Super-Node SLP: Optimized Vectorization for Code Sequences Containing Operators and Their Inverse Elements

Vasileios Porpodas[1], Rodrigo C. O. Rocha[2]
Evgueni Brevnov[1], Luís F. W. Góes[3]
and Timothy Mattson[1]

Intel[1], University of Edinburgh[2], PUC Minas[3]

CGO 2019

Legal Disclaimer & Optimization Notice

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI'00]

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI'00]
- GCC and LLVM implementations are based on Bottom-Up SLP [Rosen et al. GCC-DEV'07]

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI'00]
- GCC and LLVM implementations are based on Bottom-Up SLP [Rosen et al. GCC-DEV'07]
- SLP and loop-vectorizer complement each other:

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI'00]
- GCC and LLVM implementations are based on Bottom-Up SLP [Rosen et al. GCC-DEV'07]
- SLP and loop-vectorizer complement each other:
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI'00]
- GCC and LLVM implementations are based on Bottom-Up SLP [Rosen et al. GCC-DEV'07]
- SLP and loop-vectorizer complement each other:
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed
- Run SLP after the Loop Vectorizer

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
```

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

LV

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
  A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
```

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

LV

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
  A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
  A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

LV

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
  A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
  A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
  A[i+2,i+6,i+10,i+14] = B[i+2,i+6,i+10,i+14]
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

LV

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

```
for (i=0; i<N; i+=16)
  A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
  A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
  A[i+2,i+6,i+10,i+14] = B[i+2,i+6,i+10,i+14]
  A[i+3,i+7,i+11,i+15] = B[i+3,i+7,i+11,i+15]
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

LV

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
   A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
   A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
   A[i+2,i+6,i+10,i+14] = B[i+2,i+6,i+10,i+14]
   A[i+3,i+7,i+11,i+15] = B[i+3,i+7,i+11,i+15]
```

SLP

SLP Vectorizer with VF = 4

```
for (i=0; i<N; i+=4)
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

LV

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
   A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
   A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
   A[i+2,i+6,i+10,i+14] = B[i+2,i+6,i+10,i+14]
   A[i+3,i+7,i+11,i+15] = B[i+3,i+7,i+11,i+15]
```

SLP

SLP Vectorizer with VF = 4

```
for (i=0; i<N; i+=4)
```

SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
for (i=0; i<N; i+=4)
  A[i] = B[i]
  A[i+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
```

LV

Loop Vectorization (LV) with VF = 4

