VW-SLP: Auto-Vectorization with Adaptive Vector Width

Vasileios Porpodas [1] Rodrigo C. O. Rocha [2] Luís F. W. Góes [3]

[1]Intel Santa Clara, USA

[2]University of Edinburgh, UK

[3]PUC Minas, Brazil

PACT 2018

Legal Disclaimer & Optimization Notice

Optimization Notice

What is Vector-Width (VW) ?

- Wikipedia: In Computer Science a **vector** is

  “A one-dimensional array”

What is Vector-Width (VW) ?

- Wikipedia: In Computer Science a **vector** is

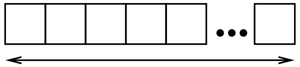
  “A one-dimensional array”

What is Vector-Width (VW) ?

- Wikipedia: In Computer Science a **vector** is

  "A one-dimensional array"

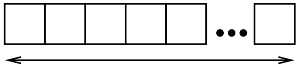
- Vector-Width is the size of the vector



What is Vector-Width (VW) ?

- Wikipedia: In Computer Science a **vector** is

  "A one-dimensional array"



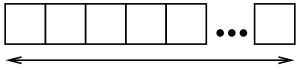
- Vector-Width is the size of the vector
- Aka Vector Length (VL) or Vector Factor (VF)

What is Vector-Width (VW) ?

- Wikipedia: In Computer Science a **vector** is

  "A one-dimensional array"

- Vector-Width is the size of the vector
- Aka Vector Length (VL) or Vector Factor (VF)
- VW == VL == VF



Vector Width
Vector Length
Vector Factor

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- Vectorizes across instructions, *NOT* iterations

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- Vectorizes across instructions, *NOT* iterations

LV

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+4]
```

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- Vectorizes across instructions, *NOT* iterations

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+4]
```

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

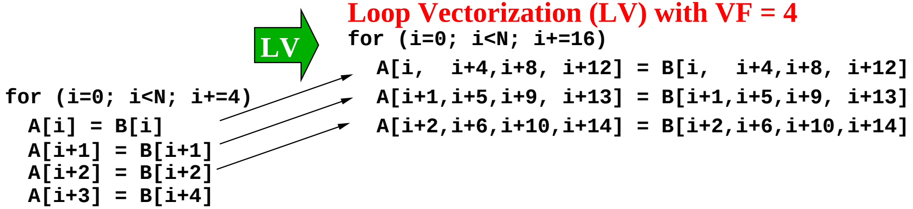
LV

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]
```

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- Vectorizes across instructions, *NOT* iterations



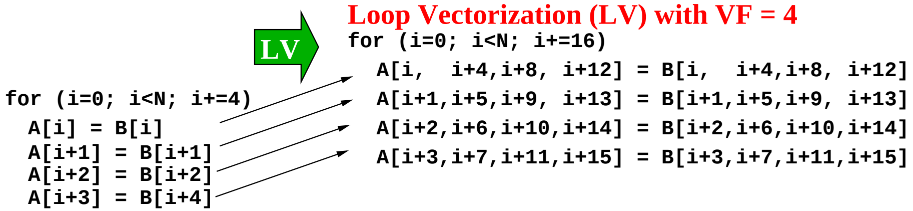
Loop Vectorization (LV) with VF = 4

```
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]
```

```
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+4]
```

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

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: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

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: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

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]
```

