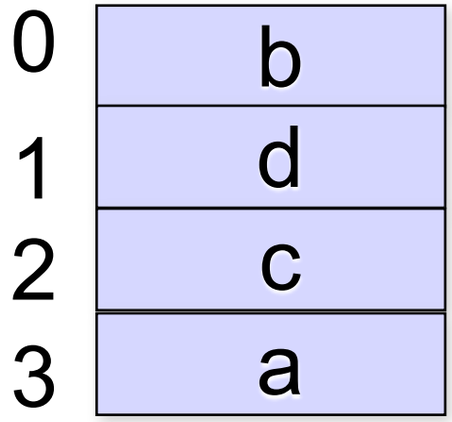
access sequence: b d a c d c



access graph



maximum weighted path= max. weighted Hamilton path covering (MWHC)



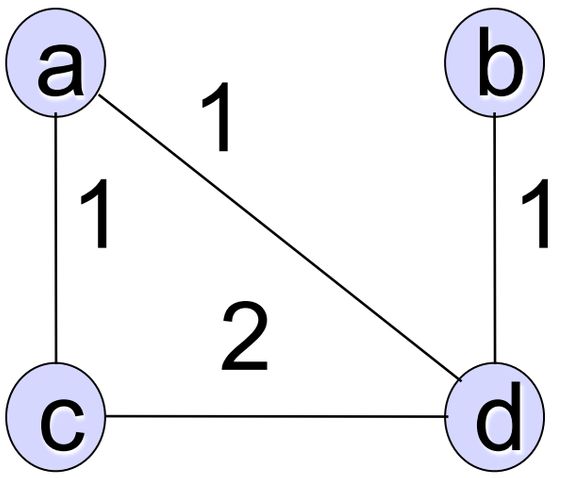
memory layout

SOA used as a building block for more complex situations

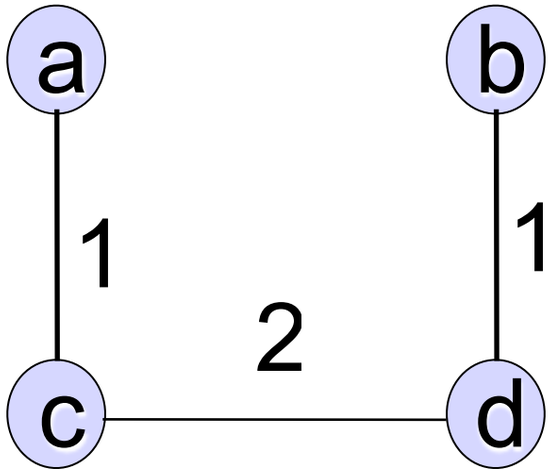
[Bartley, 1992; Liao, 1995]

SOA example: Access sequence, access graph and Hamiltonian paths

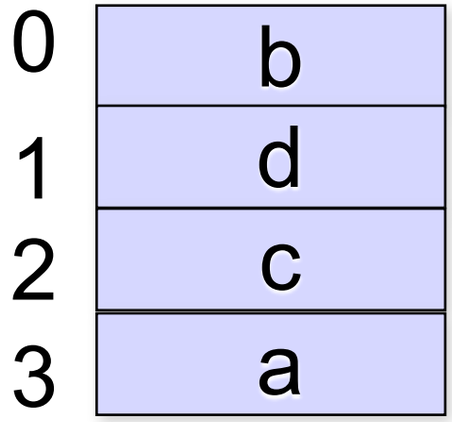
access sequence: b d a c d c



access graph



maximum weighted path= max. weighted Hamilton path covering (MWHC)



memory layout

SOA used as a building block for more complex situations

⇒ significant interest in good SOA algorithms

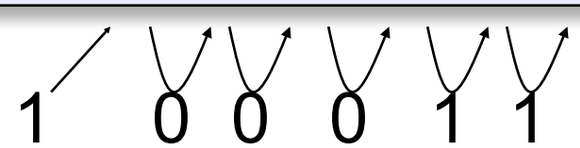
[Bartley, 1992; Liao, 1995]

Naïve SOA

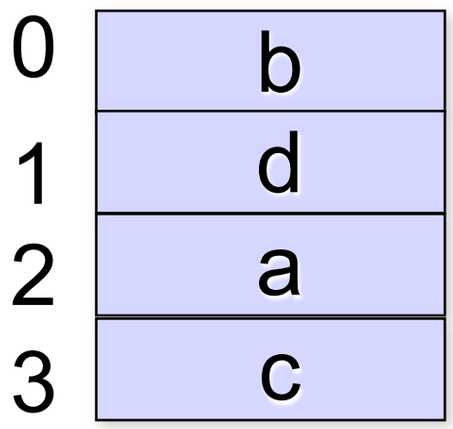
Nodes are added in the order
in which they are used in the program.

Example:

Access sequence: $S = (b, d, a, c, d, c)$



b



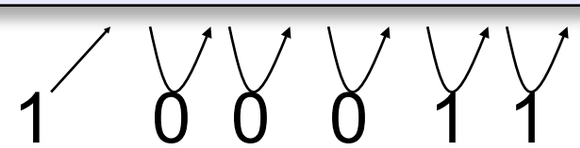
memory layout

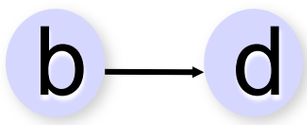
Naïve SOA

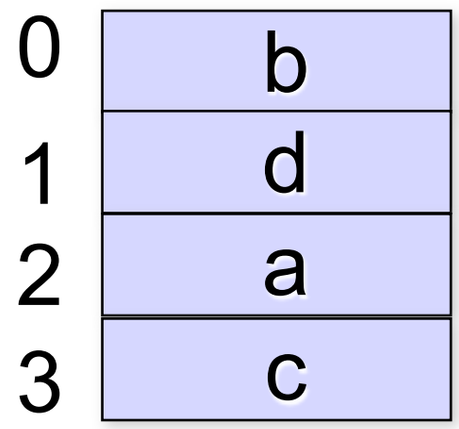
Nodes are added in the order
in which they are used in the program.

Example:

Access sequence: $S = (b, d, a, c, d, c)$







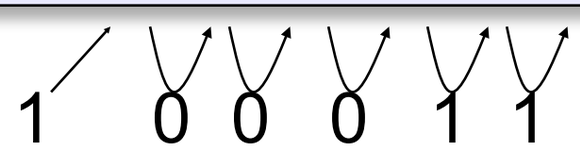
memory layout

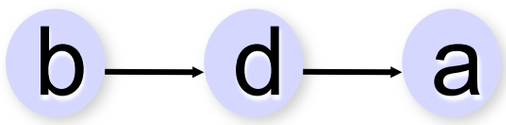
Naïve SOA

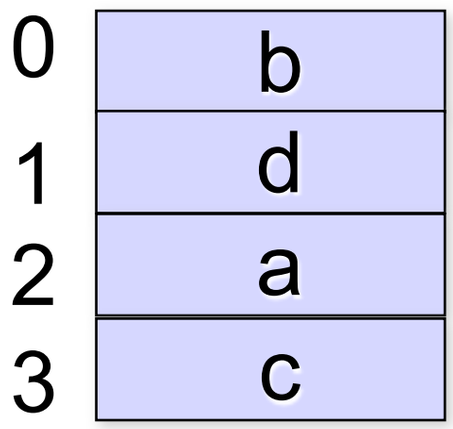
Nodes are added in the order
in which they are used in the program.

Example:

Access sequence: $S = (b, d, a, c, d, c)$







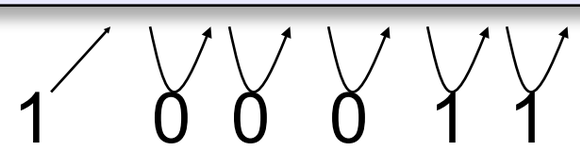
memory layout

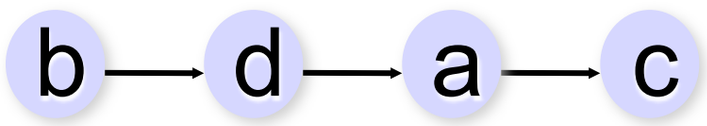
Naïve SOA

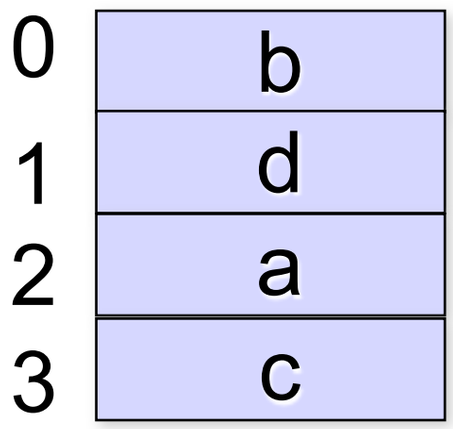
Nodes are added in the order
in which they are used in the program.

Example:

Access sequence: $S = (b, d, a, c, d, c)$







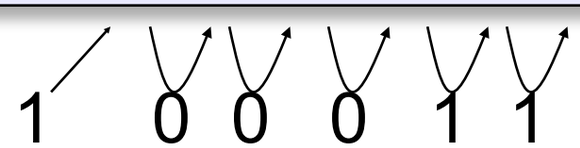
memory layout

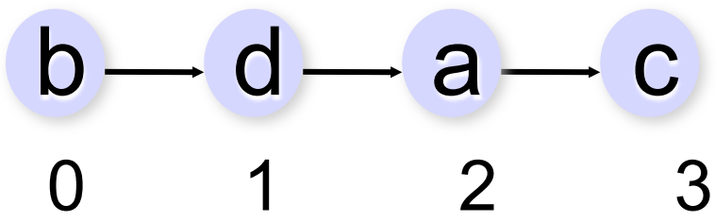
Naïve SOA

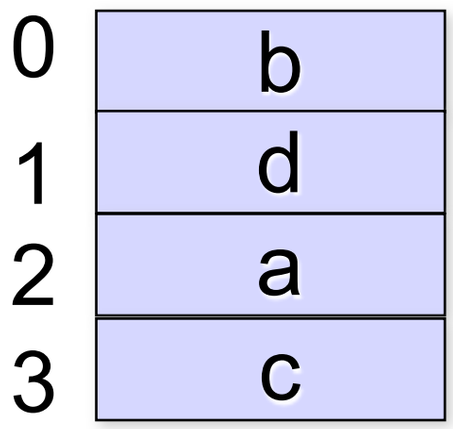
Nodes are added in the order
in which they are used in the program.