```
for (i=0; i<N; i+=16)
   A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
   A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
   A[i+2,i+6,i+10,i+14] = B[i+2,i+6,i+10,i+14]
   A[i+3,i+7,i+11,i+15] = B[i+3,i+7,i+11,i+15]
```

SLP

SLP Vectorizer with VF = 4

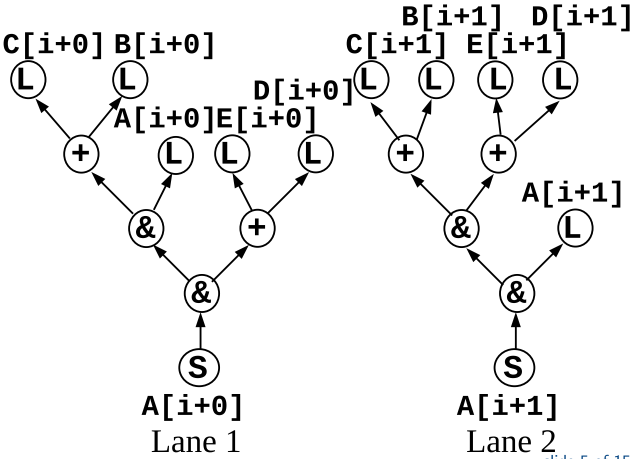
```
for (i=0; i<N; i+=4)
   A[i:i+3] = B[i:i+3]
```

State-of-the-art [Look-Ahead SLP CGO'18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

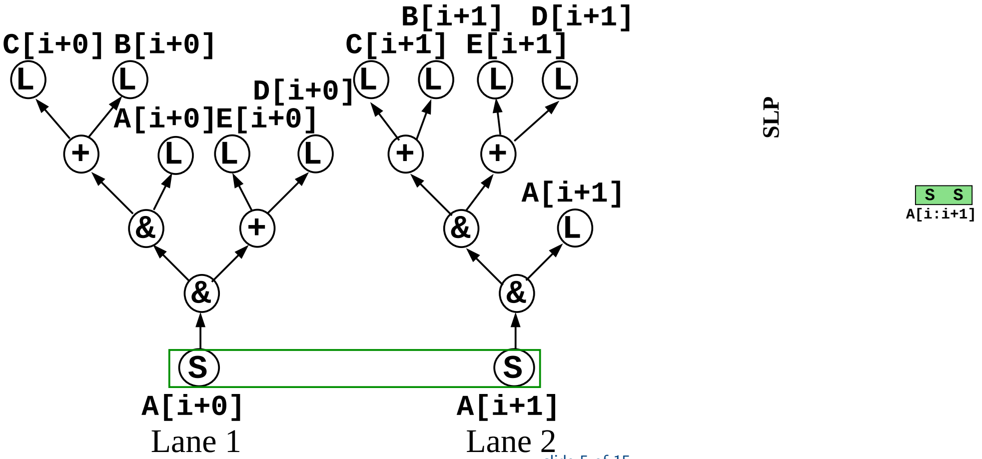
State-of-the-art [Look-Ahead SLP CGO'18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic



State-of-the-art [Look-Ahead SLP CGO'18]

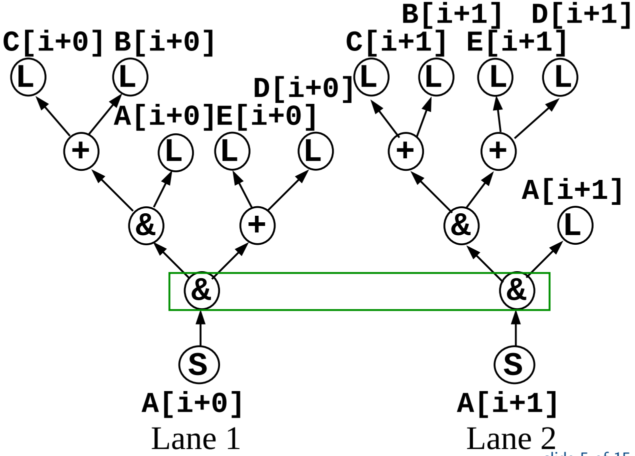
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

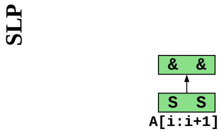


 http://vporpo.me

State-of-the-art [Look-Ahead SLP CGO'18]

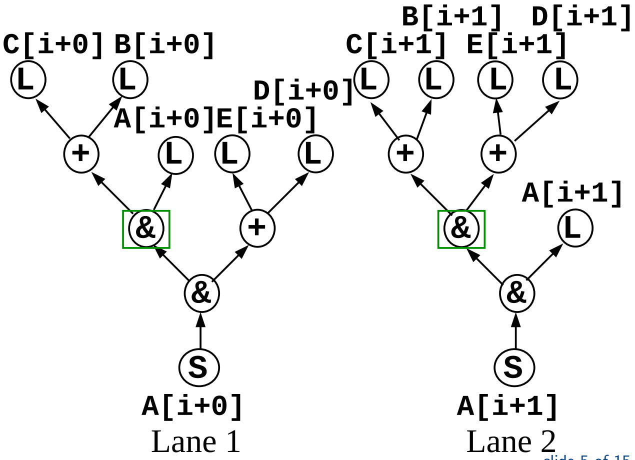
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

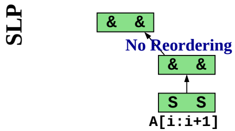




State-of-the-art [Look-Ahead SLP CGO'18]

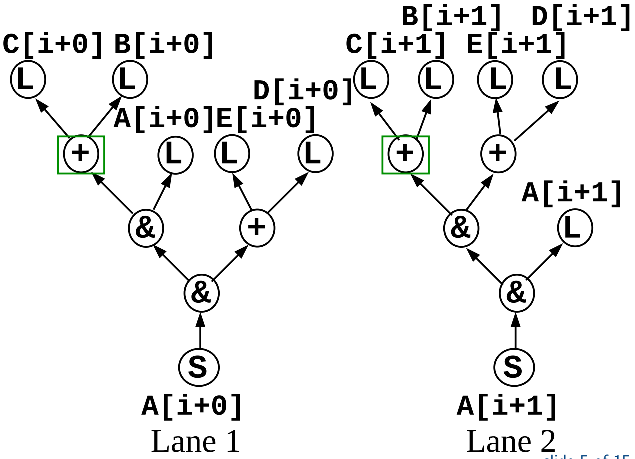
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

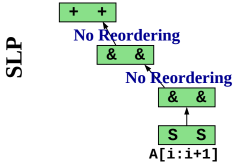




State-of-the-art [Look-Ahead SLP CGO'18]

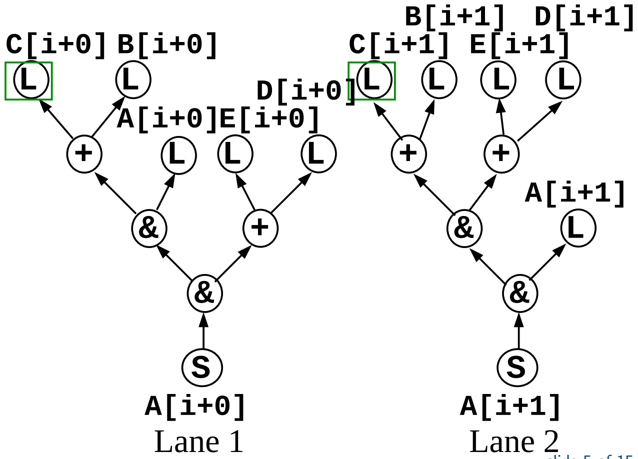
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

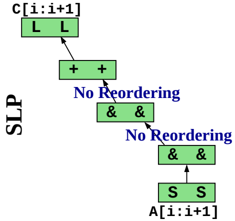




State-of-the-art [Look-Ahead SLP CGO'18]

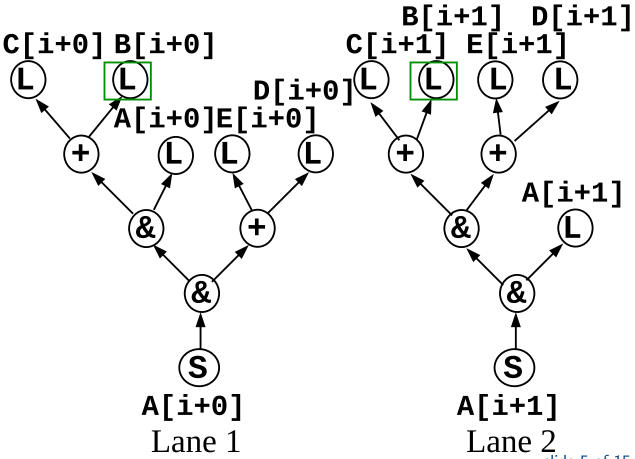
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

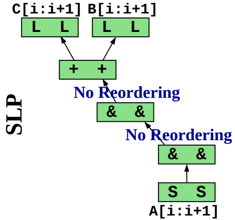




State-of-the-art [Look-Ahead SLP CGO'18]

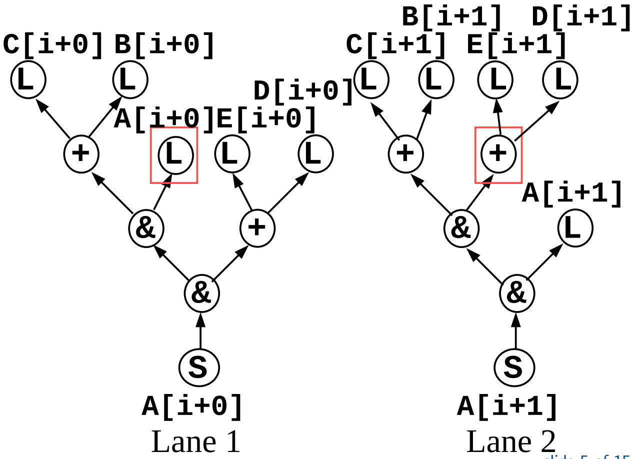
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

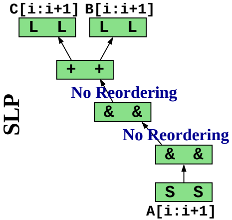




State-of-the-art [Look-Ahead SLP CGO'18]

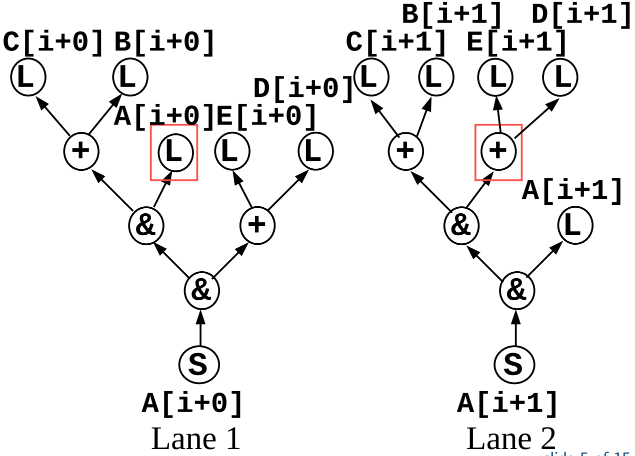
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

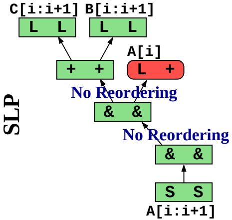




State-of-the-art [Look-Ahead SLP CGO'18]

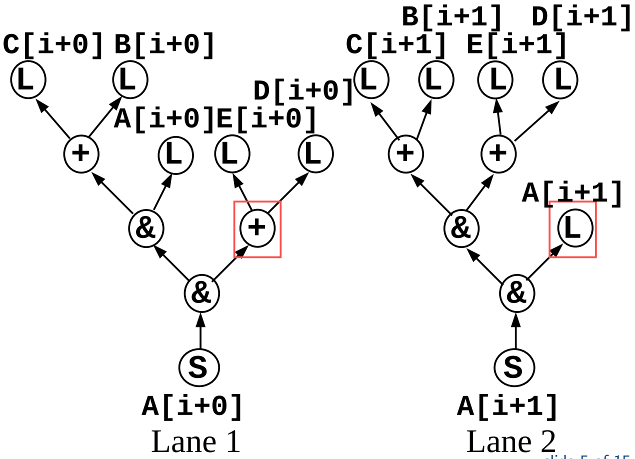
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

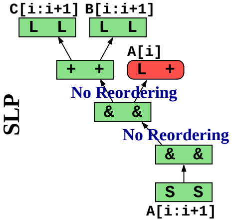




State-of-the-art [Look-Ahead SLP CGO'18]

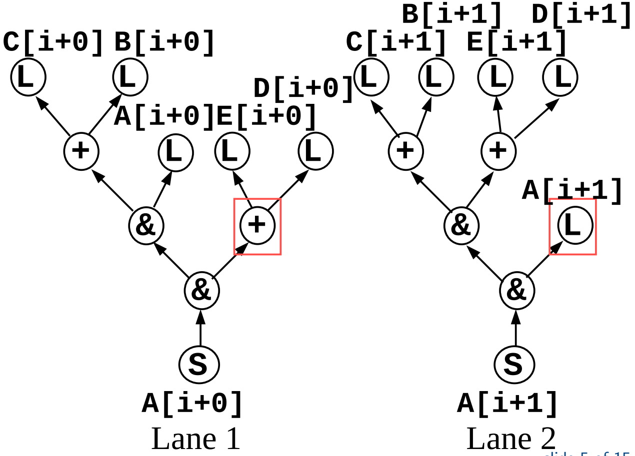
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

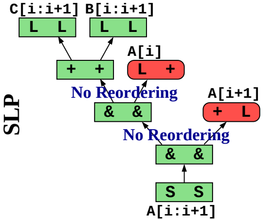




State-of-the-art [Look-Ahead SLP CGO'18]

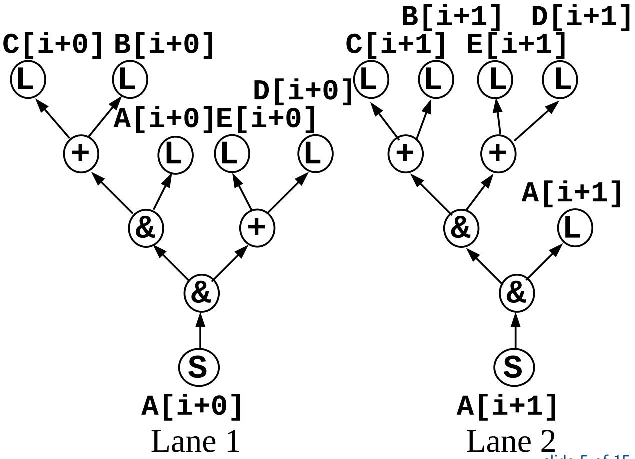
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

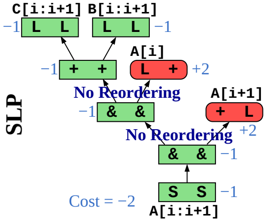




State-of-the-art [Look-Ahead SLP CGO'18]