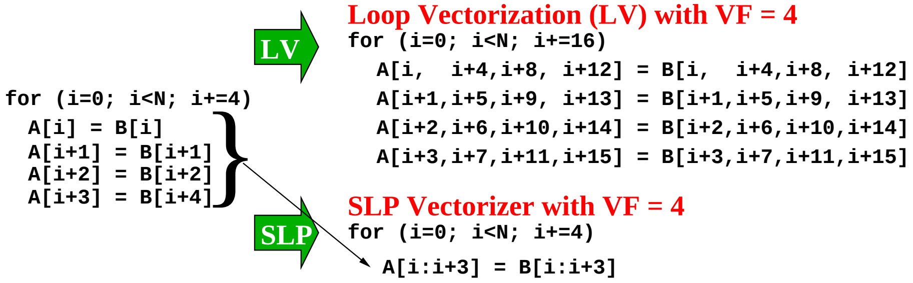
SLP

SLP Vectorizer with VF = 4

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

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

Loop Vectorization (LV) with VF = 4

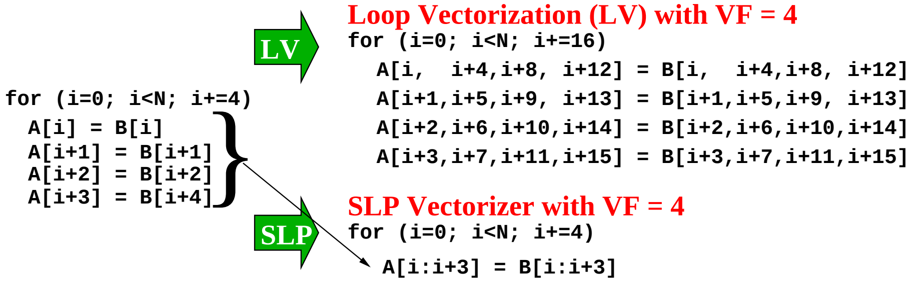
```
for (i=0; i<N; i+=16)
  A[i,  i+4,i+8, i+12] = B[i,  i+4,i+8, i+12]
  A[i+1,i+5,i+9, i+13] = B[i+1,i+5,i+9, i+13]
  A[i+2,i+6,i+10,i+14] = B[i+2,i+6,i+10,i+14]
  A[i+3,i+7,i+11,i+15] = B[i+3,i+7,i+11,i+15]
```

SLP Vectorizer with VF = 4

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

SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism
- Bottom-Up SLP implemented in GCC and LLVM
- 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+4]
```

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]
```

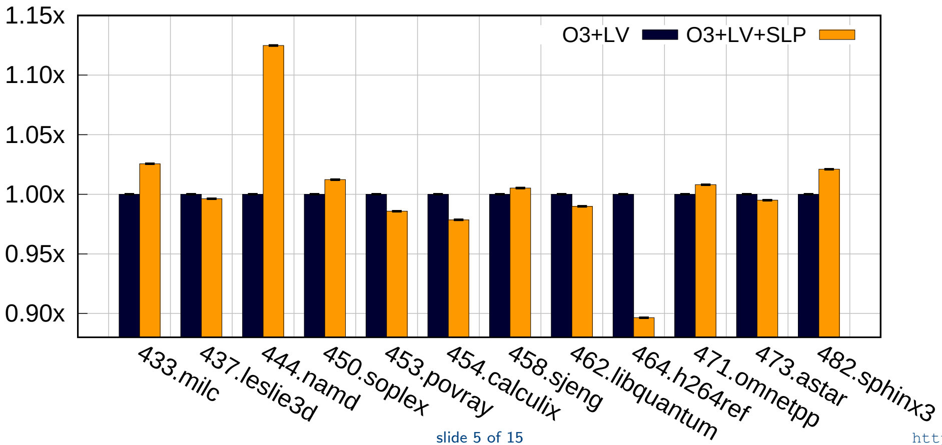
SLP Vectorizer with VF = 4

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

- Note: LV may be able to optimize the interleaved loads/stores

Isn't the Loop Vectorizer (LV) good enough ?

- SPEC 2006, 10 runs, Intel® Core™ i7-4790
- LLVM runs both LV and SLP

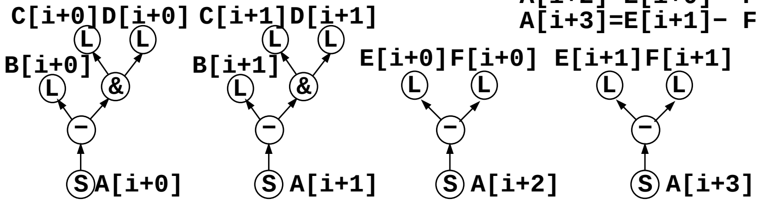


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

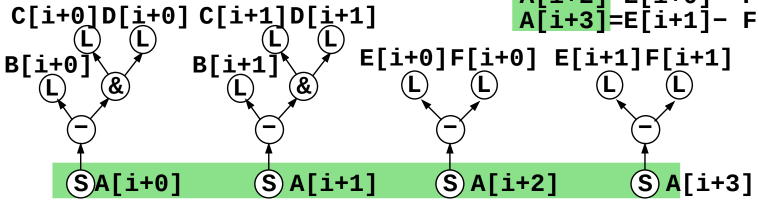
How SLP Works

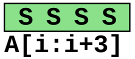
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



How SLP Works

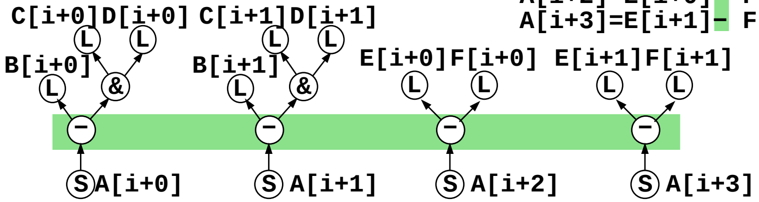
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