Example:

Access sequence: $S = (b, d, a, c, d, c)$







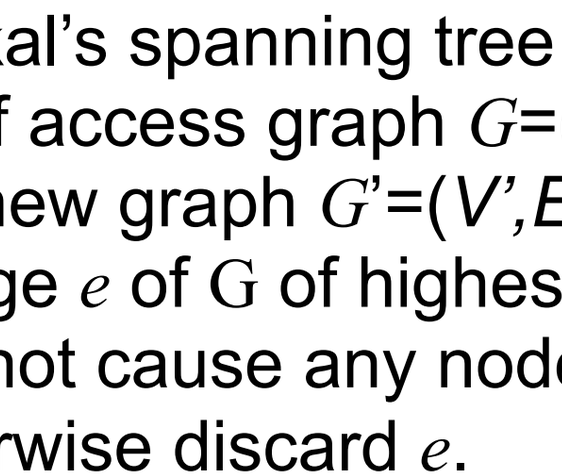
memory layout

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

G

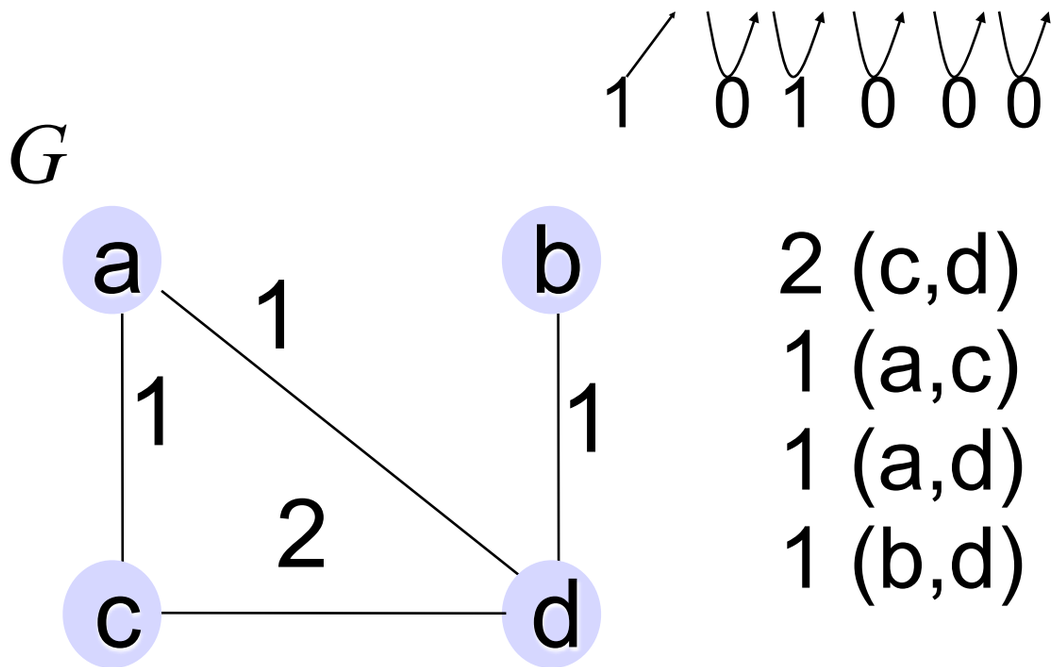


Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*



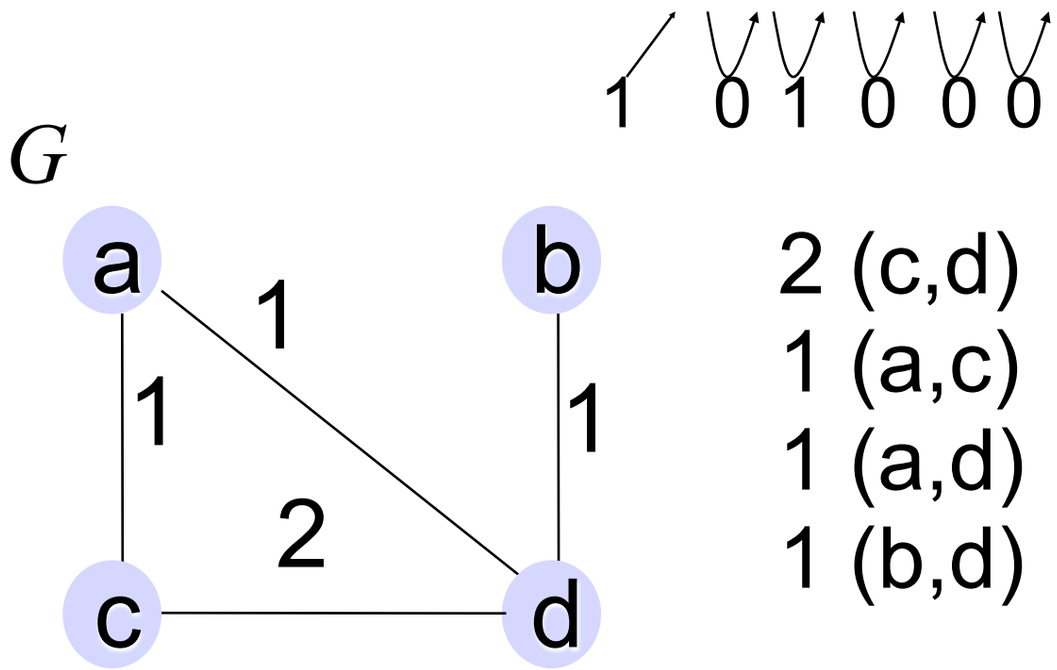
2 (c,d)
1 (a,c)
1 (a,d)
1 (b,d)

Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*



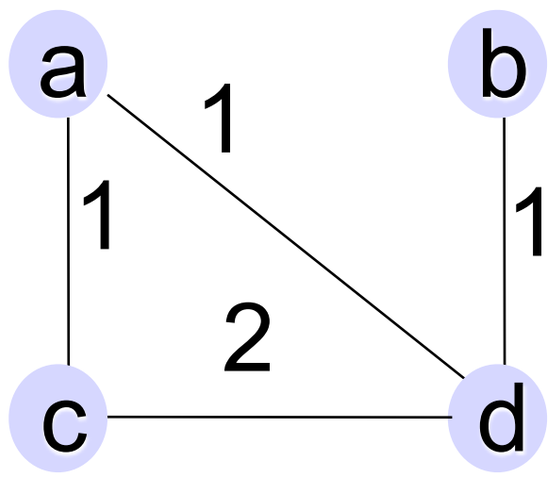
Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

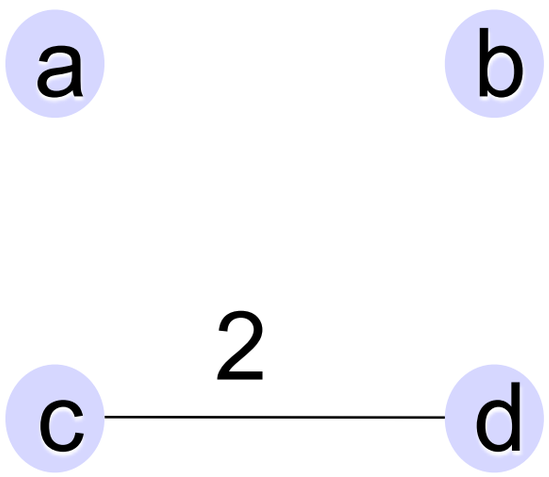
1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

G



2 (c,d)
1 (a,c)
1 (a,d)
1 (b,d)

G'

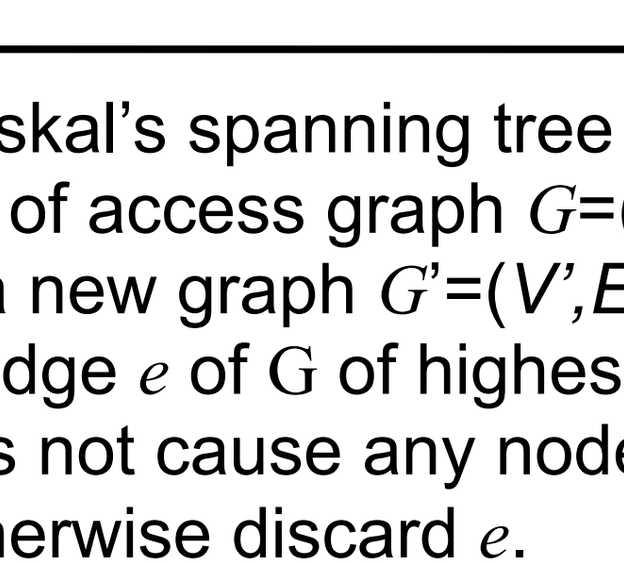


Implicit edges of weight 0 for all unconnected nodes.

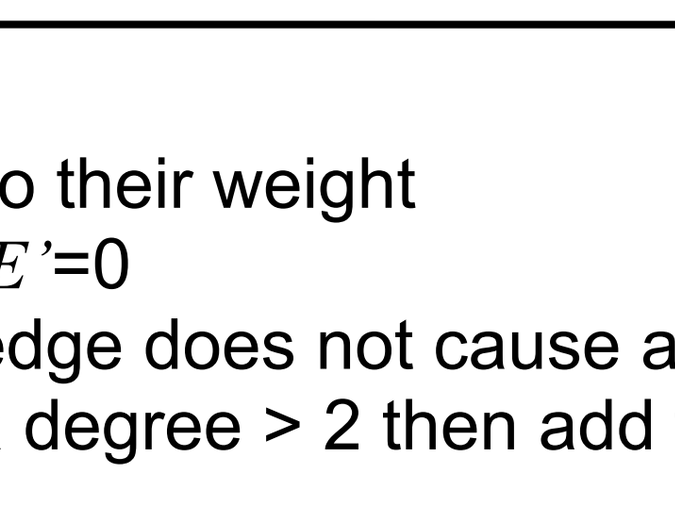
Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*



2 (c,d)
1 (a,c)
1 (a,d)
1 (b,d)



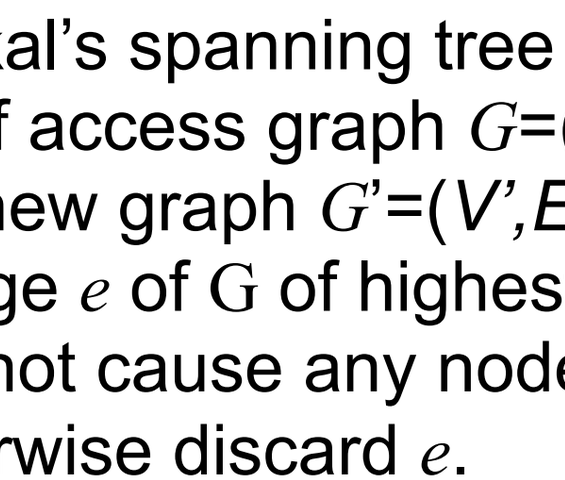
Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