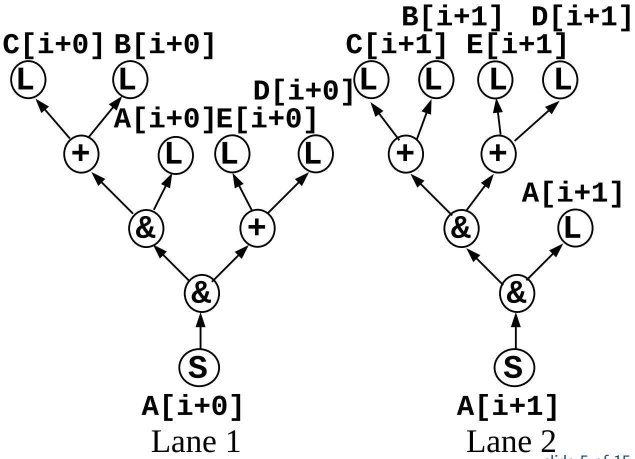
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

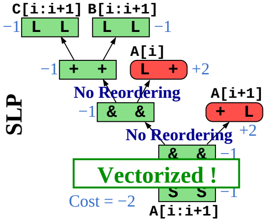




State-of-the-art [Look-Ahead SLP CGO'18]

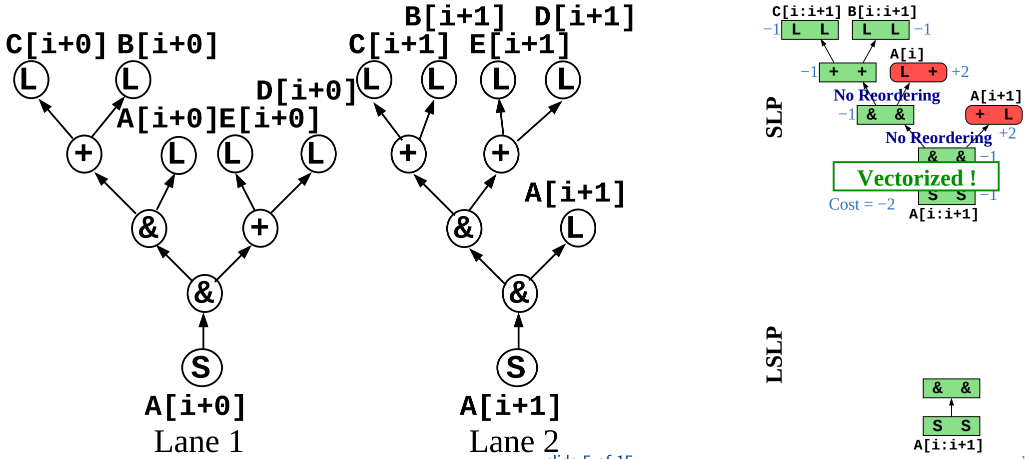
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic





State-of-the-art [Look-Ahead SLP CGO'18]

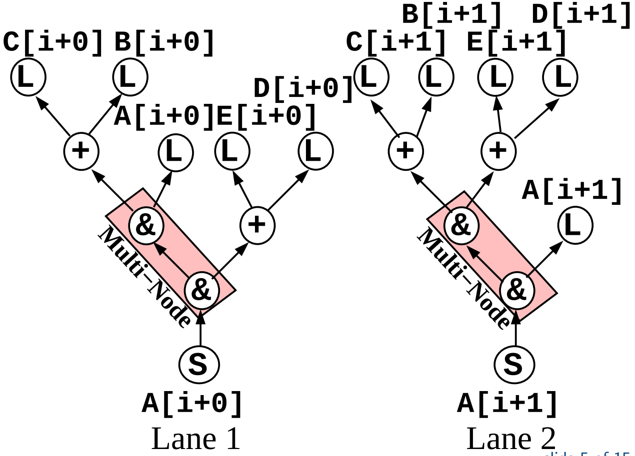
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

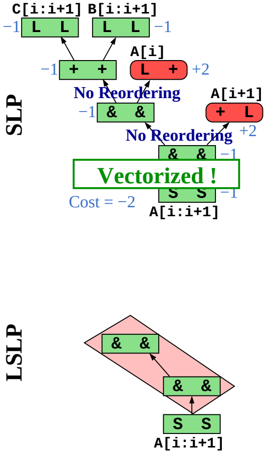


http://vporpo.me

State-of-the-art [Look-Ahead SLP CGO'18]

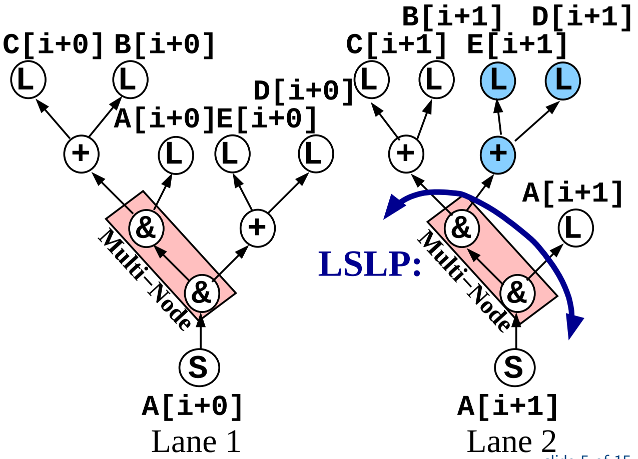
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

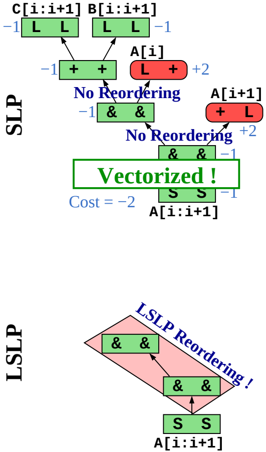




State-of-the-art [Look-Ahead SLP CGO'18]

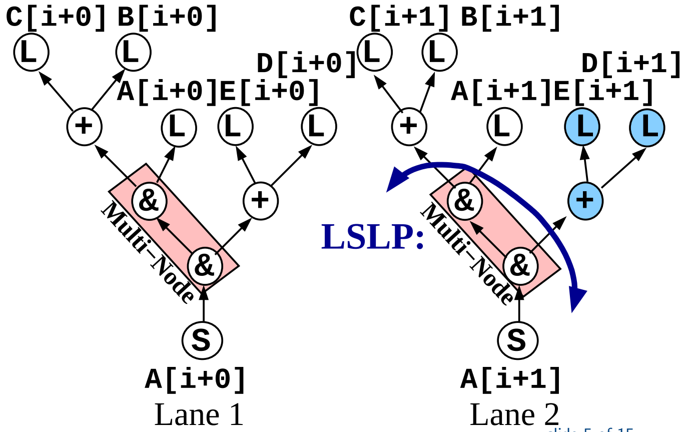
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

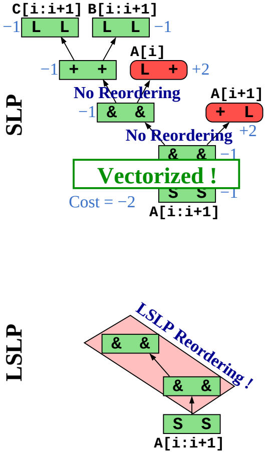




State-of-the-art [Look-Ahead SLP CGO'18]

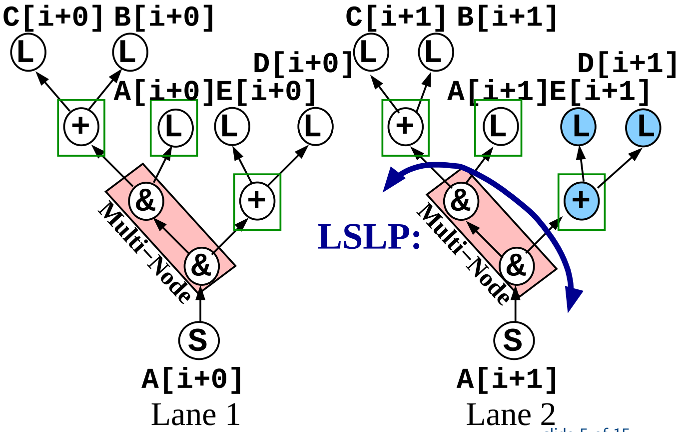
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

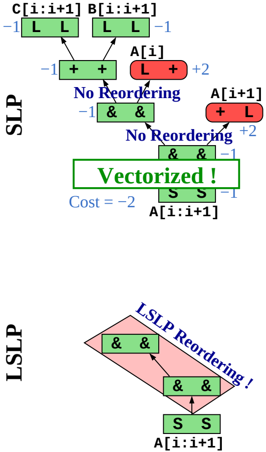




State-of-the-art [Look-Ahead SLP CGO'18]

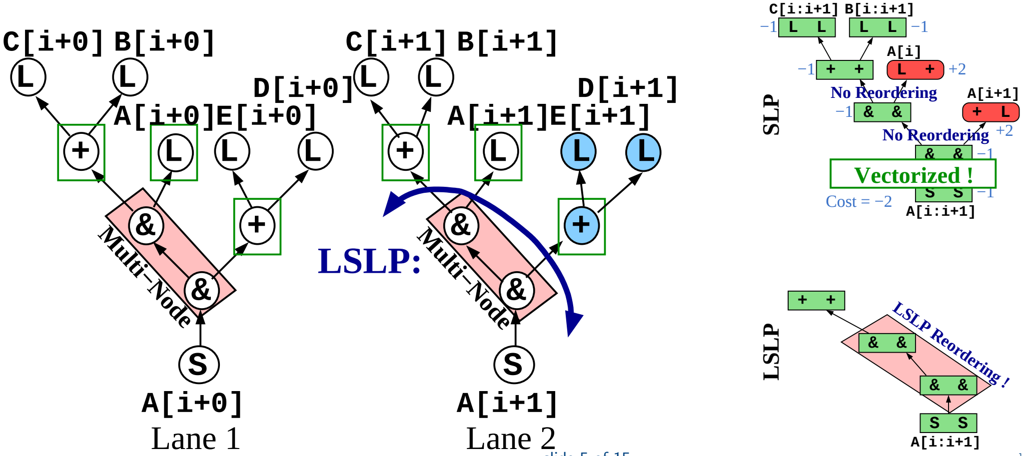
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic





State-of-the-art [Look-Ahead SLP CGO'18]

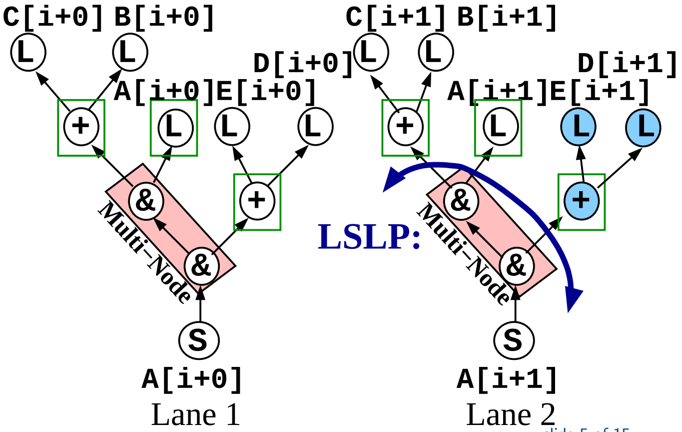
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

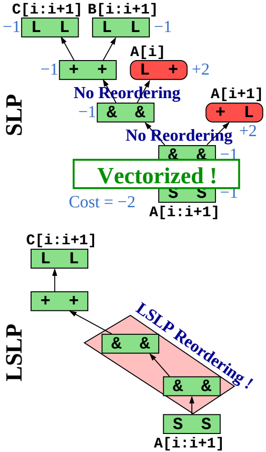


http://vporpo.me

State-of-the-art [Look-Ahead SLP CGO'18]

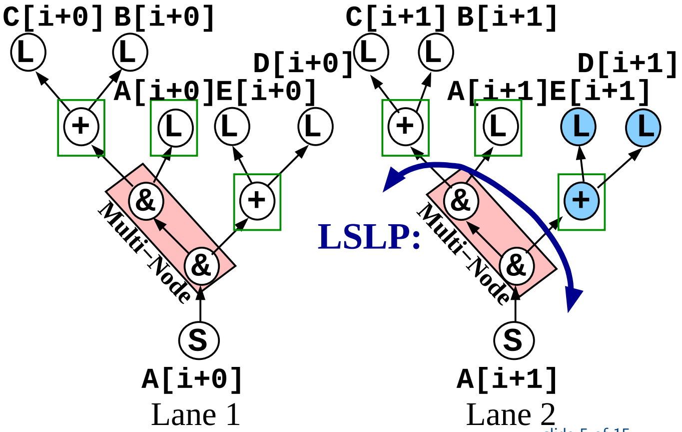
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

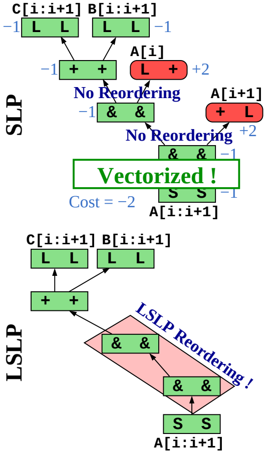




State-of-the-art [Look-Ahead SLP CGO'18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

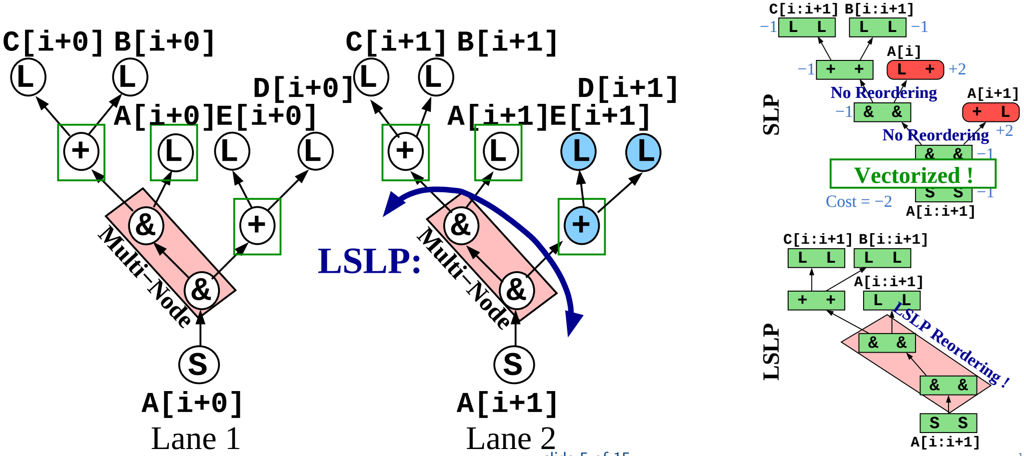




 http://vporpo.me

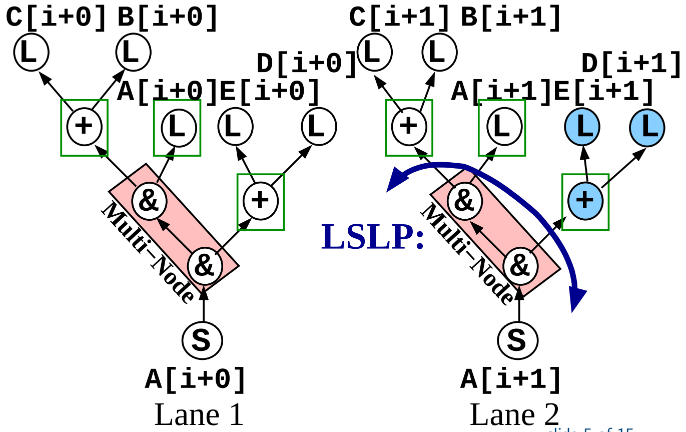
State-of-the-art [Look-Ahead SLP CGO'18]

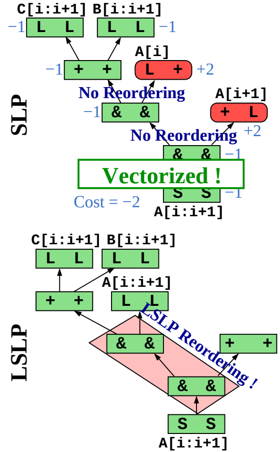
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic



State-of-the-art [Look-Ahead SLP CGO'18]

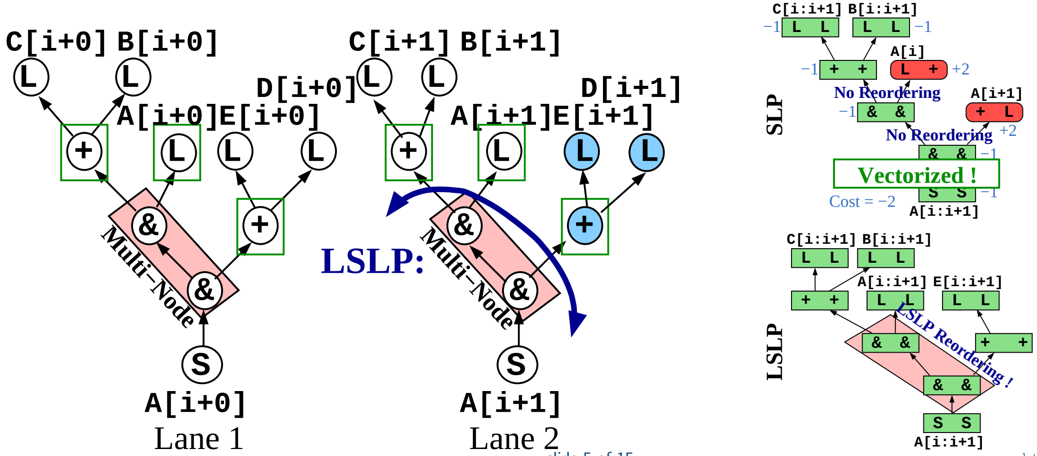
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic





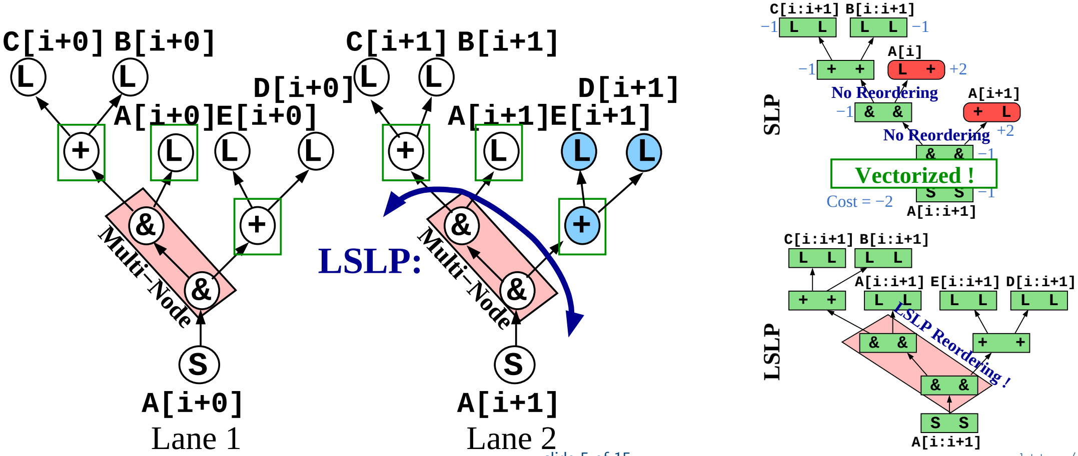
State-of-the-art [Look-Ahead SLP CGO'18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic



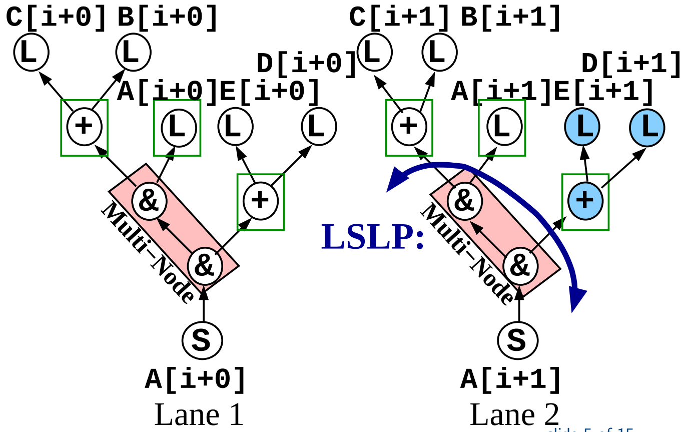
State-of-the-art [Look-Ahead SLP CGO'18]

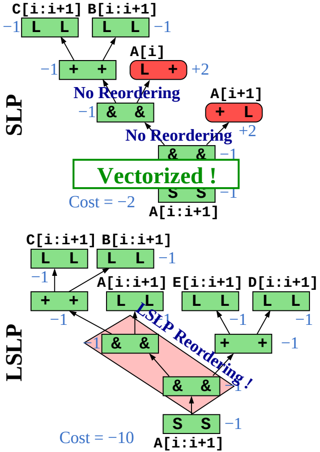
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic



State-of-the-art [Look-Ahead SLP CGO'18]

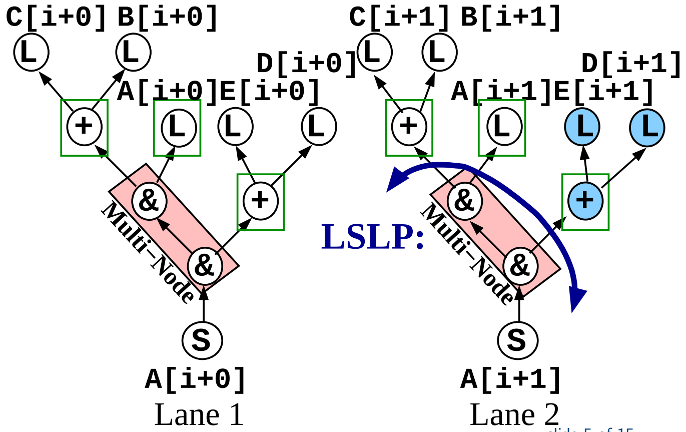
- Form Multi-Nodes and reorder operands with Look-Ahead heuristic