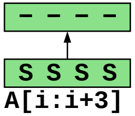




How SLP Works

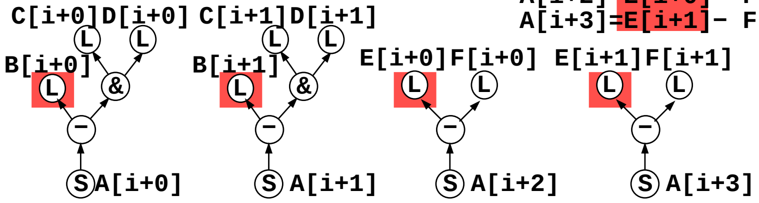
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

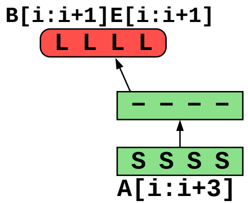




How SLP Works

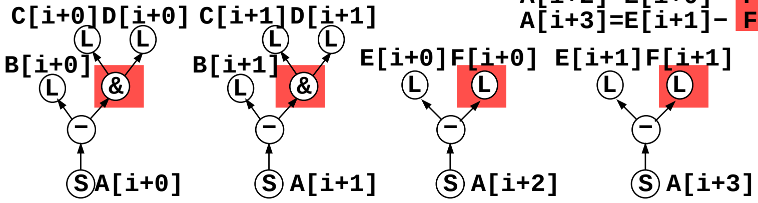
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

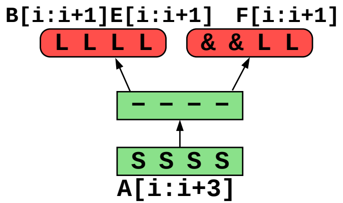




How SLP Works

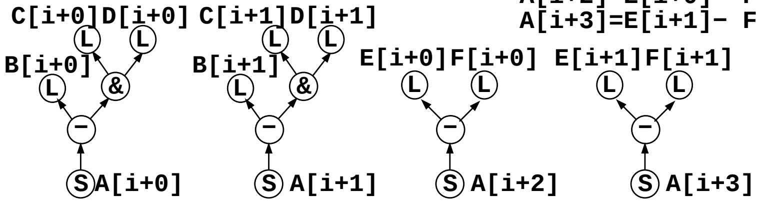
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

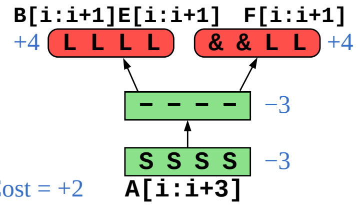




How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

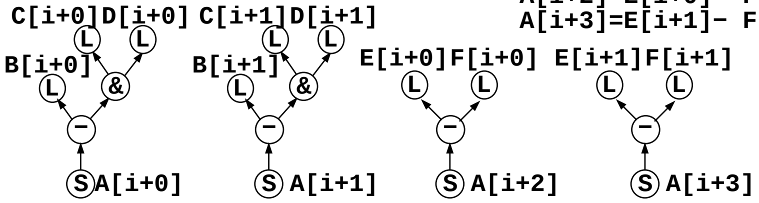


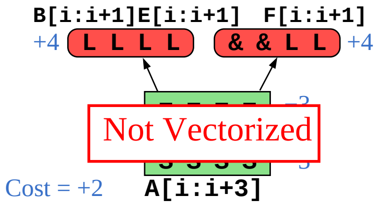


Cost = +2

How SLP Works

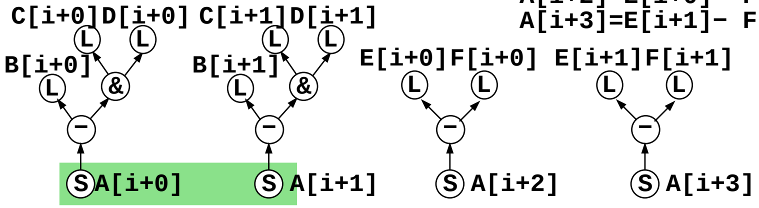
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

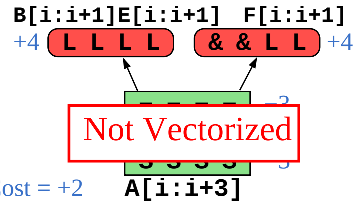




How SLP Works

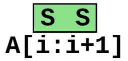
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```