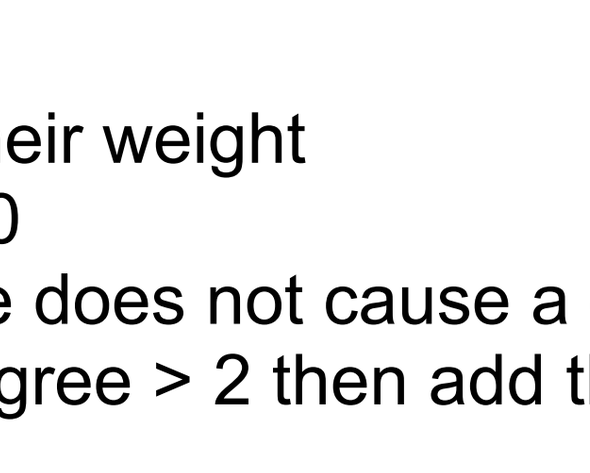
1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

G



2 (c,d)
1 (a,c)
1 (a,d)
1 (b,d)

G'



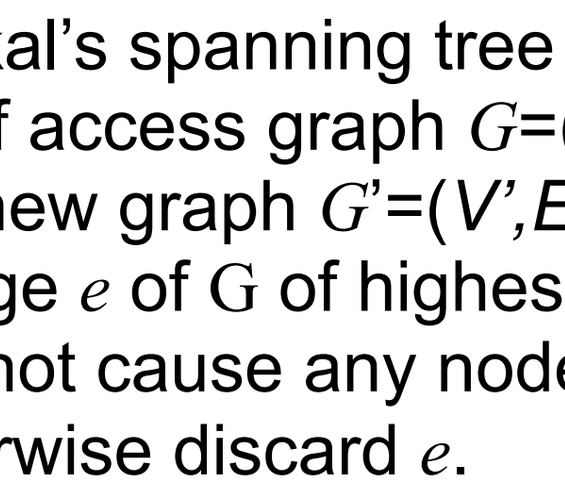
Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

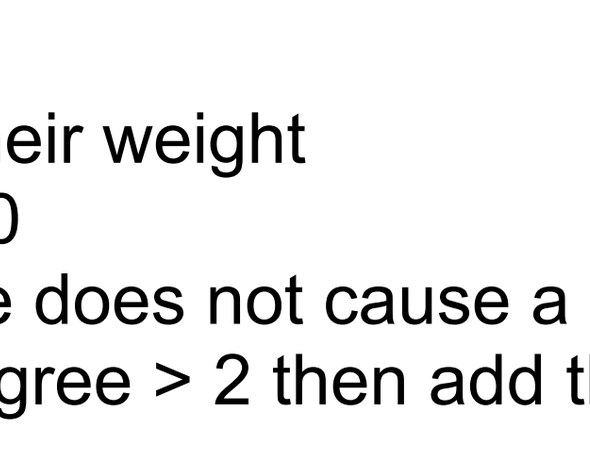
1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

G



2 (c,d)
1 (a,c)
1 (a,d)
1 (b,d)

G'

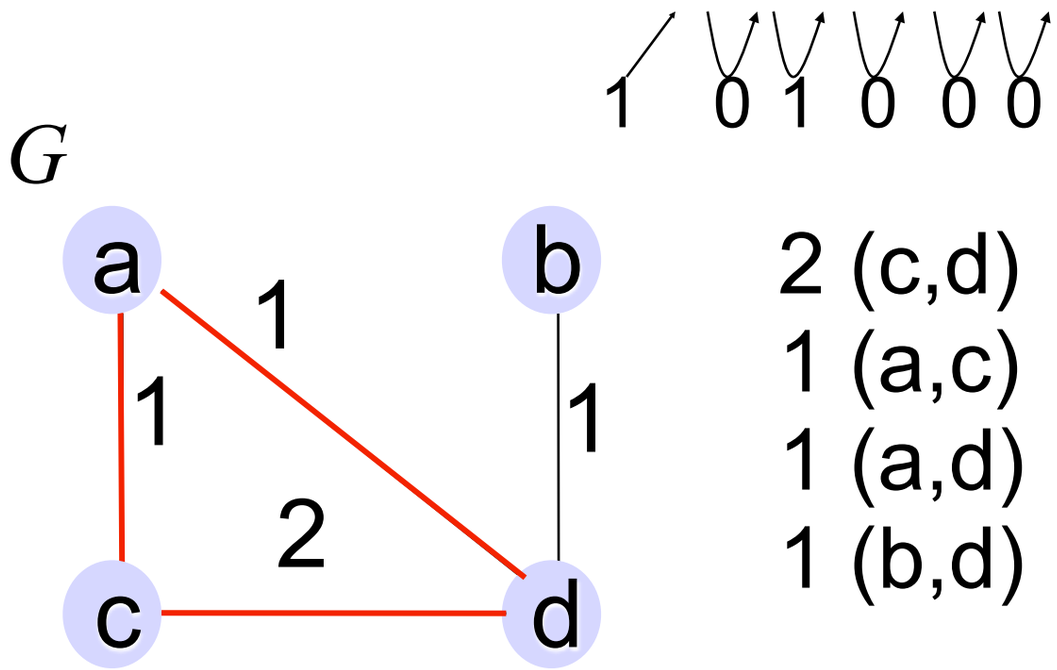


Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

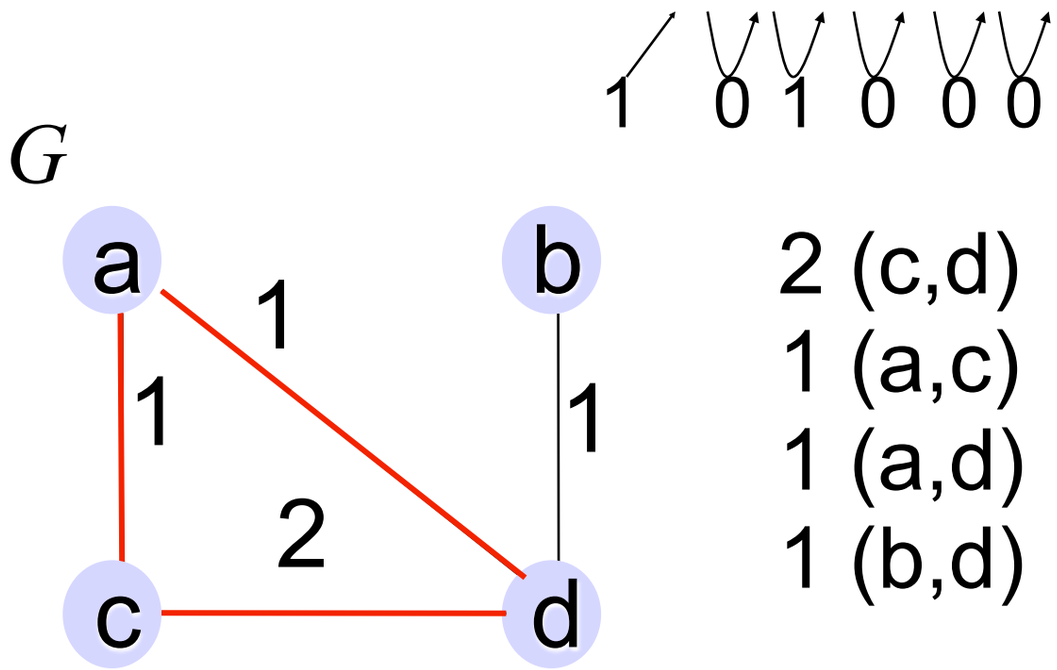


Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

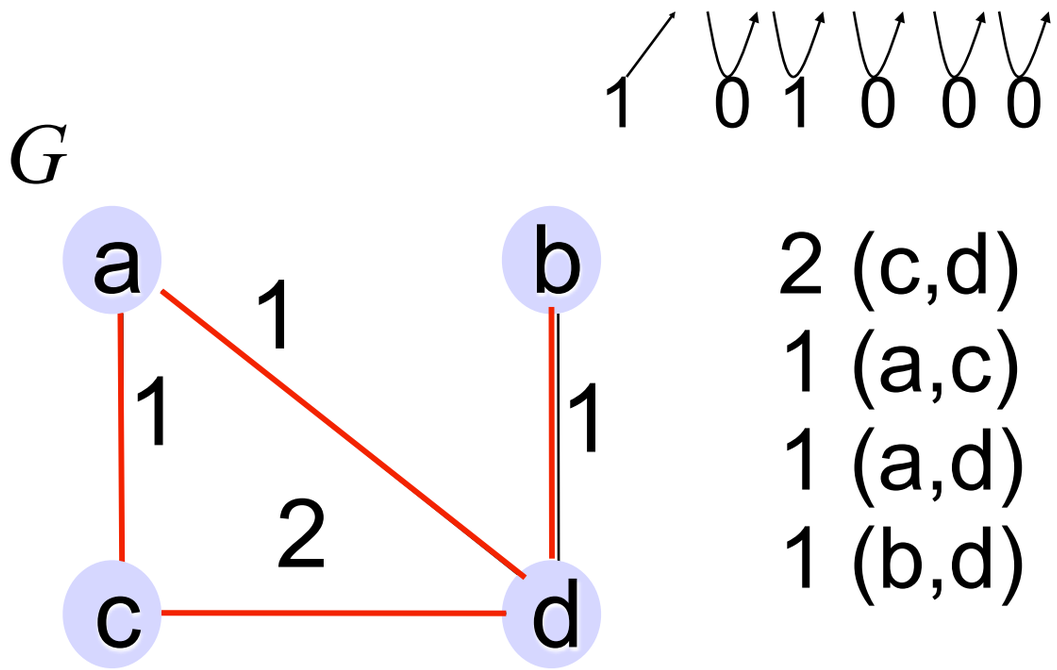


Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm

Similar to Kruskal's spanning tree algorithms:

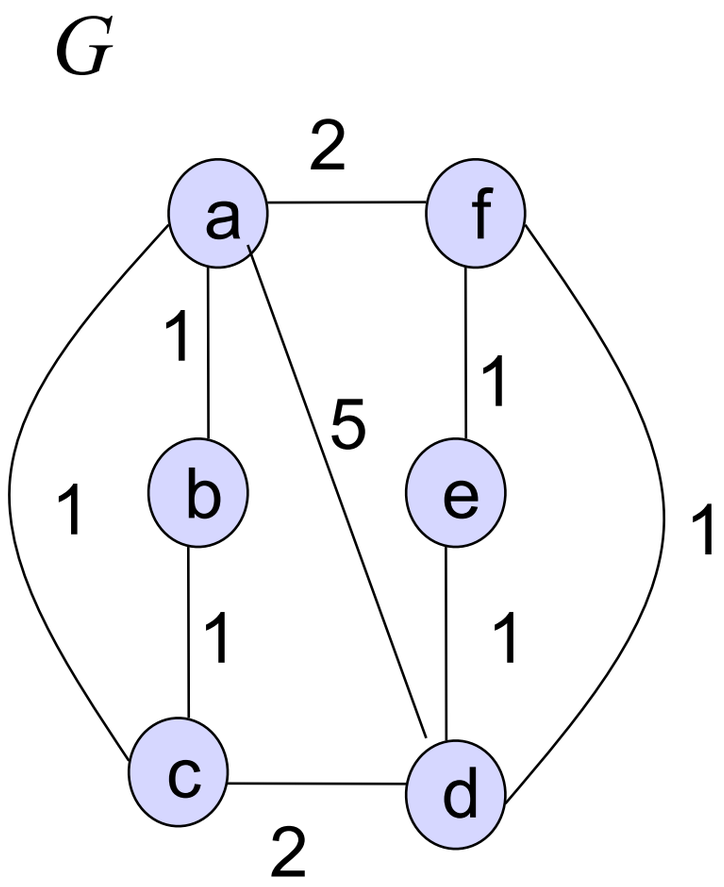
1. Sort edges of access graph G=(V,E) according to their weight
2. Construct a new graph G'=(V',E'), starting with E'=0
3. Select an edge e of G of highest weight; If this edge does not cause a cycle in G' and does not cause any node in G' to have a degree > 2 then add this node to E' otherwise discard e.
4. Goto 3 as long as not all edges from G have been selected
   and as long as G' has less than the maximum number of edges (|V|-1).
   Example: Access sequence: *S=(b, d, a, c, d, c)*