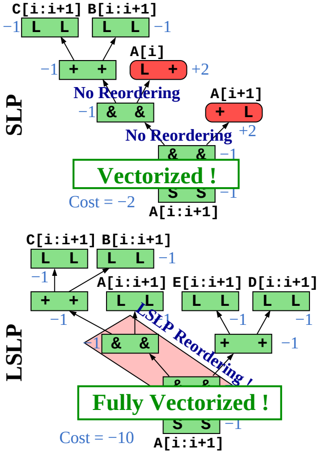




State-of-the-art [Look-Ahead SLP CGO'18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic





Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

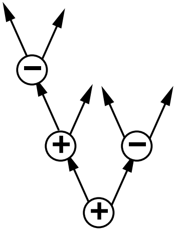
- The inverse element of ADD(+) is SUB(-)

Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements

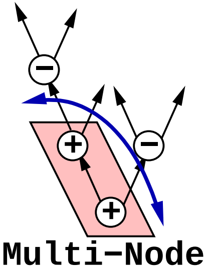
Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements



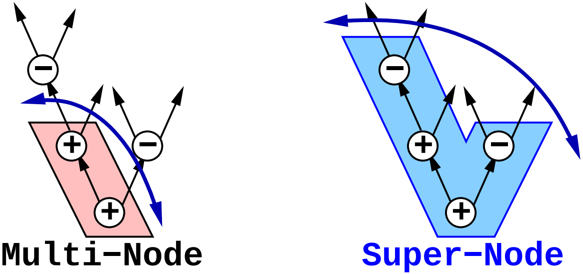
Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements



Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements
- Super-Nodes **can** reorder across them when legal

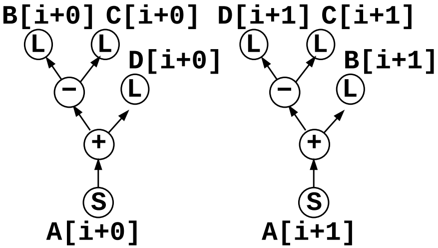


LSLP fails in the presence of inverse elements

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

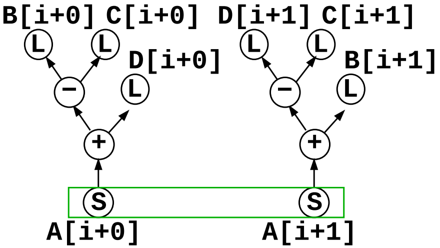
LSLP fails in the presence of inverse elements

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```



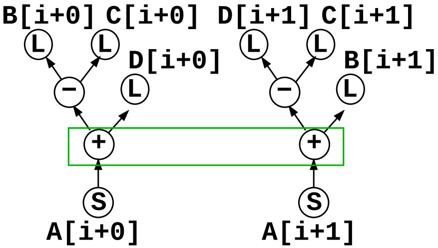
LSLP fails in the presence of inverse elements

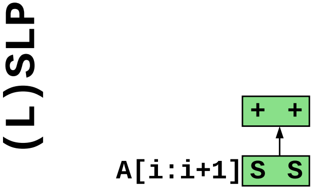
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```



LSLP fails in the presence of inverse elements

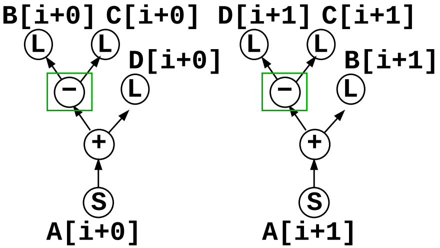
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

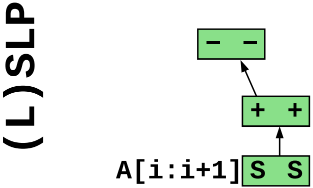




LSLP fails in the presence of inverse elements

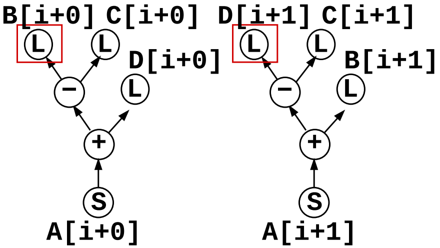
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

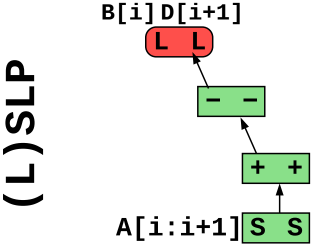




LSLP fails in the presence of inverse elements

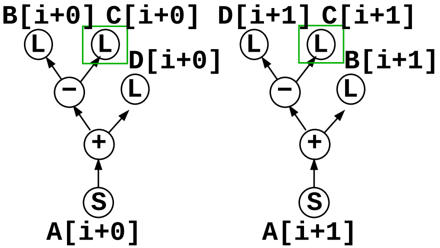
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

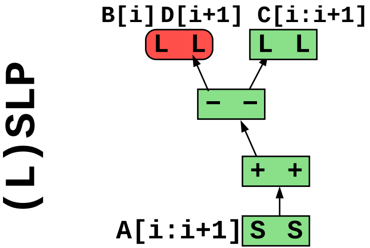




LSLP fails in the presence of inverse elements

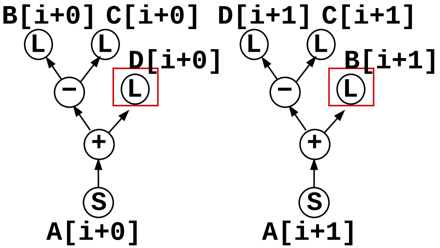
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

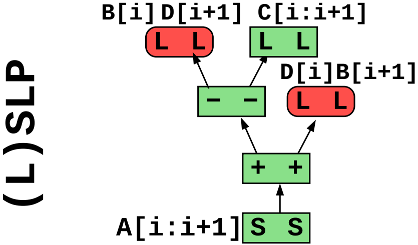




LSLP fails in the presence of inverse elements

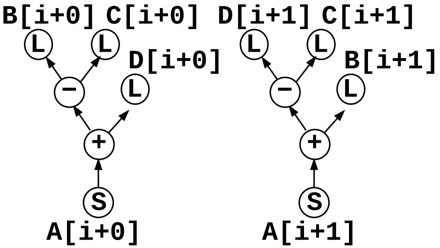
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

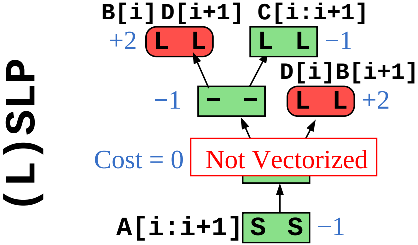




LSLP fails in the presence of inverse elements

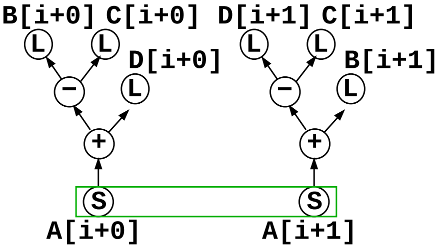
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

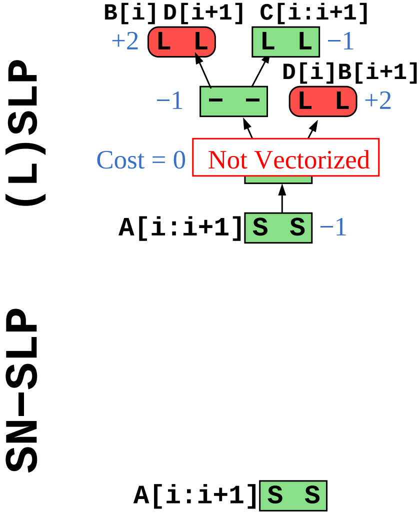




LSLP fails in the presence of inverse elements

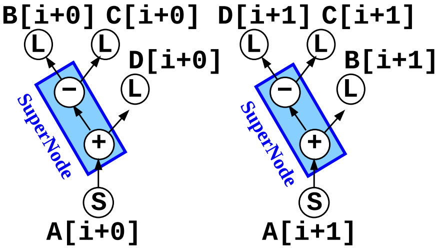
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]−C[i+0]+D[i+0];
A[i+1]=D[i+1]−C[i+1]+B[i+1];
```