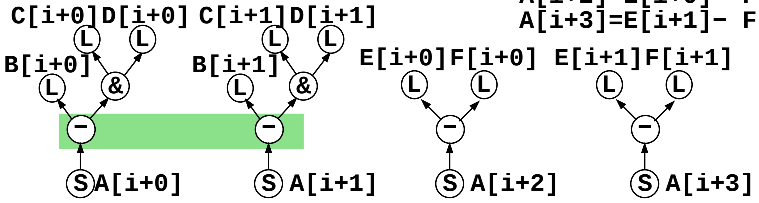
Not Vectorized

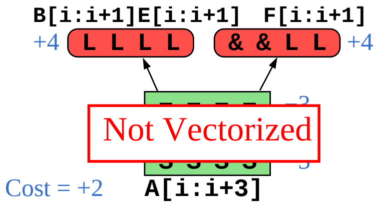
Cost = +2

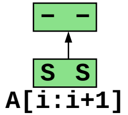


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

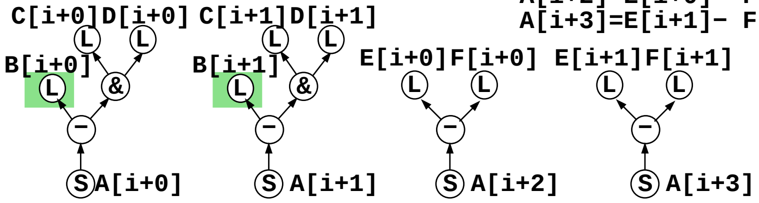


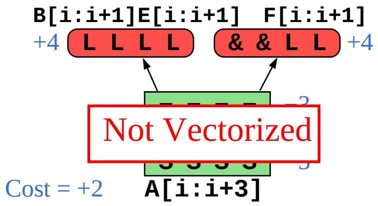


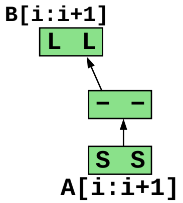


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

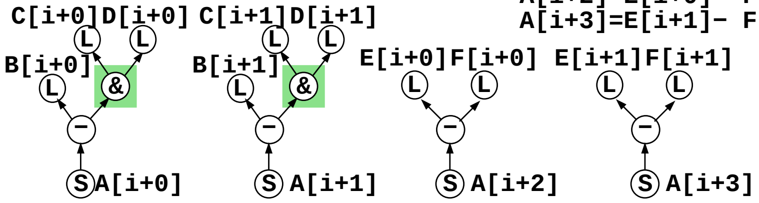


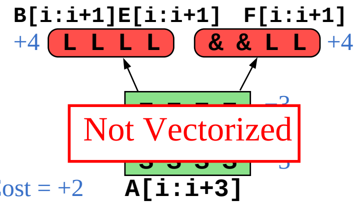




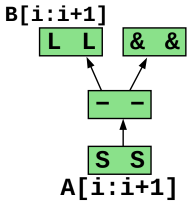
How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



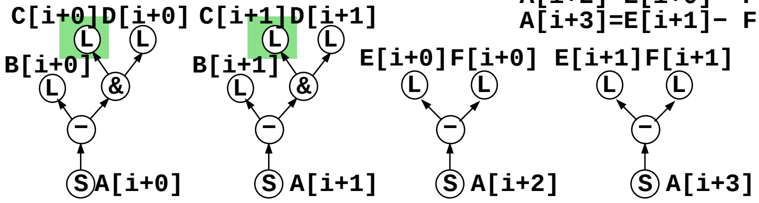


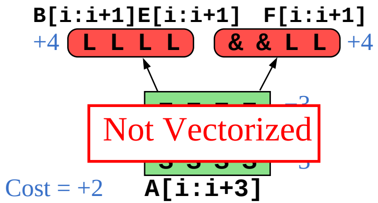
Cost = +2



How SLP Works

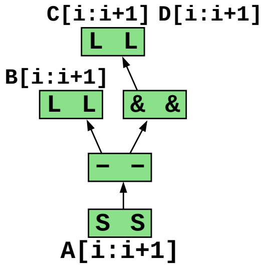
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```





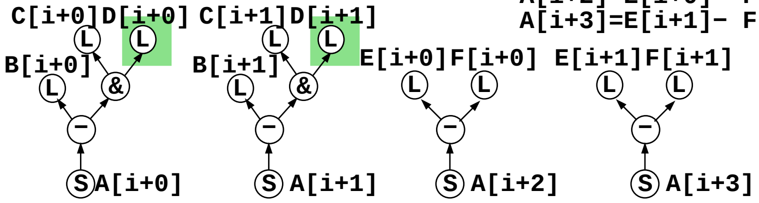
Not Vectorized

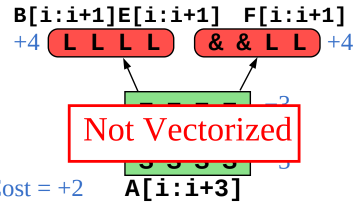
Cost = +2



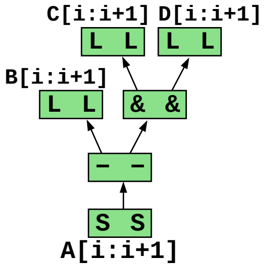
How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