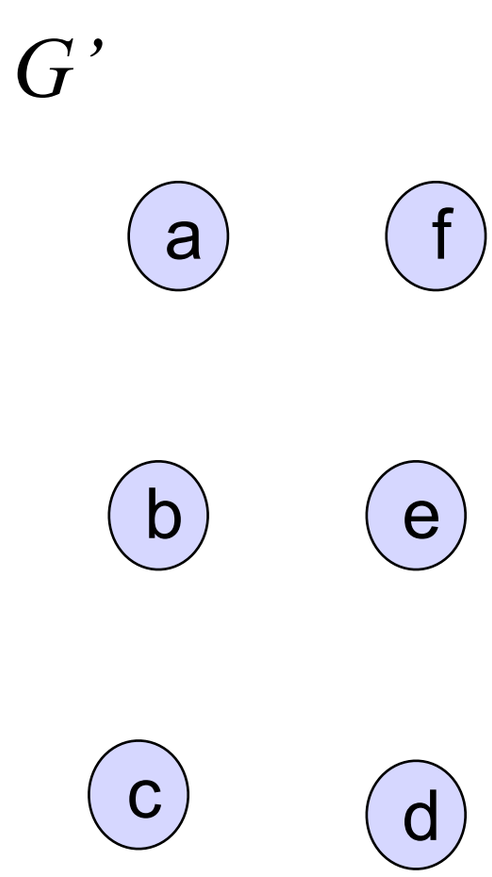


Implicit edges of weight 0 for all unconnected nodes.

Liao's algorithm on a more complex graph

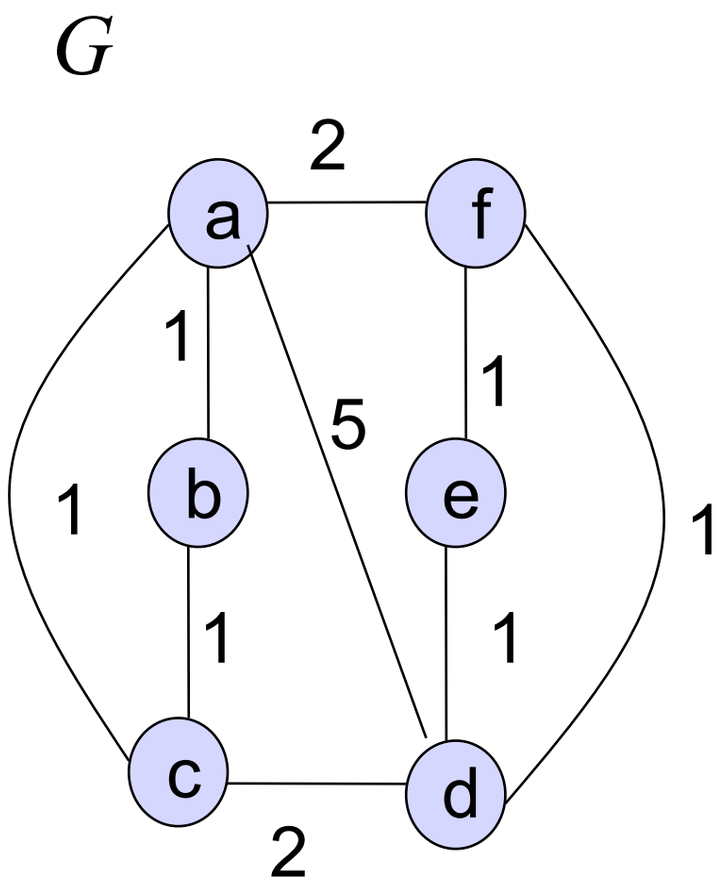
a b c d e f a d a d a c d f a d

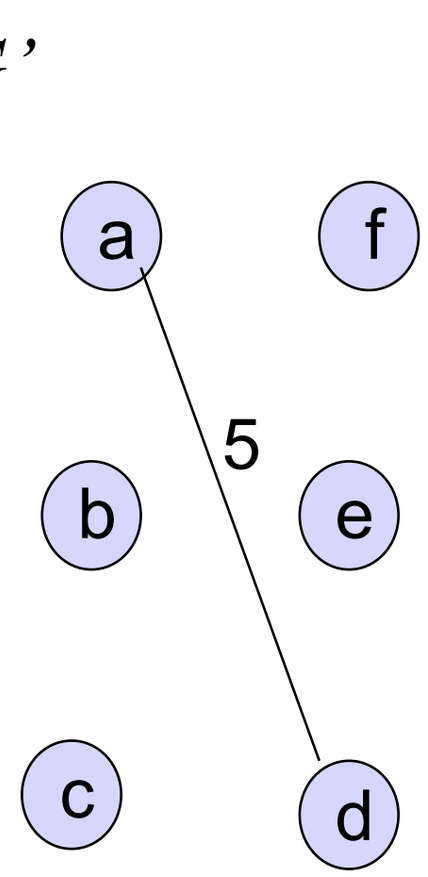




Liao's algorithm on a more complex graph

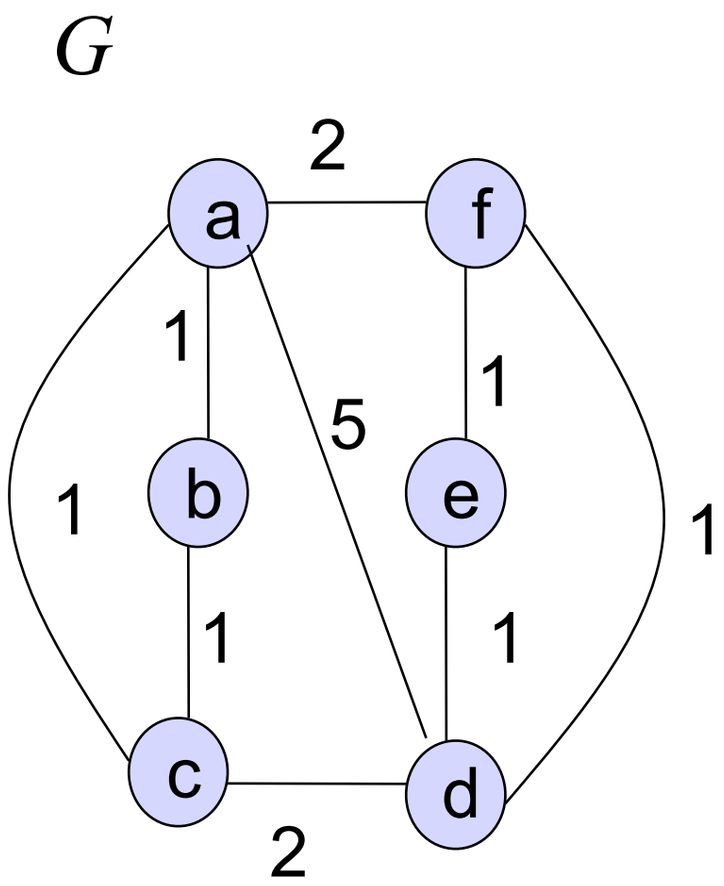
a b c d e f a d a d a c d f a d

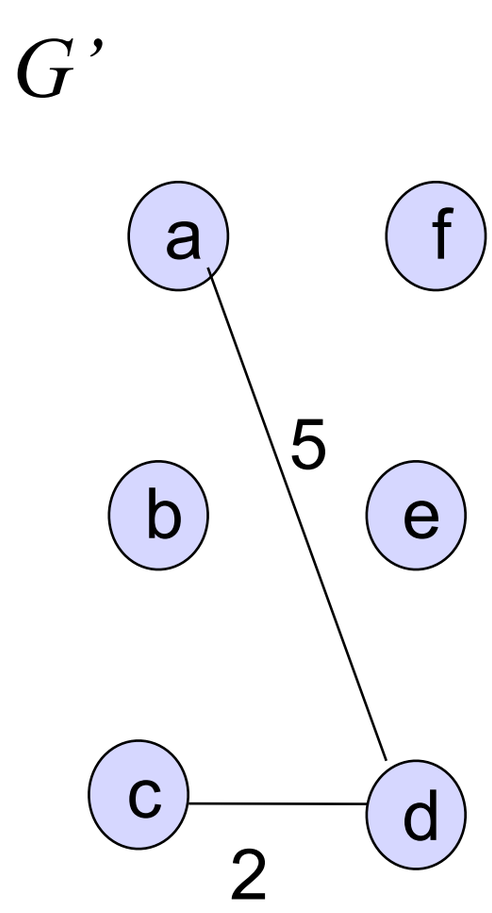




Liao's algorithm on a more complex graph

a b c d e f a d a d a c d f a d

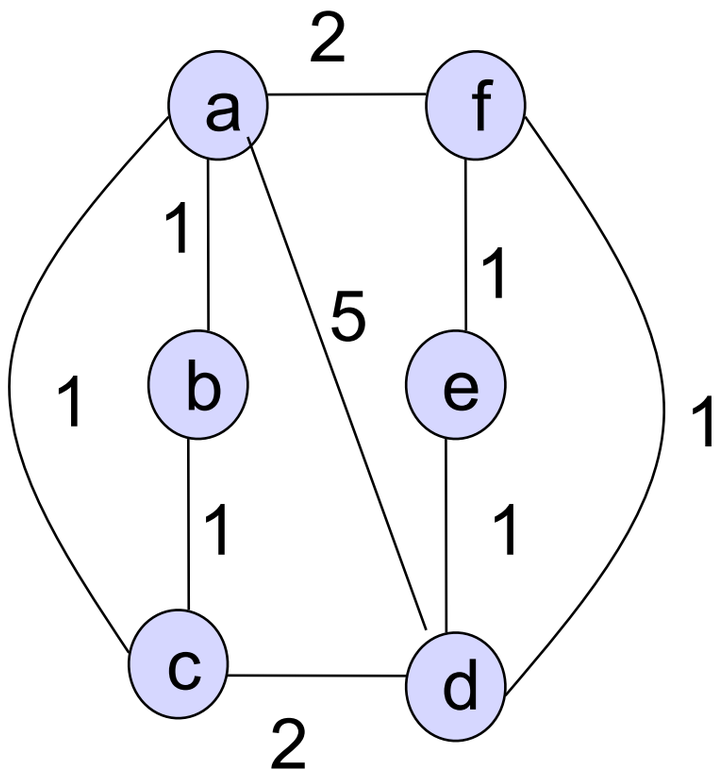




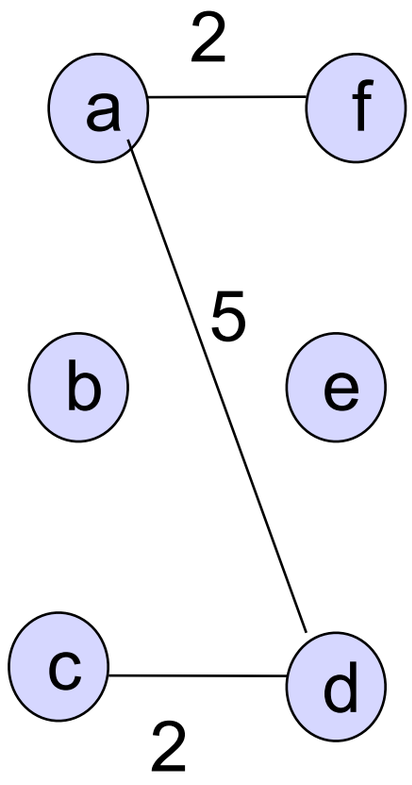
Liao's algorithm on a more complex graph