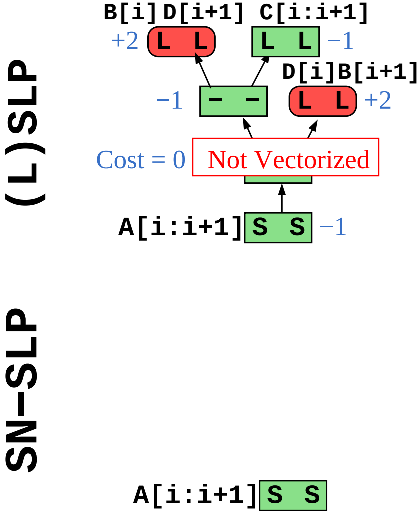




LSLP fails in the presence of inverse elements

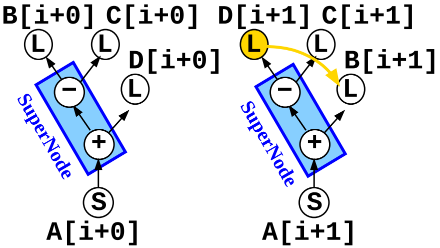
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]−C[i+0]+D[i+0];
A[i+1]=D[i+1]−C[i+1]+B[i+1];
```

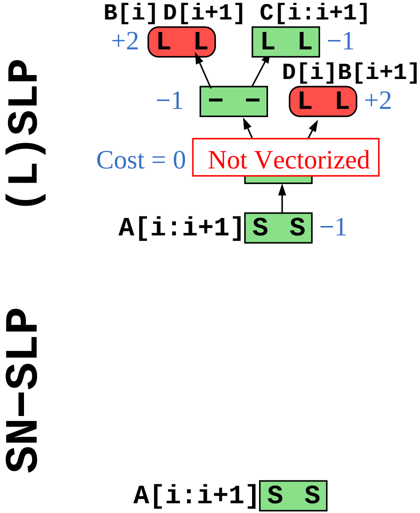




LSLP fails in the presence of inverse elements

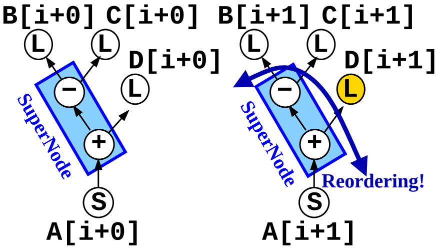
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

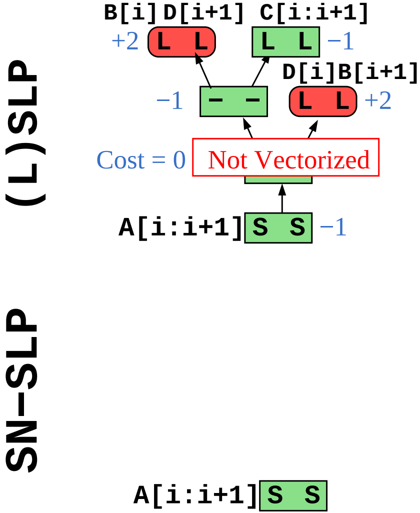




LSLP fails in the presence of inverse elements

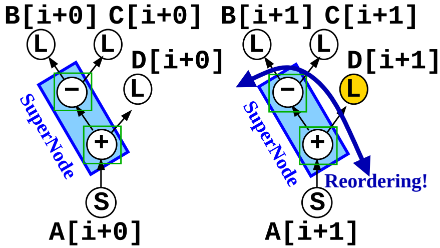
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]−C[i+0]+D[i+0];
A[i+1]=D[i+1]−C[i+1]+B[i+1];
```

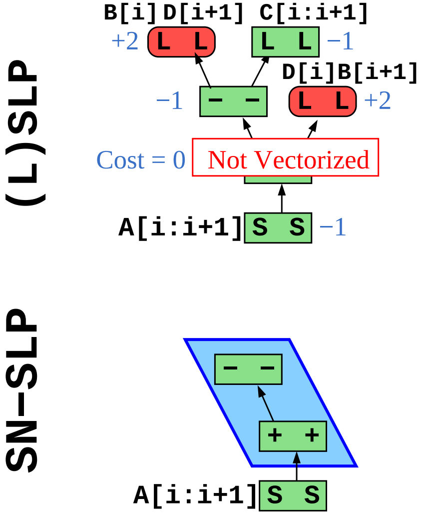




LSLP fails in the presence of inverse elements

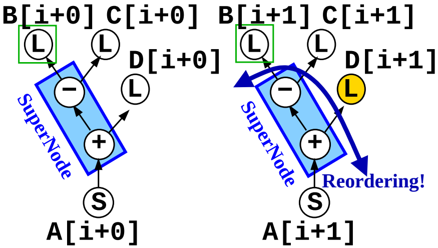
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]−C[i+0]+D[i+0];
A[i+1]=D[i+1]−C[i+1]+B[i+1];
```

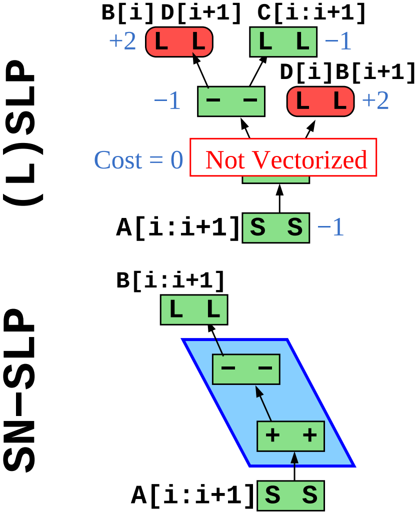




LSLP fails in the presence of inverse elements

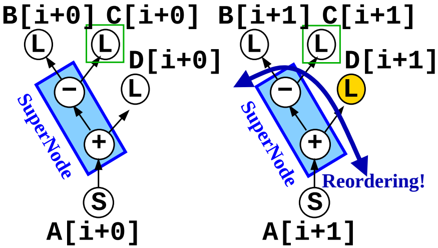
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

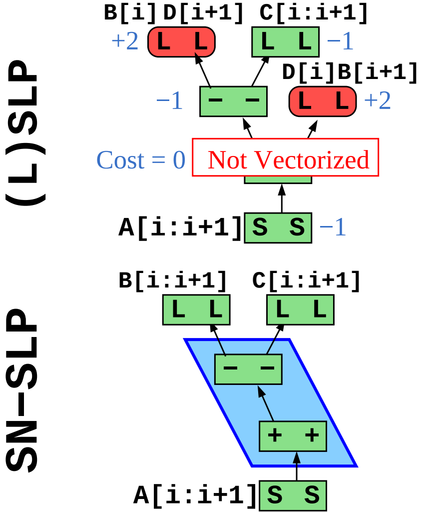




LSLP fails in the presence of inverse elements

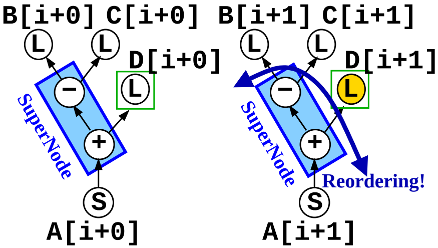
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

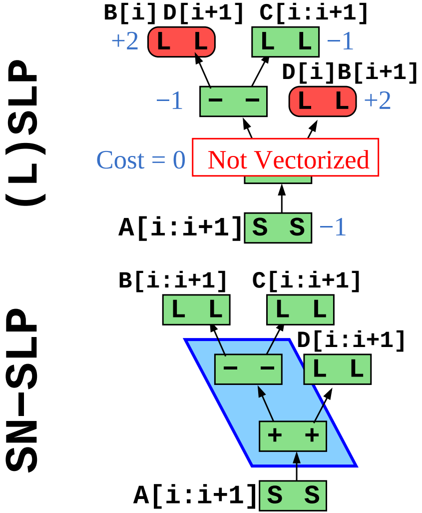




LSLP fails in the presence of inverse elements

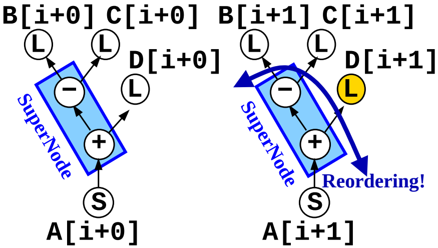
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```

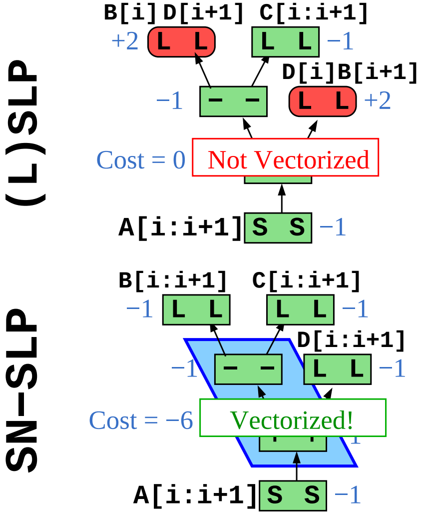




LSLP fails in the presence of inverse elements

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=D[i+1]-C[i+1]+B[i+1];
```



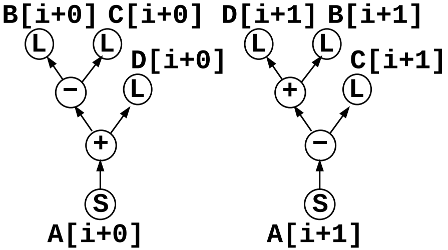


SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

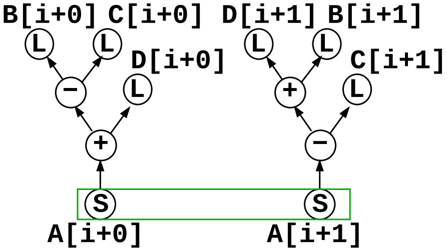
SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

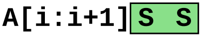


SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

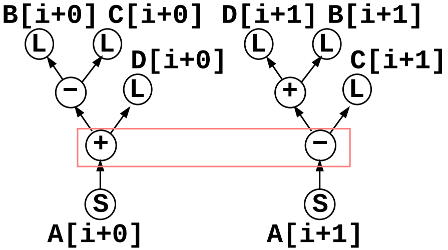


(L)SLP

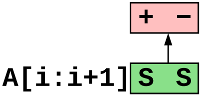


SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

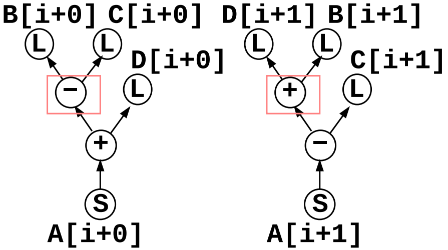


(L)SLP

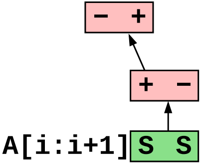


SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

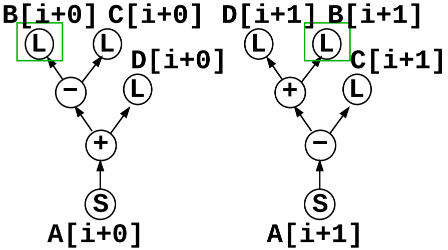


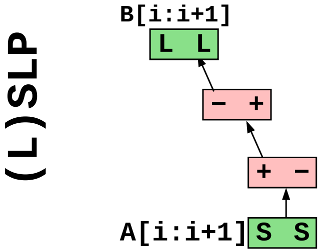
(L)SLP



SuperNode internal nodes can be reorderd too

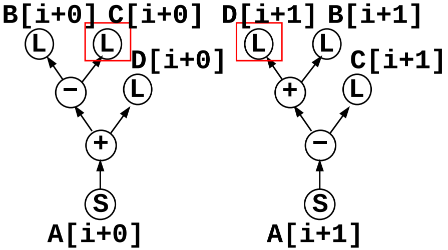
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

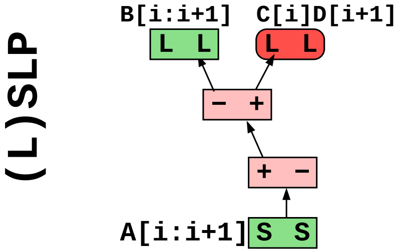




SuperNode internal nodes can be reorderd too

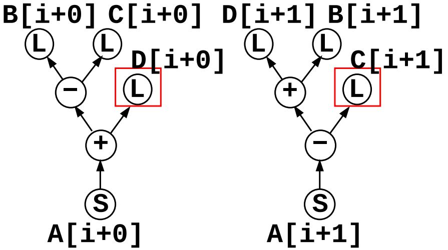
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

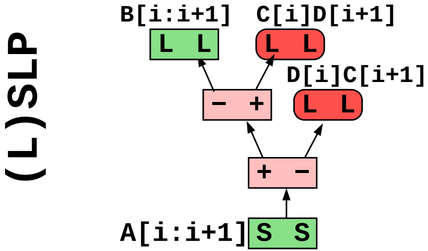




SuperNode internal nodes can be reorderd too

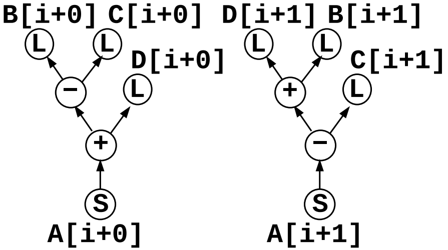
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

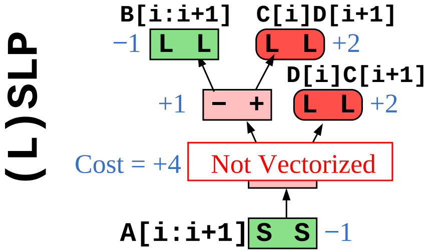




SuperNode internal nodes can be reorderd too

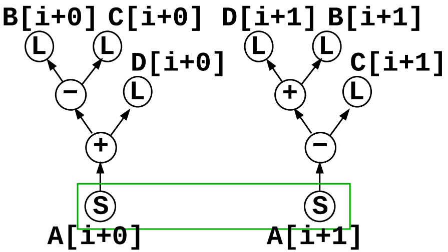
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