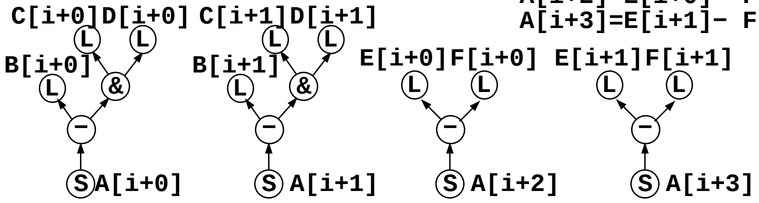


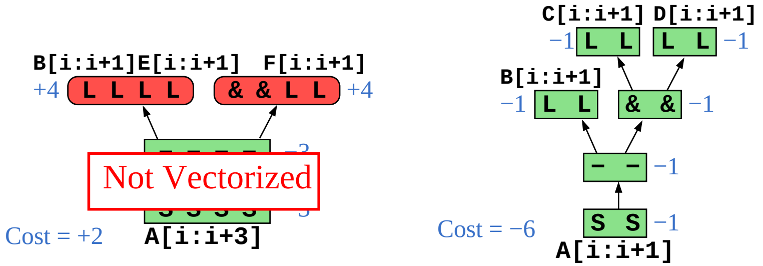
Cost = +2



How SLP Works

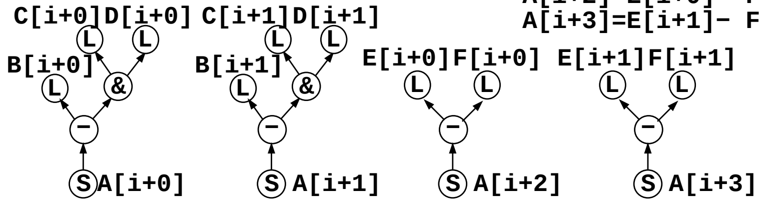
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

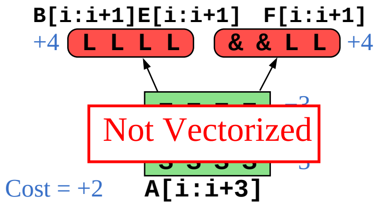


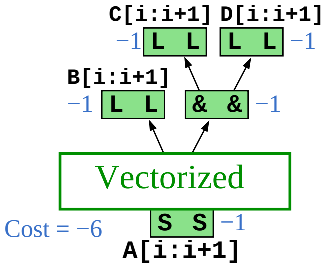


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

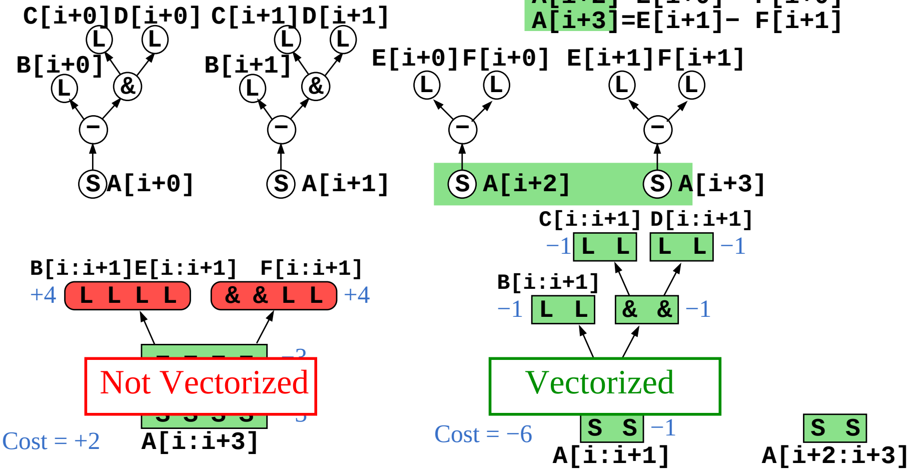

C[i+0] D[i+0] C[i+1] D[i+1]
B[i+0] B[i+1]
E[i+0] F[i+0] E[i+1] F[i+1]
L L & L L L & L L L L L
− − − −
S A[i+0] S A[i+1] S A[i+2] S A[i+3]