a b c d e f a d a d a c d f a d

G



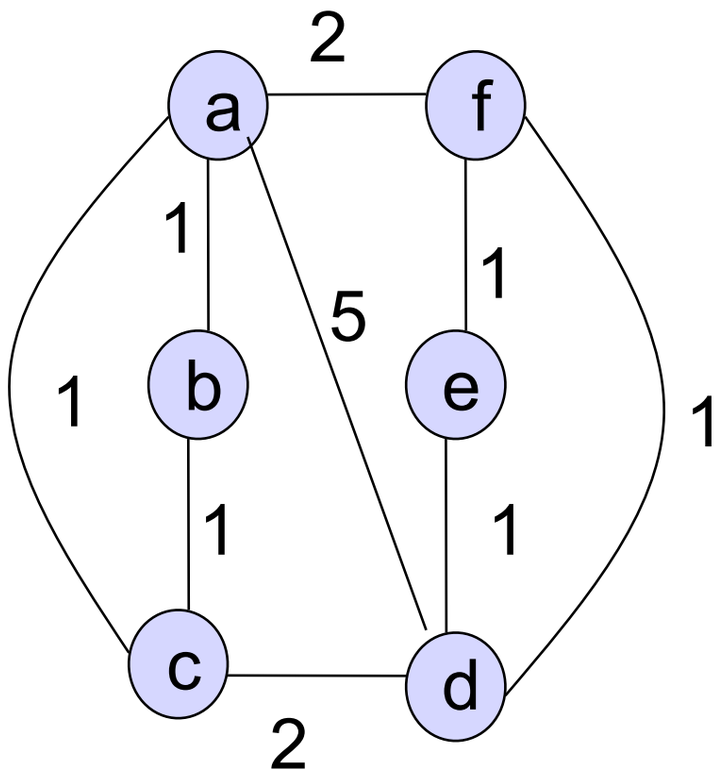
G'



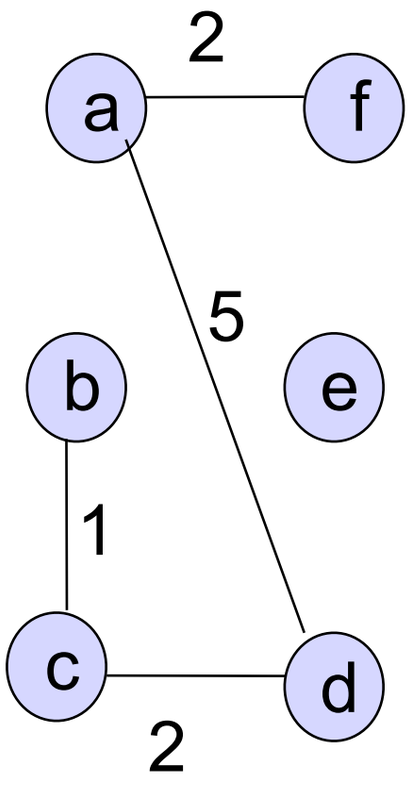
Liao's algorithm on a more complex graph

a b c d e f a d a d a c d f a d

G



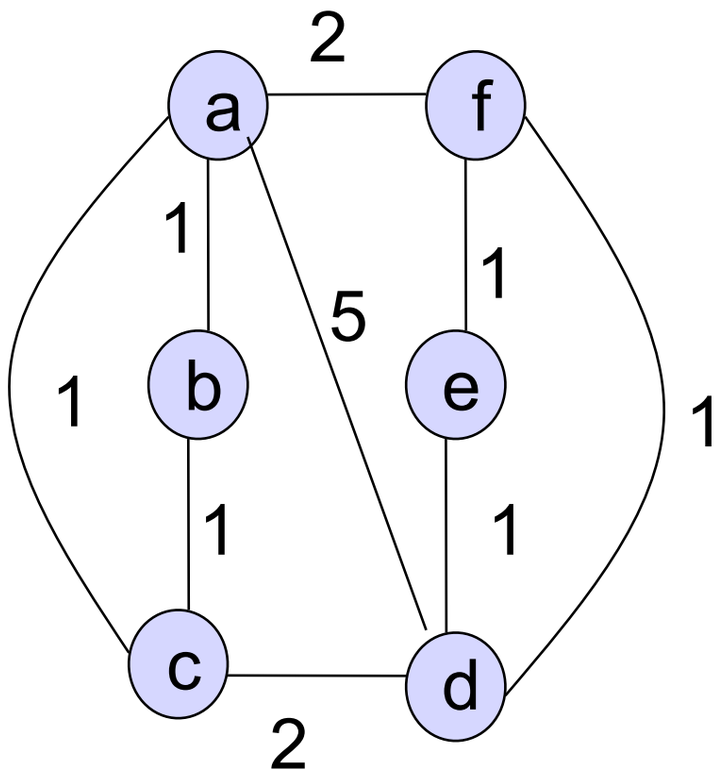
G'



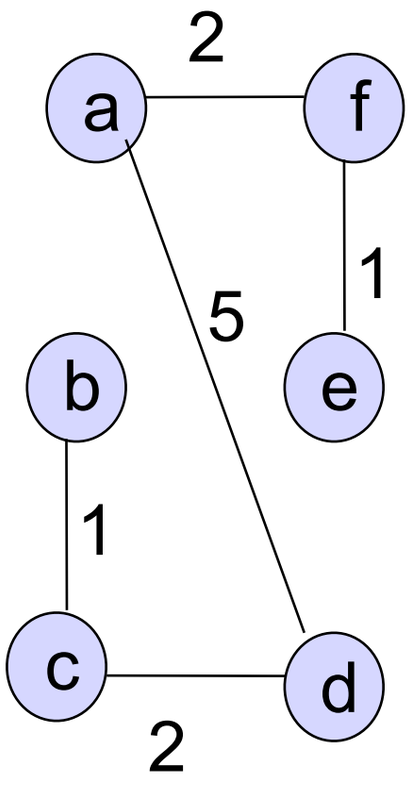
Liao's algorithm
on a more complex graph

a b c d e f a d a d a c d f a d

G

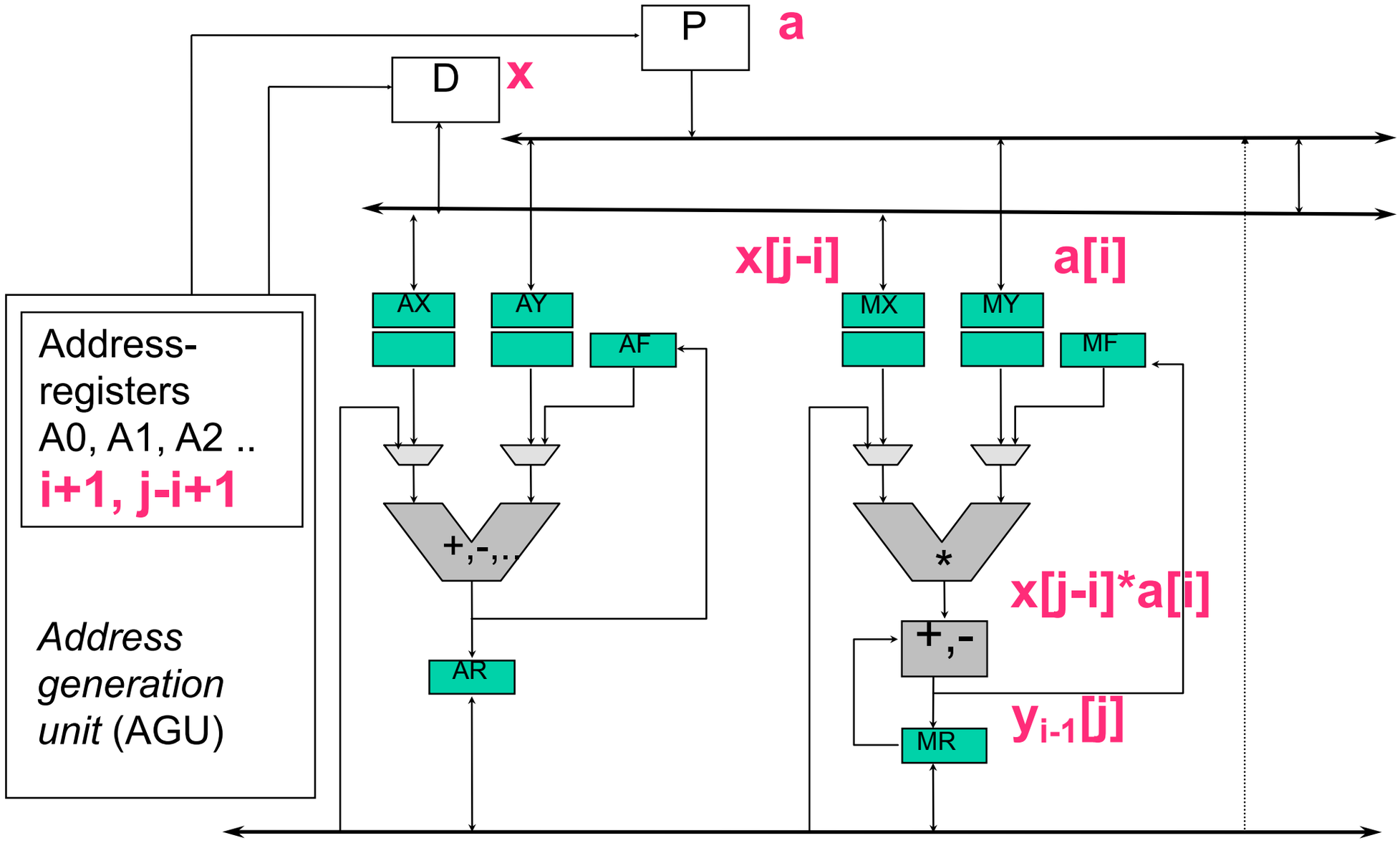


G'