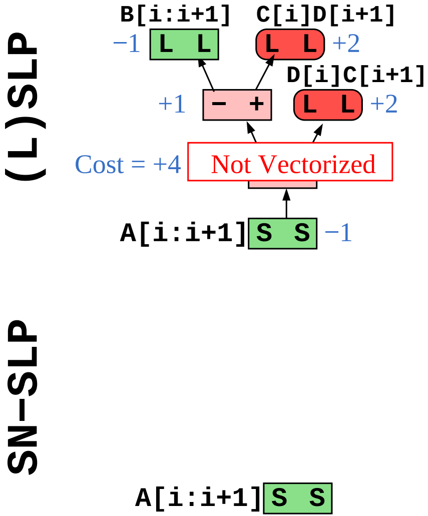




SuperNode internal nodes can be reorderd too

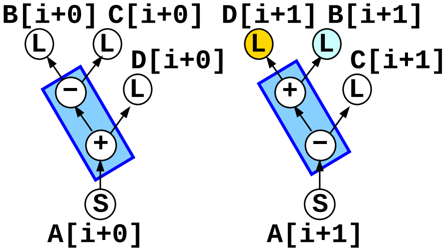
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

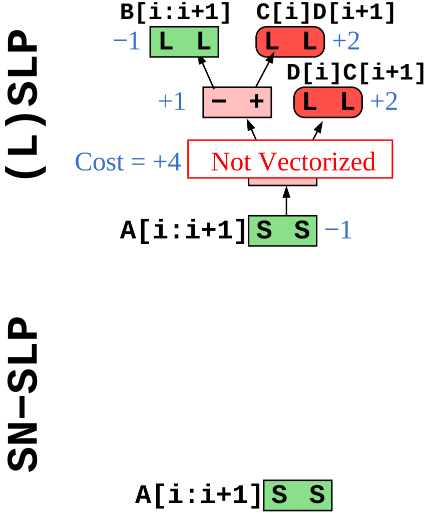




SuperNode internal nodes can be reorderd too

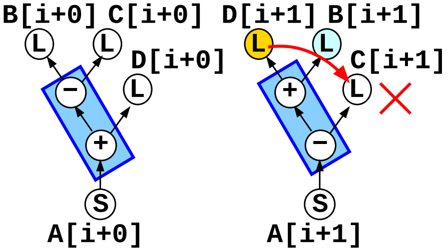
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

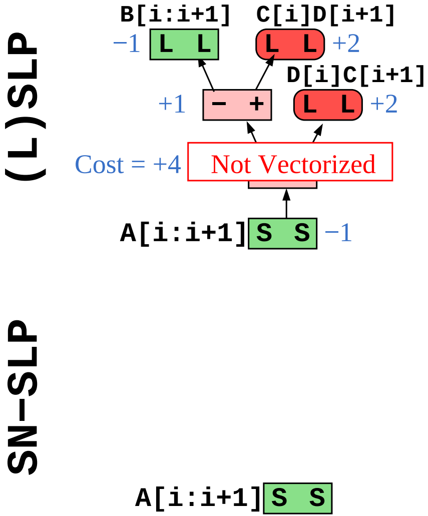




SuperNode internal nodes can be reorderd too

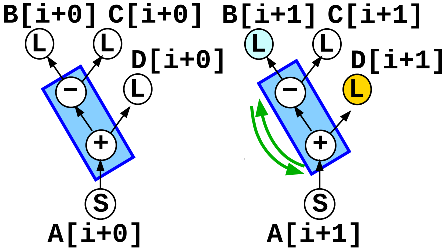
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

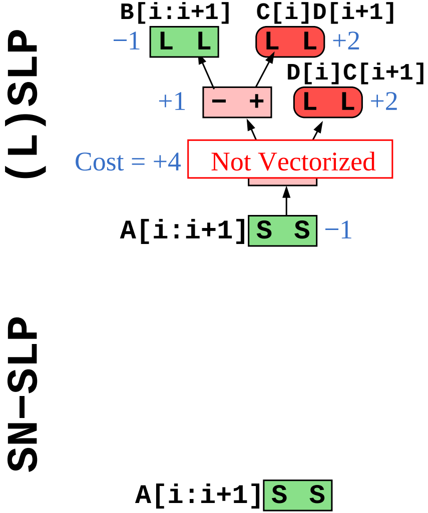




SuperNode internal nodes can be reorderd too

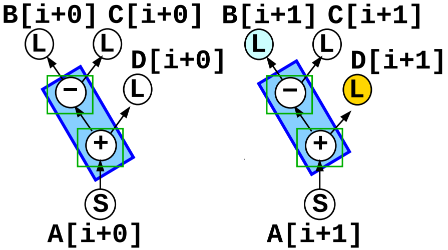
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

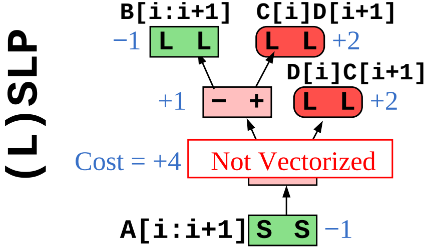


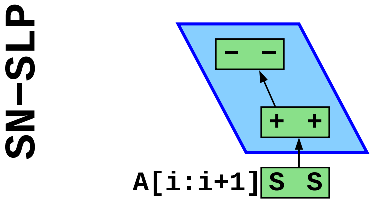


SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

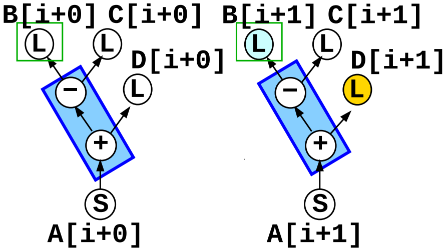


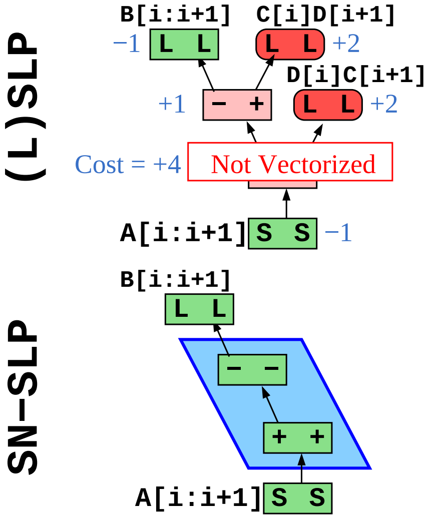




SuperNode internal nodes can be reorderd too

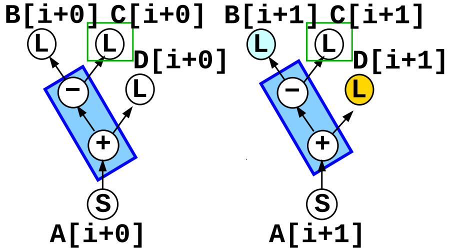
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

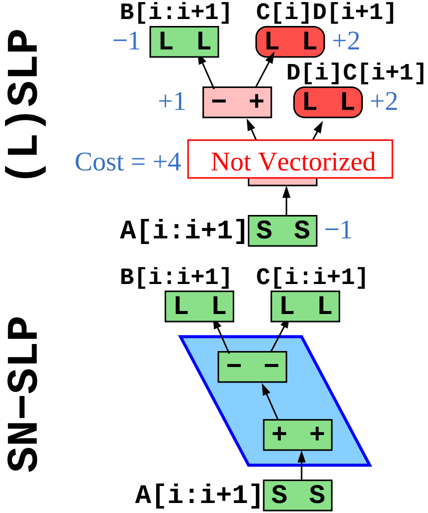




SuperNode internal nodes can be reorderd too

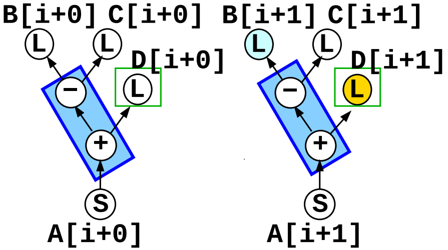
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

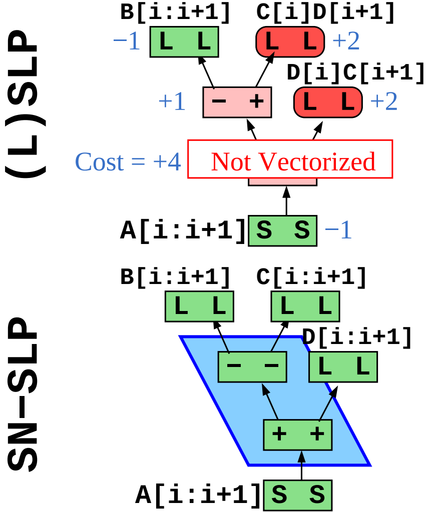




SuperNode internal nodes can be reorderd too

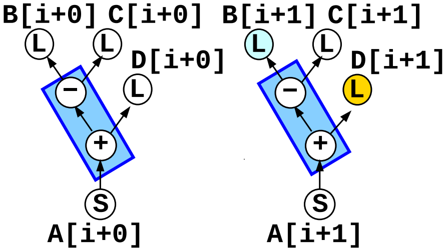
```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```

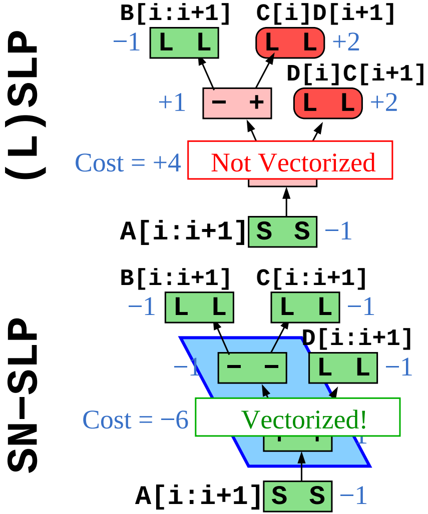




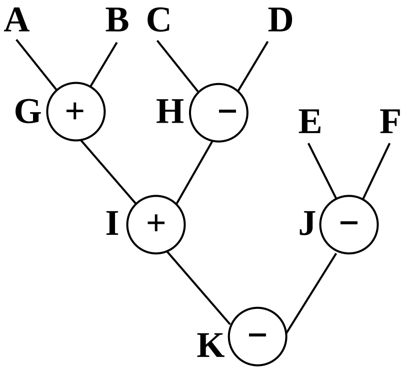
SuperNode internal nodes can be reorderd too