How SLP Works

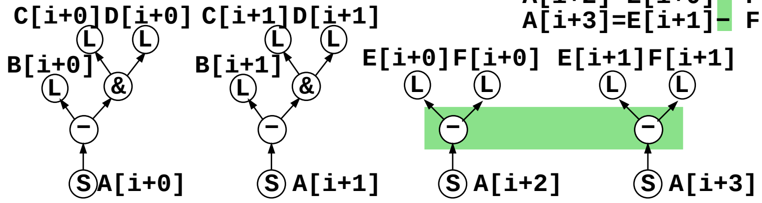
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

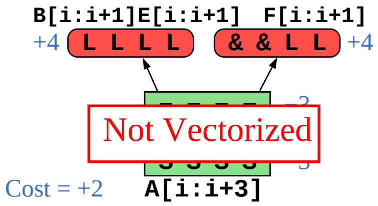


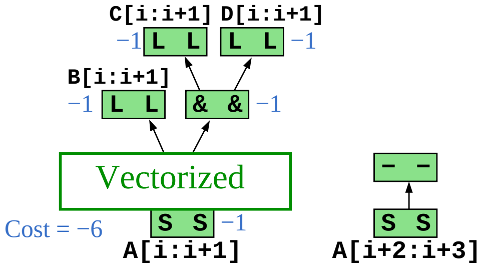
http://vporpo.me

How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

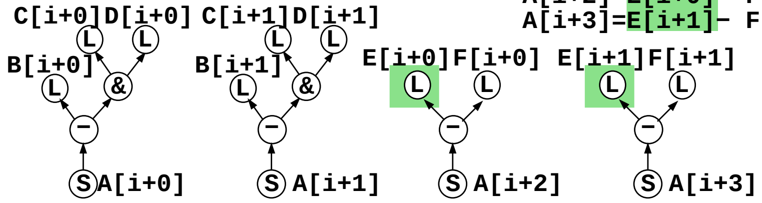


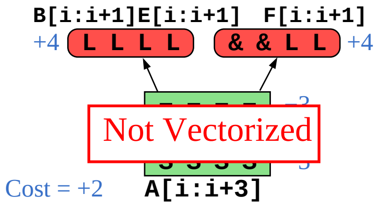


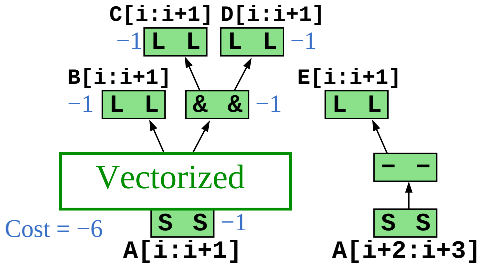


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

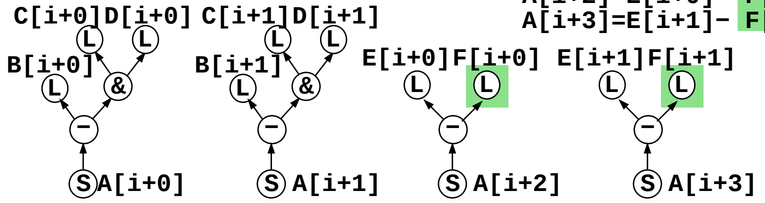


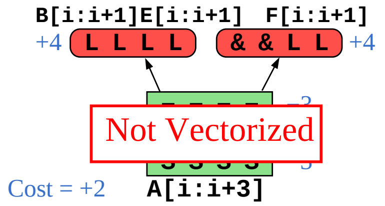


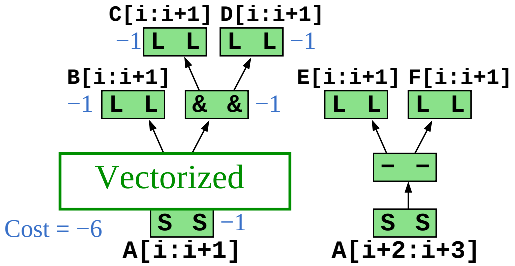


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

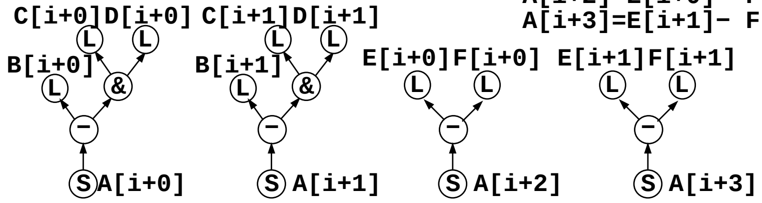


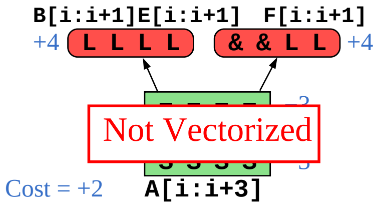


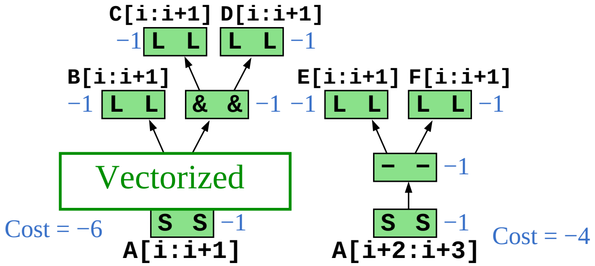


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

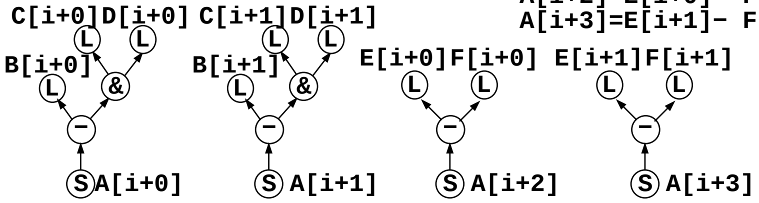

C[i+0] D[i+0] C[i+1] D[i+1]
B[i+0] B[i+1]
E[i+0] F[i+0] E[i+1] F[i+1]
A[i+0] A[i+1] A[i+2] A[i+3]

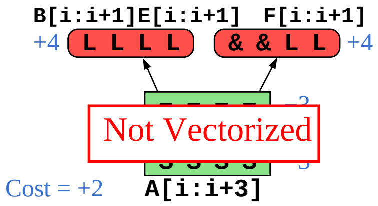


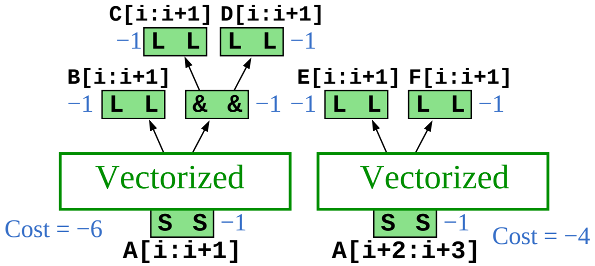