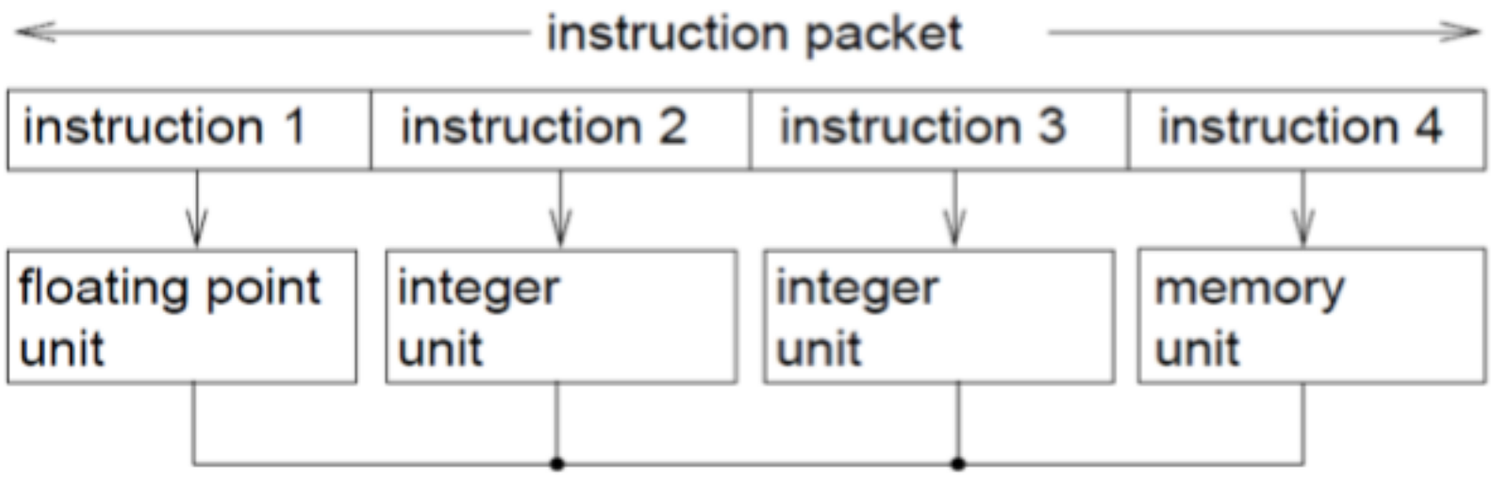
Exploitation of instruction level parallelism (ILP)

Several transfers in the same cycle:



Exploiting ILP

- Normally looked at as a way if improving performance
- However on vliw and dsp architectures
  - Also helps reduce code size
- If the parallel units are not doing anything
  - then have to fill with a nop



Exploitation of instruction level parallelism (ILP)

```
1: MR := MR+(MX*MY);
2: MX:=D[A1];
3: MY:=P[A2];
4: A1- -;
5: A2++;
6: D[0]:= MR;
.....
```

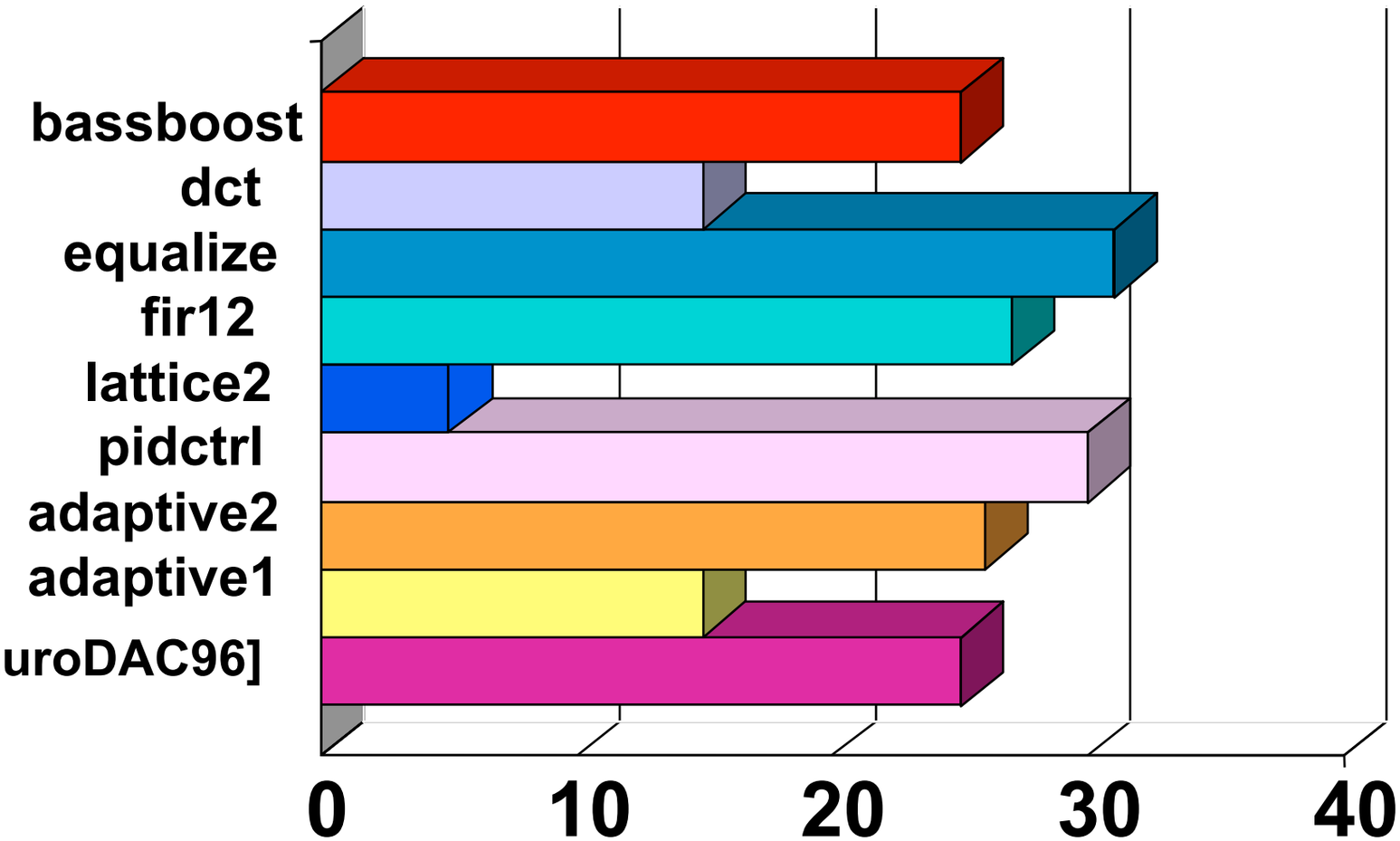
```
1´: MR := MR+(MX*MY), MX:=D[A1],
    MY:=P[A2], A1- -, A2++;
2´: D[0]:= MR;
```

Normally tackled using dependence graph and list scheduling

- Modelling of possible parallelism using n-ary compatibility relation, e.g. ~(1,2,3,4,5)
- Generation of integer programming (IP)- model (max. 50 statements/model)
- Using standard-IP-solver to solve equations

Exploitation of instruction level parallelism (ILP)

Results obtained through integer programming:

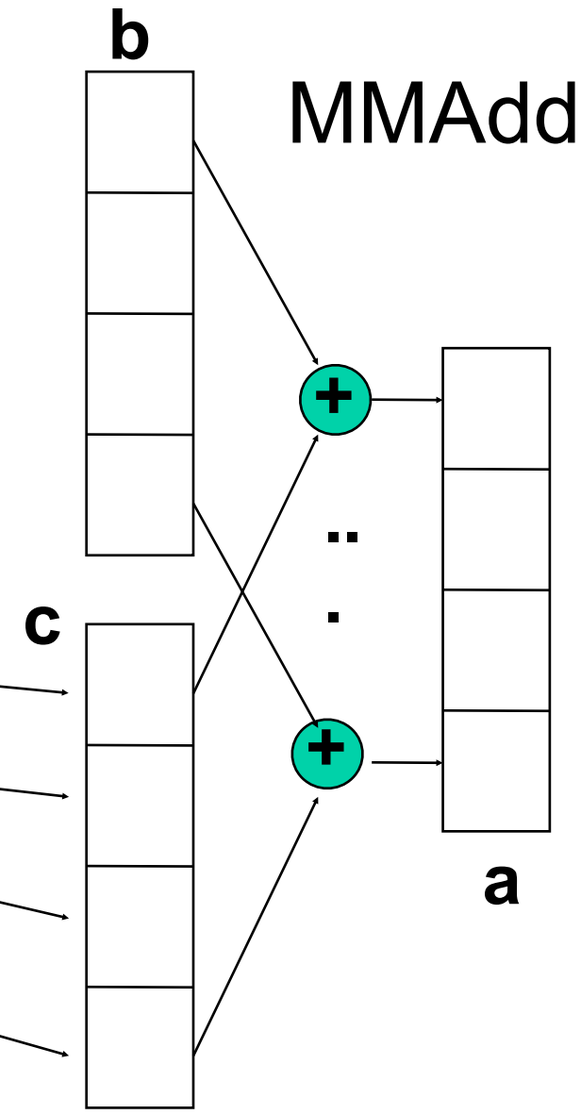


Compaction times: 2 .. 35 sec

Exploitation of Multimedia Instructions

```
FOR i:=0 TO n DO
 a[i] = b[i] + c[i]
```