```
long A[],B[],C[],D[];
A[i+0]=B[i+0]-C[i+0]+D[i+0];
A[i+1]=B[i+1]+D[i+1]-C[i+1];
```



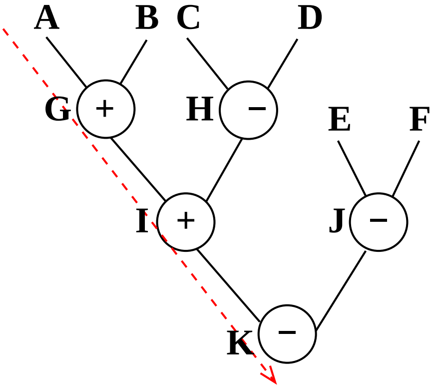


Legality



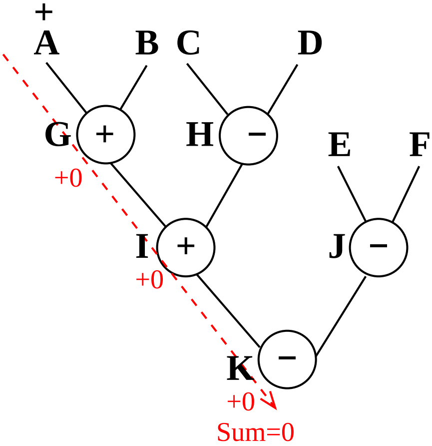
		Accumulated Path Operation (APO)	Linearized

Legality



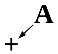
		Accumulated Path Operation (APO)	Linearized

Legality

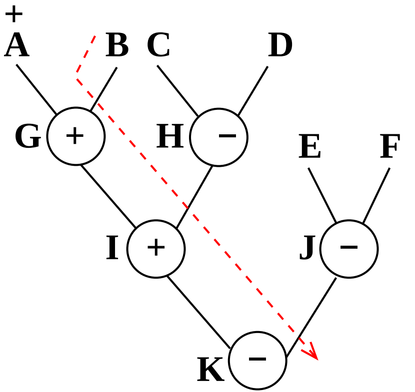


	Sum	Accumulated Path Operation (APO)
A	0	+

Linearized

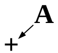


Legality

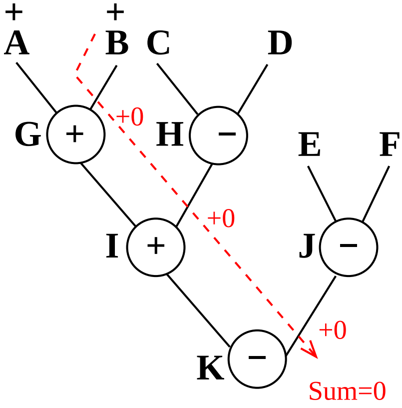


	Sum	Accumulated Path Operation (APO)
A	0	+

Linearized

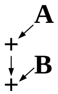


Legality

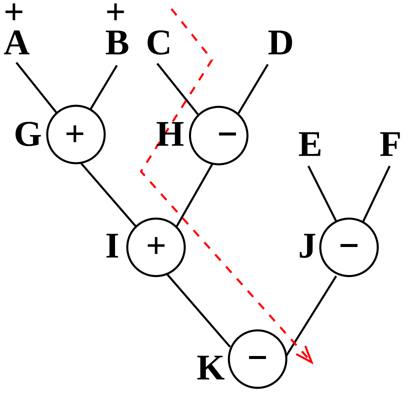


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+

Linearized

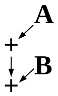


Legality

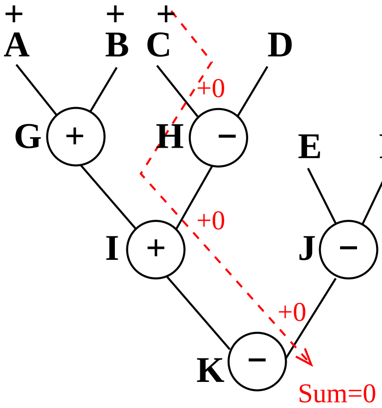


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+

Linearized

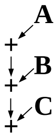


Legality

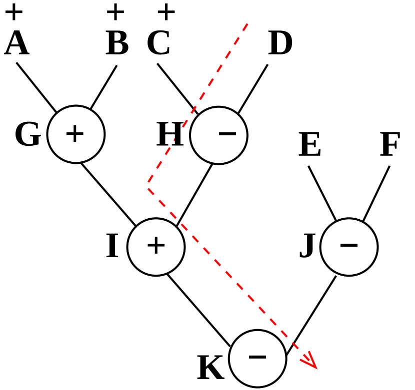


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+

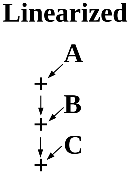
Linearized



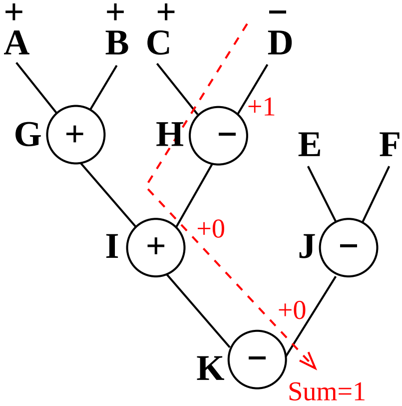
Legality



	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+



Legality

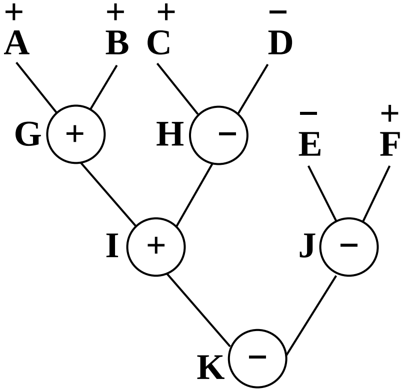


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−

Linearized

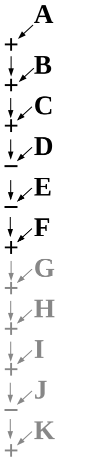
A → + → B → + → C → + → D → −

Legality

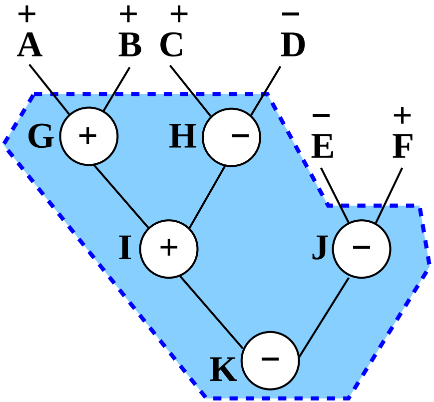


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

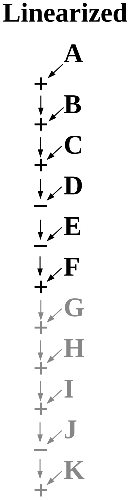
Linearized



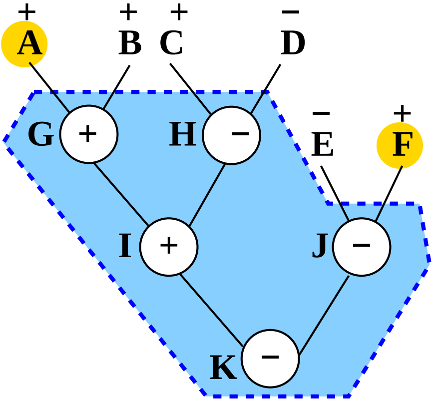
Legality



	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

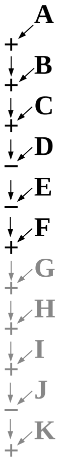


Legality

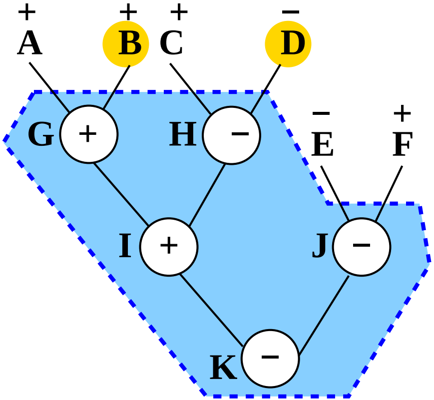


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

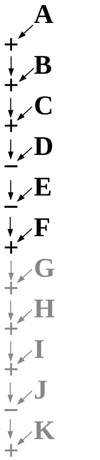
Linearized



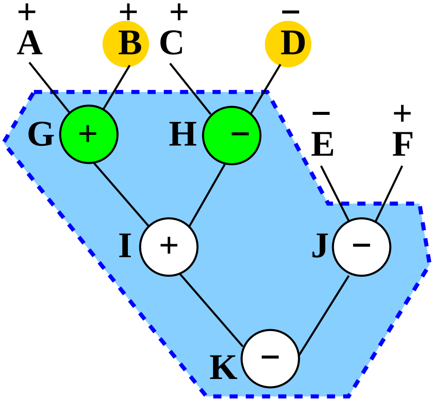
Legality



	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

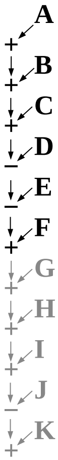


Legality

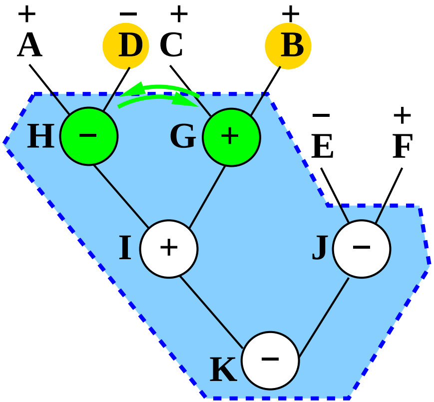


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

Linearized

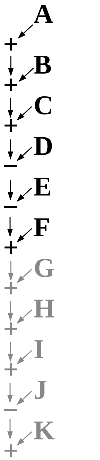


Legality

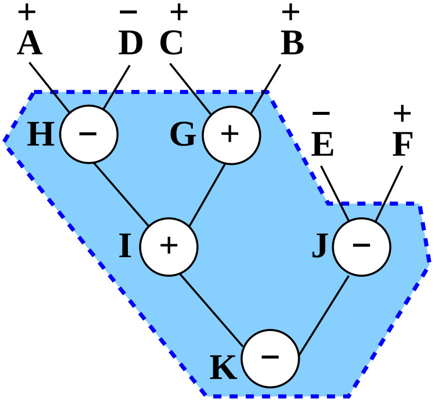


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

Linearized

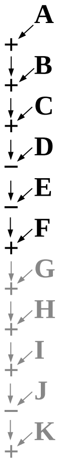


Legality

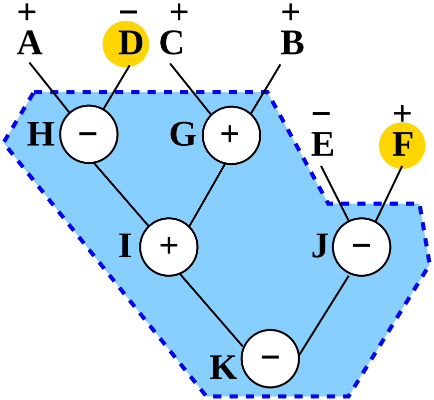


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

Linearized

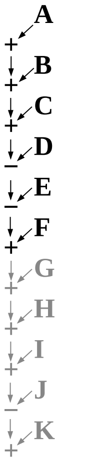


Legality

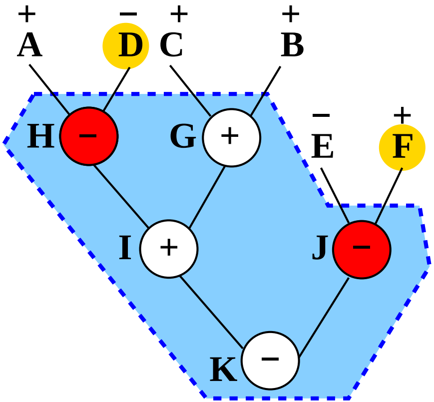


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H	0	+
I	0	+
J	1	−
K	0	+

Linearized

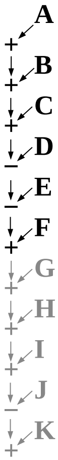


Legality

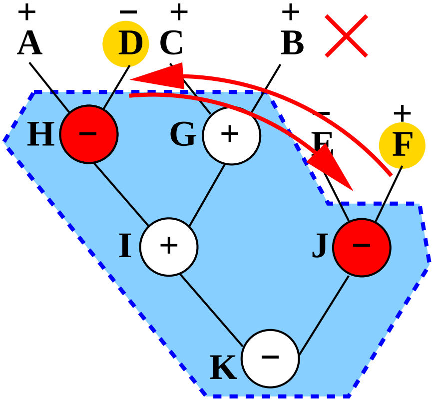


	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H		+
I	0	+
J	0	−
K	0	+

Linearized

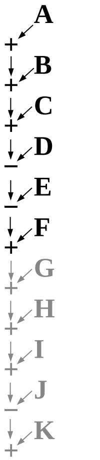


Legality



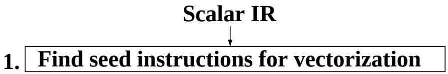
	Sum	Accumulated Path Operation (APO)
A	0	+
B	0	+
C	0	+
D	1	−
E	1	−
F	2	+
G	0	+
H		+
I	0	+
J	0	−
K	0	+

Linearized



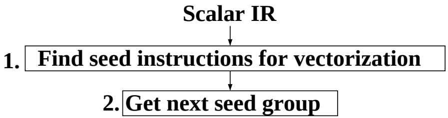
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions



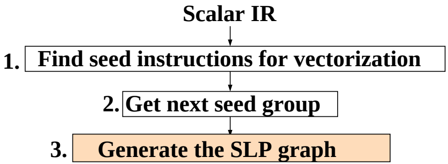
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions



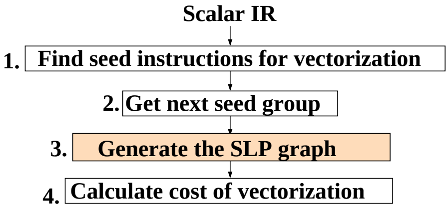
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions



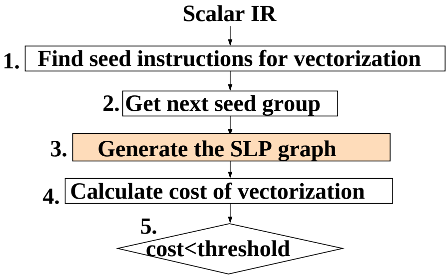
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)



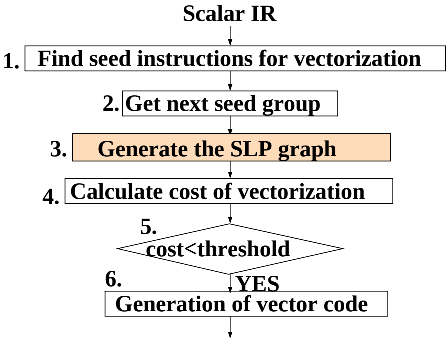
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability



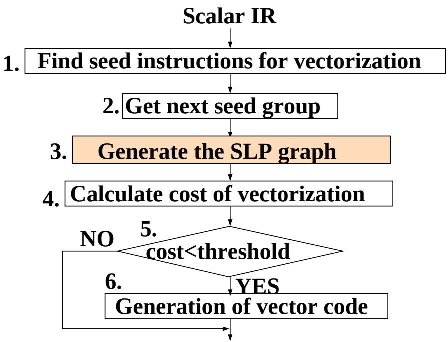
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability
- Generate vector code



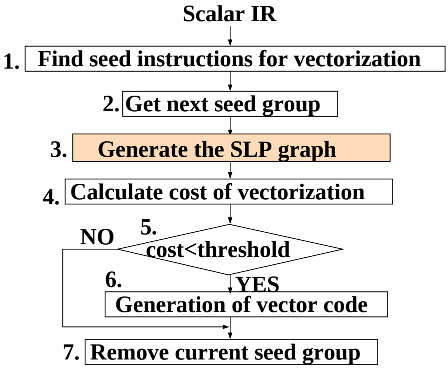