How SLP Works

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

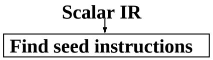




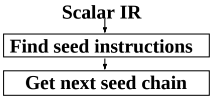


Scalar IR

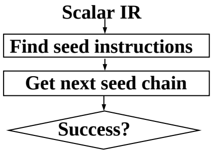
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```



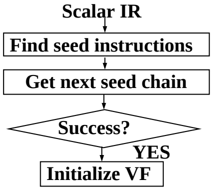
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```



```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```

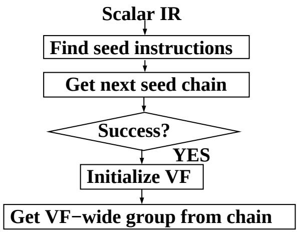


```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```



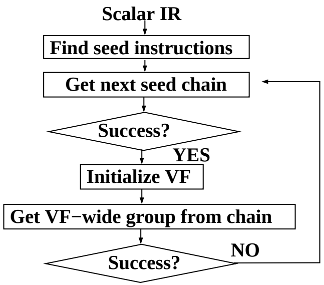
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```

VF=4



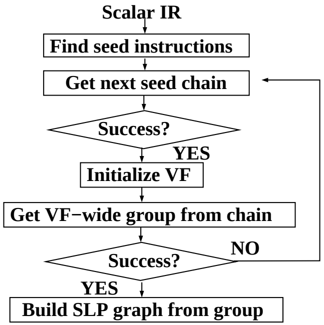
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

VF=4



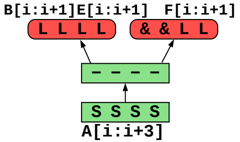
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```

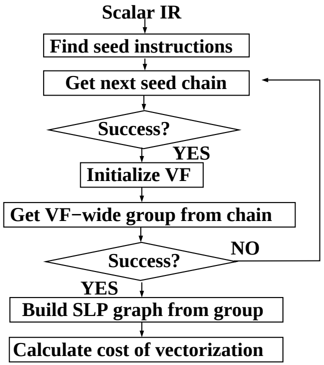
VF=4



```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

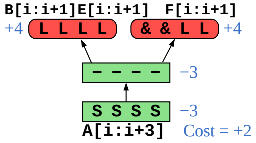
VF=4

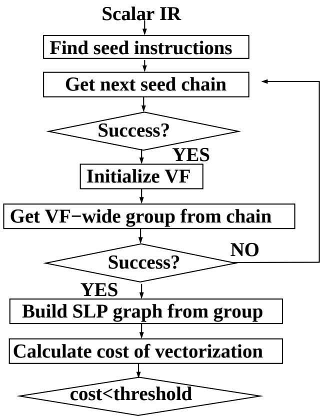