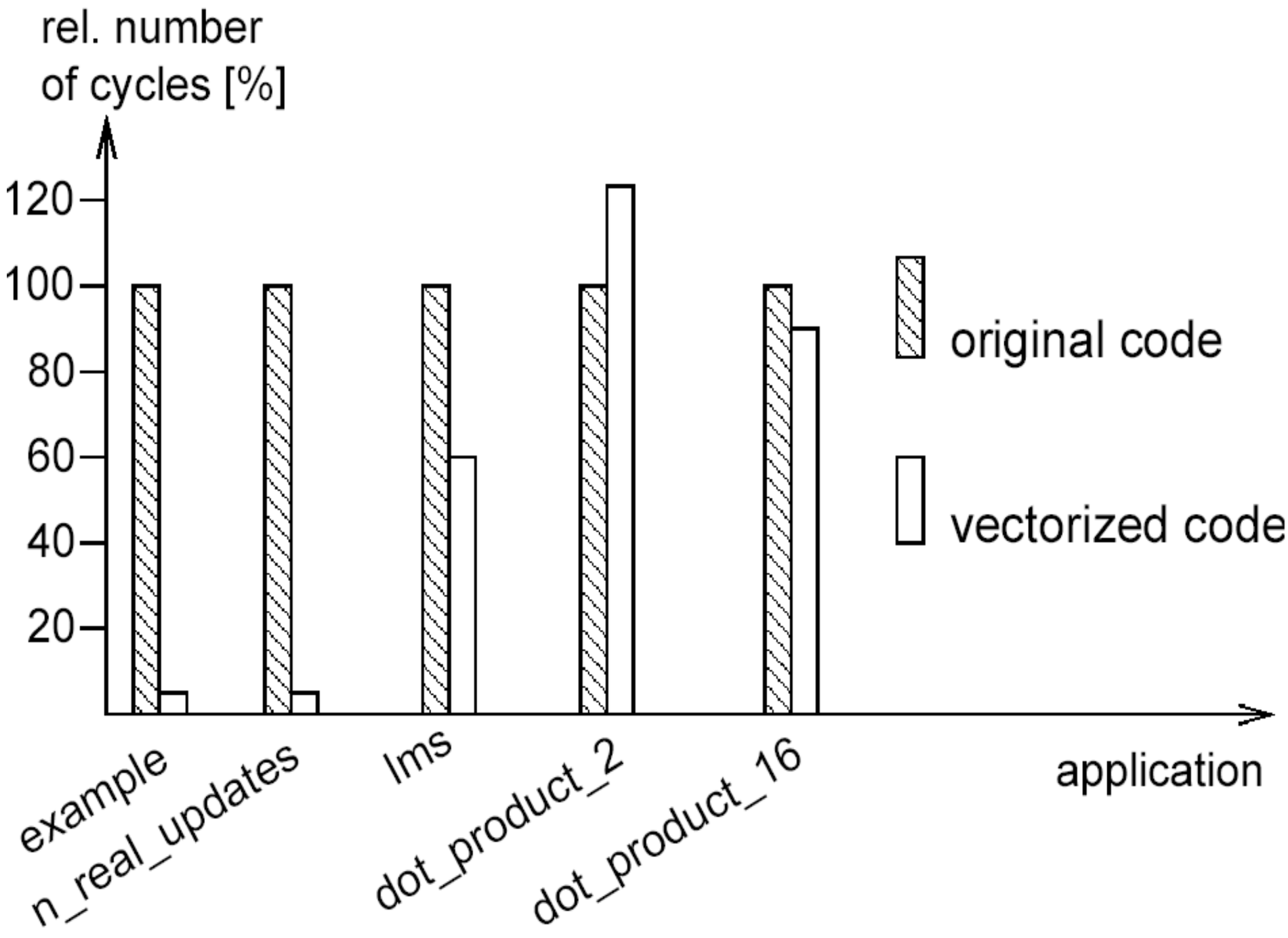
```
FOR i:=0 STEP 4 TO n DO
 a[i  ]=b[i  ]+c[i ];
 a[i+1]=b[i+1]+c[i+1];
 a[i+2]=b[i+2]+c[i+2];
 a[i+3]=b[i+3]+c[i+3];
```



MMAdd (4 x 8/16 bit)

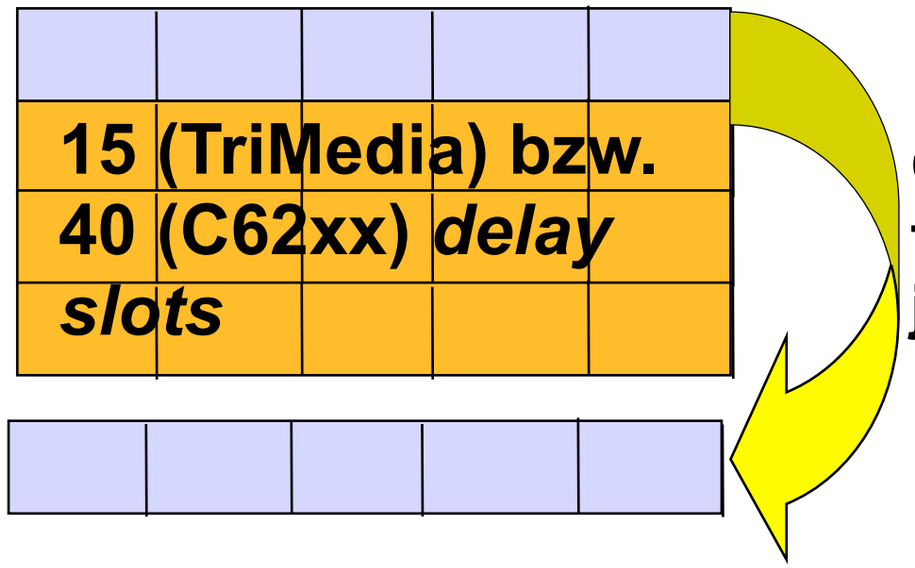
A form of limited vectorisation. Normally performed in code generation stage rather then at restructure stage

Improvements for M3 DSP due to vectorization



Avoiding branch delay using predication

Large *branch delay penalty:*



Avoiding this penalty:
predicated execution:
[c] *instruction*
c=*true*: instruction executed
c=*false*: effectively NOOP

Realisation of *if-statements*

with conditional jumps or with *predicated execution:*

```
if (c)
{ a = x + y;
  b = x + z;
}
else
{ a = x - y;
  b = x - z;
}
```

Cond. instructions:

```
    [c]  ADD x,y,a
|| [c]  ADD x,z,b
|| [!c] SUB  x,y,a
|| [!c] SUB  x,z,b
```

1 cycle

Cost of implementation methods for IF-Statements

Sourcecode: if (c1) {t1; if (c2) t2}

No precondition (no enclosing IF or enclosing IFs implemented with cond. jumps)

1. **Conditional jump:**
   BNE c1, L;
   t1;
   L: ...

2. **Conditional Instruction:**
   [c1] t1

Precondition (enclosing IF not implemented with conditional jumps)

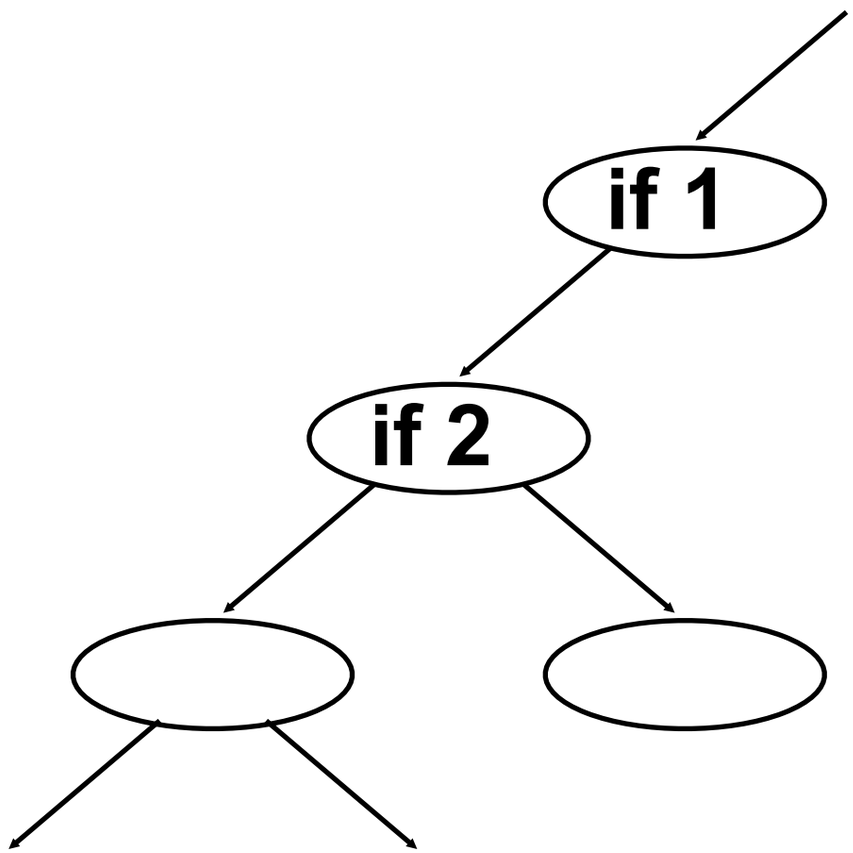
3. **Conditional jump :**
   [c1] c:=c2
   [~c1] c:=0
   BNE c, L;
   t2;
   L: ...

4. **Conditional Instruction :**
   [c1] c:=c2
   [~c1] c:=0
   [c] t2

Additional computations to compute effective condition c

Optimization for nested IF-statements

Goal: compute fastest implementation for all IF-statements

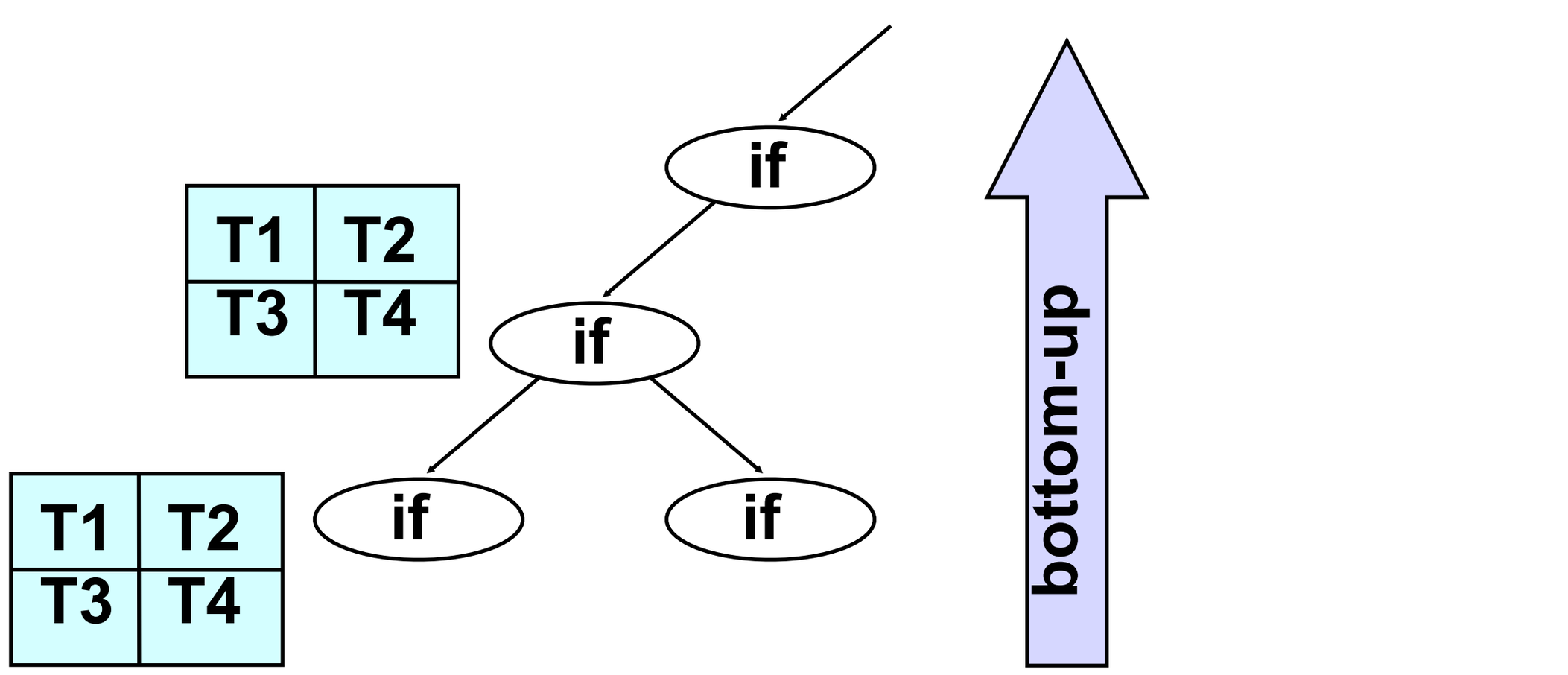


- Selection of fastest implementation for if-1 requires knowledge of how fast if-2 can be implemented.
- Execution time of if-2 depends on setup code, and, hence, also on how if 1 is implemented
- cyclic dependency!