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability
- Generate vector code



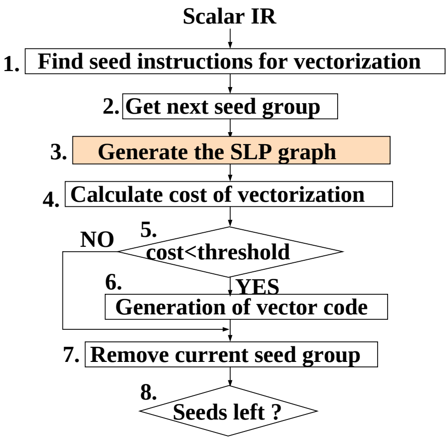
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability
- Generate vector code



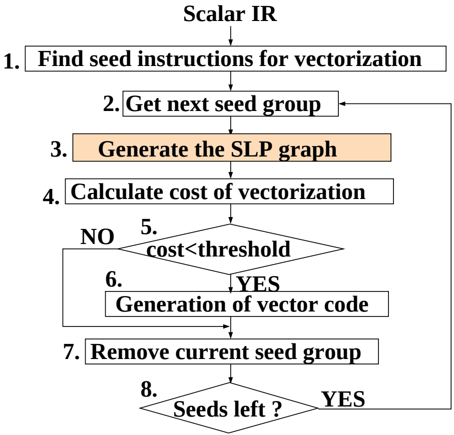
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability
- Generate vector code



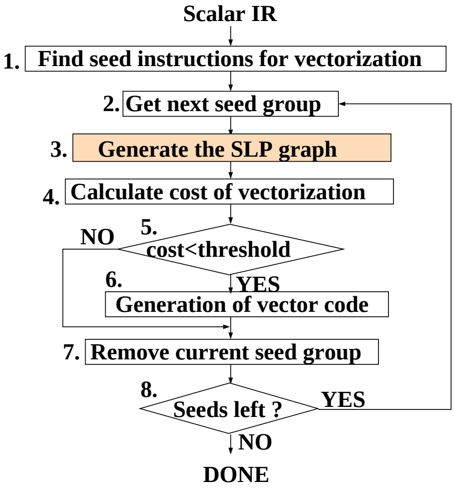
SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability
- Generate vector code
- Repeat



SN-SLP Algorithm

- Seed instructions are usually:
  1. Consecutive Stores
  2. Reductions
- Graph contains groups of vectorizable instructions
- Cost: weighted instr. count (TTI)
- Check overall profitability
- Generate vector code
- Repeat



Experimental Setup

- Implemented in LLVM trunk

Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core^{TM} i5-6440HQ CPU

Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core™ i5-6440HQ CPU
- Compiler flags: -O3 -ffast-math -march=native -mtune=native

Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core™ i5-6440HQ CPU
- Compiler flags: -O3 -ffast-math -march=native -mtune=native
- Kernels from unmodified functions of SPEC CPU2006

Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core™ i5-6440HQ CPU
- Compiler flags: -O3 -ffast-math -march=native -mtune=native
- Kernels from unmodified functions of SPEC CPU2006
- We evaluated the following:

Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core^{TM} i5-6440HQ CPU
- Compiler flags: -O3 -ffast-math -march=native -mtune=native
- Kernels from unmodified functions of SPEC CPU2006
- We evaluated the following:
  1. O3 : All vectorizers disabled

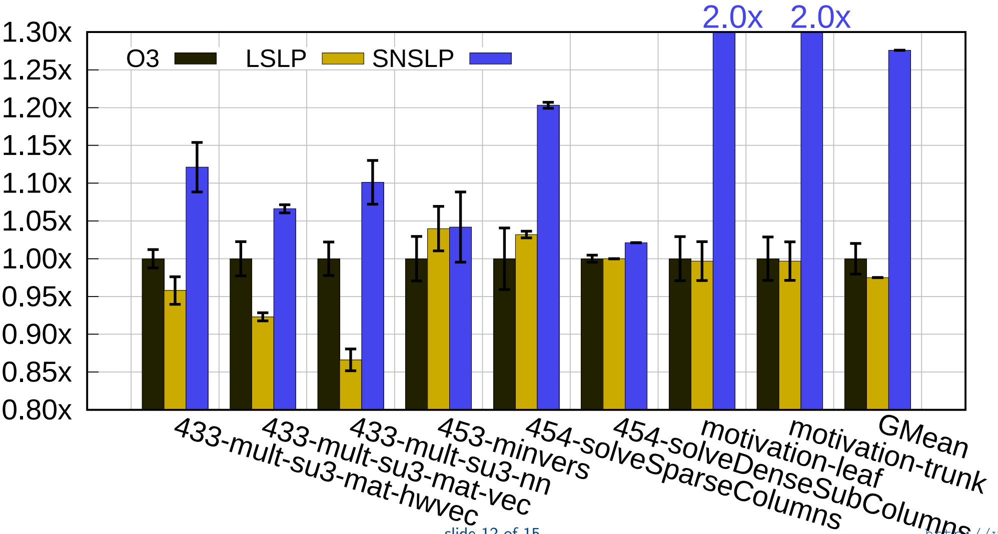
Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core^{TM} i5-6440HQ CPU
- Compiler flags: -O3 -ffast-math -march=native -mtune=native
- Kernels from unmodified functions of SPEC CPU2006
- We evaluated the following:
  1. O3 : All vectorizers disabled
  2. LSLP : O3 + LSLP

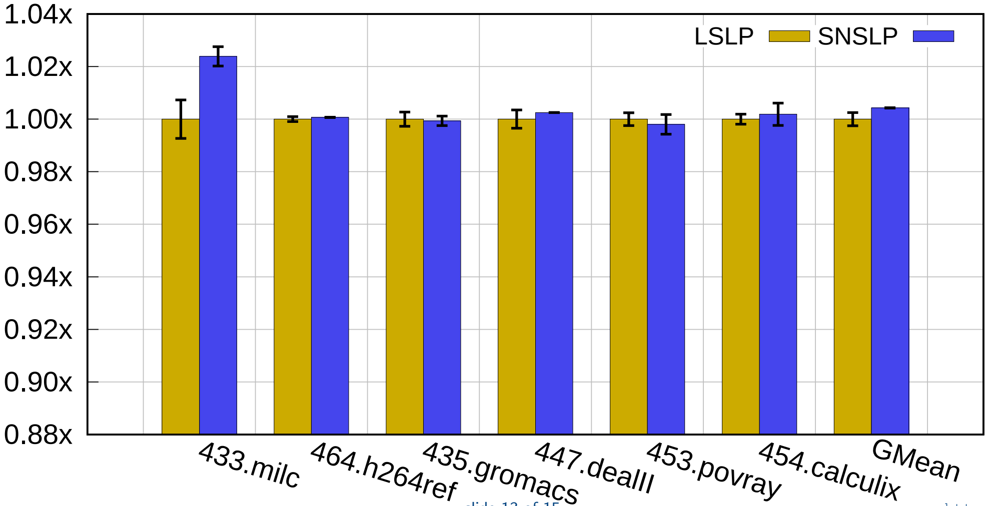
Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core™ i5-6440HQ CPU
- Compiler flags: -O3 -ffast-math -march=native -mtune=native
- Kernels from unmodified functions of SPEC CPU2006
- We evaluated the following:
  1. O3 : All vectorizers disabled
  2. LSLP : O3 + LSLP
  3. SNSLP : O3 + SN-SLP

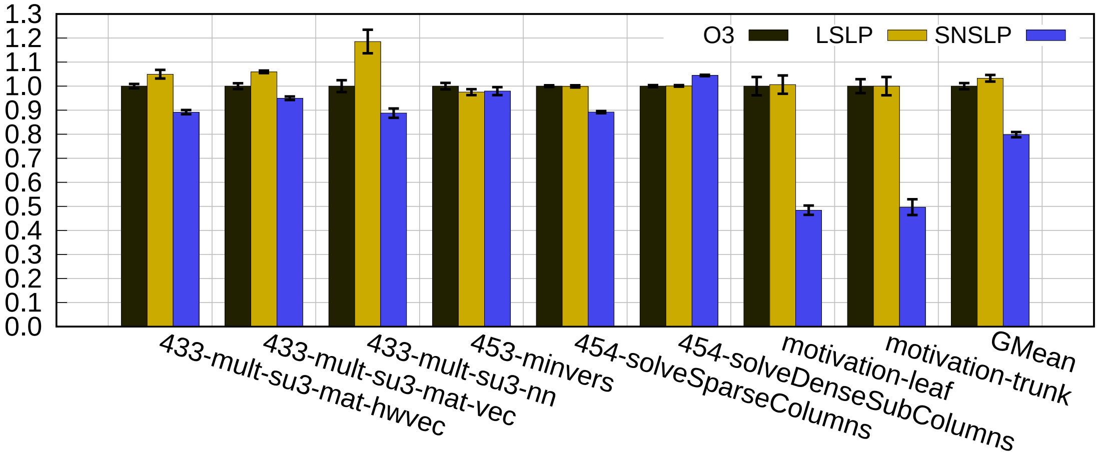
Performance of kernels



Performance (Full Benchmarks)



Total Compilation Time



Conclusion

- SN-SLP improves the effectiveness of SLP on code with inverse elements.

Conclusion

- SN-SLP improves the effectiveness of SLP on code with inverse elements.
  1. It forms Super-Nodes of commutative operations and their inverse elements
  2. It performs legal operand reordering, guided by the Look-Ahead heuristic

Conclusion

- SN-SLP improves the effectiveness of SLP on code with inverse elements.
  1. It forms Super-Nodes of commutative operations and their inverse elements
  2. It performs legal operand reordering, guided by the Look-Ahead heuristic
- Better at identifying isomorphism

Conclusion

- SN-SLP improves the effectiveness of SLP on code with inverse elements.
  1. It forms Super-Nodes of commutative operations and their inverse elements
  2. It performs legal operand reordering, guided by the Look-Ahead heuristic
- Better at identifying isomorphism
- Implemented in LLVM as an extension of SLP

Conclusion

- SN-SLP improves the effectiveness of SLP on code with inverse elements.
  1. It forms Super-Nodes of commutative operations and their inverse elements
  2. It performs legal operand reordering, guided by the Look-Ahead heuristic
- Better at identifying isomorphism
- Implemented in LLVM as an extension of SLP
- Improves performance with similar compilation time