Dynamic programming algorithm (phase 1)

For each if-statement compute 4 cost values:

- **T1 : cond. jump, no precondition**
- **T2 : cond. instructions, no precondition**
- **T3 : cond. jump, with precondition**
- **T4: cond. instructions, with precondition**

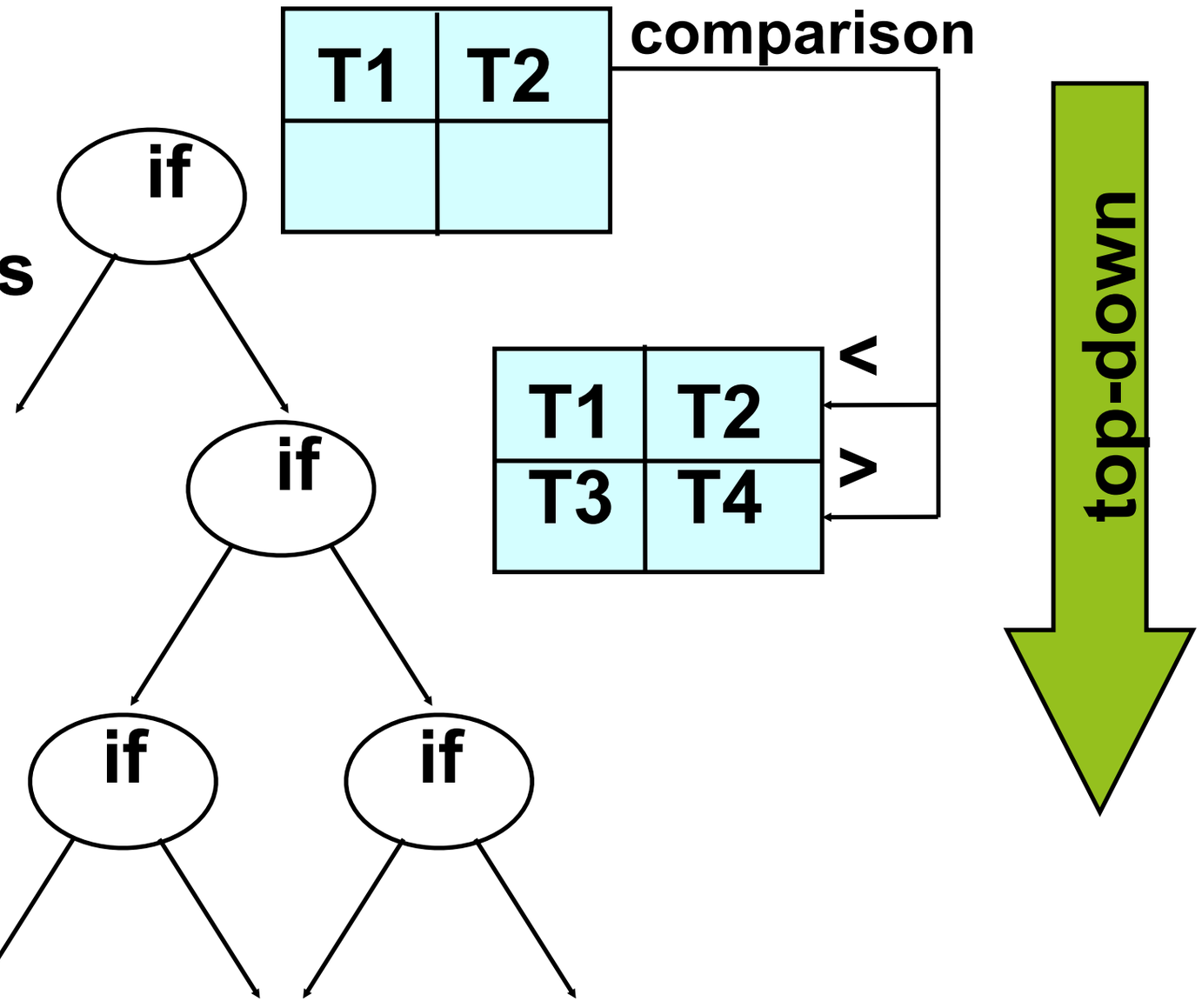


Dynamic programming (phase 2)

No precondition for top-level IF-statement.
Hence, comparison T1 < T2 suffices.

T1 < T2:
cond. branch faster, no precondition for nested IF-statements

T1 > T2:
cond. instructions faster, precondition for nested IF-state-ments



Results: TI C62xx

Runtimes (max) for 10 control-dominated examples

Example	Conditional jumps	Conditional instructions	Dynamic program.	Min (col. 2-5)	TI C compiler
1	21	11	11	11	15
2	12	13	13	12	13
3	26	21	22	21	27
4	9	12	12	9	10
5	26	30	24	24	21
6	32	23	23	23	30
7	57	173	49	49	51
8	39	244	30	30	41
9	28	27	27	27	29
10	27	30	30	27	28

Average gain: 12%

Function inlining: advantages and limitations

Advantage: low calling overhead

```
Function sq(c:integer)
 return:integer;
begin
 return c*c
end;
....
a=sq(b);
....
```

branching

```
push PC;
push b;
BRA sq;
 pull R1;
 mul R1,R1,R1;
 pull R2;
 push R1;
 BRA (R2)+1;
pull R1;
ST R1,a;
```

Inlining

```
....
LD R1,b;
MUL R1,R1,R1;
ST R1,a
....
```

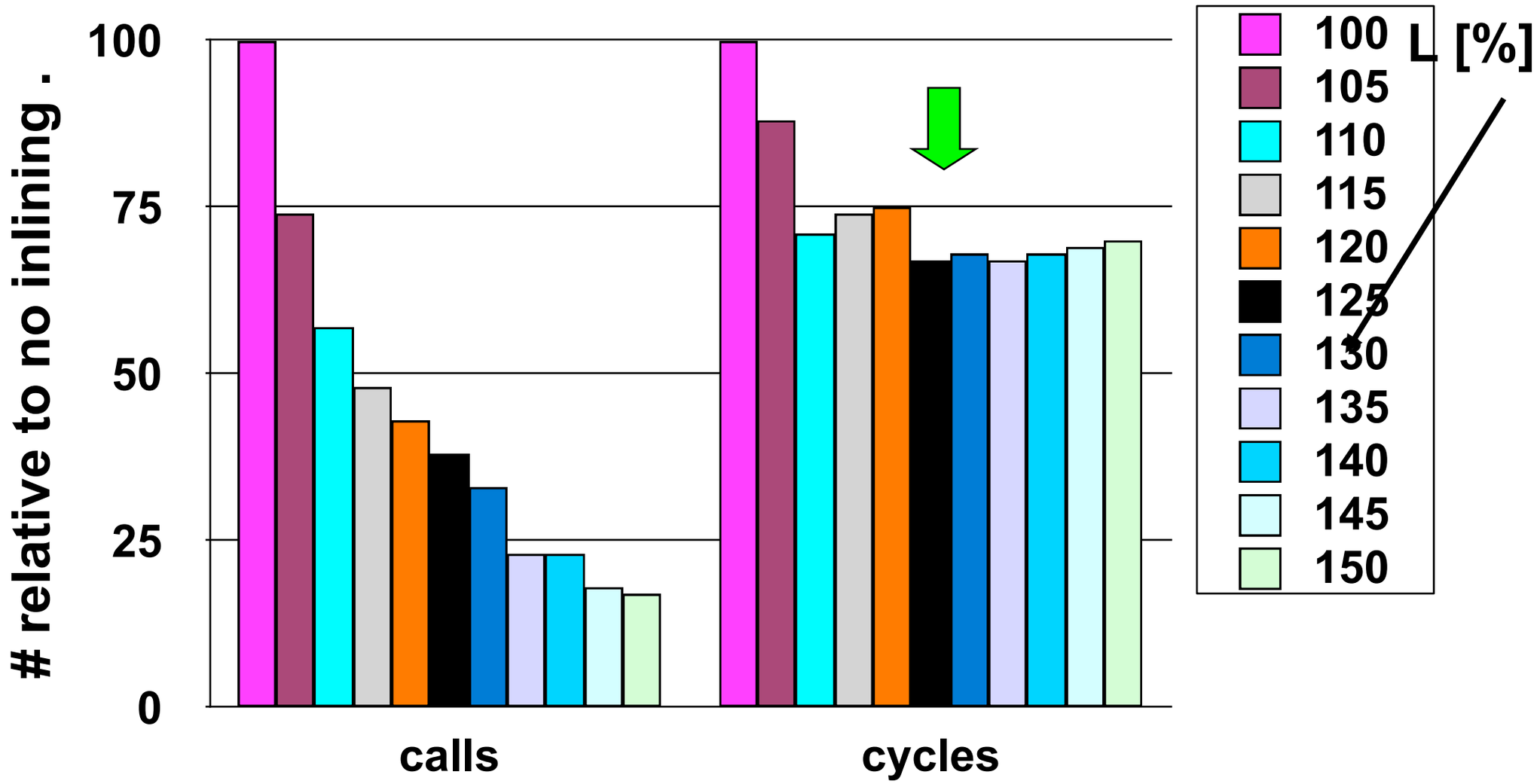
Limitations:

- Not all functions are candidates.
- Code size explosion.
- Requires manual identification using ‘inline’ qualifier.

Goal:

- Controlled code size
- Automatic identification of suitable functions.

Results for GSM speech and channel encoder: #calls, #cycles (TI ‘C62xx)



33% speedup for 25% increase in code size.
# of cycles not a monotonically decreasing function of the code size!

Summary

- Address generating units
    - Single offset assignment problem
- Instruction level parallelism
    - Going beyond list scheduling
- Vectorisation for multimedia instructions
- Avoiding branch delay
    - Using predicated instructions
- Function inlining
    - Trade off between code size and performance