```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```

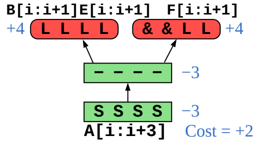
VF=4

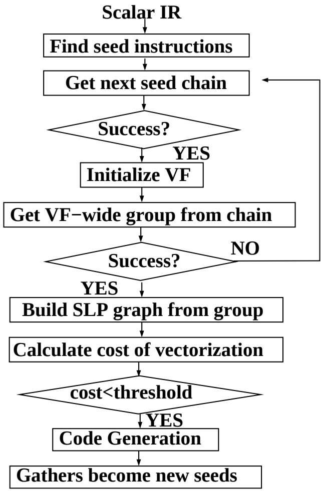




```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

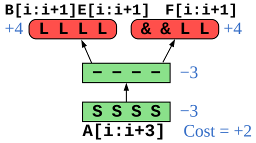
VF=4

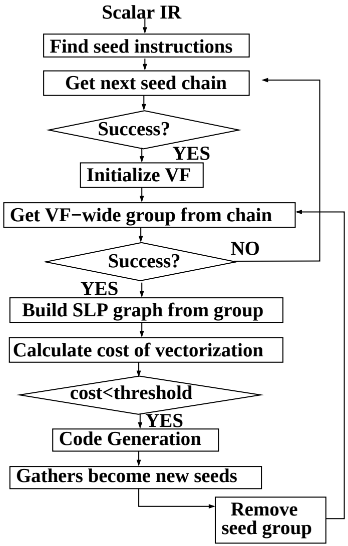




```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
  ...
```

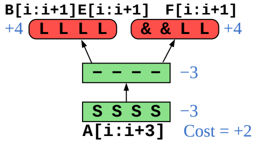
VF=4

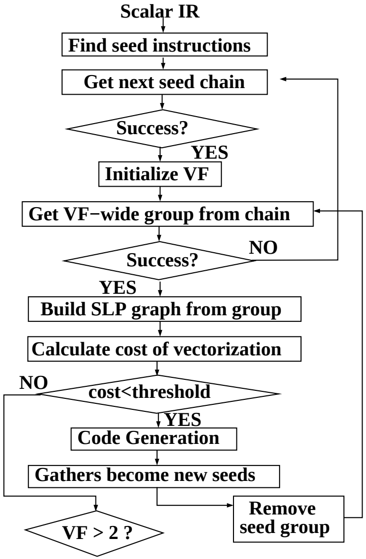




VF=4

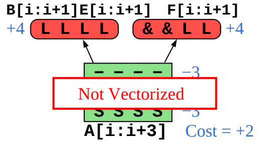
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

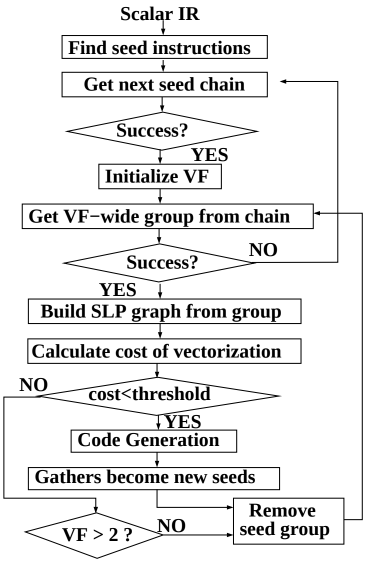




```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

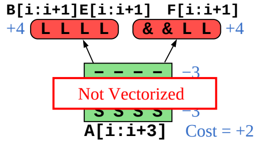
VF=4

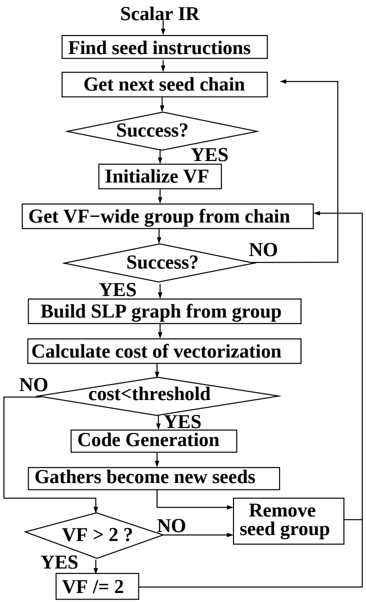




VF=4

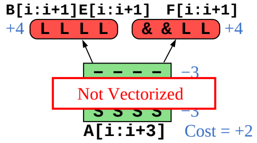
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```



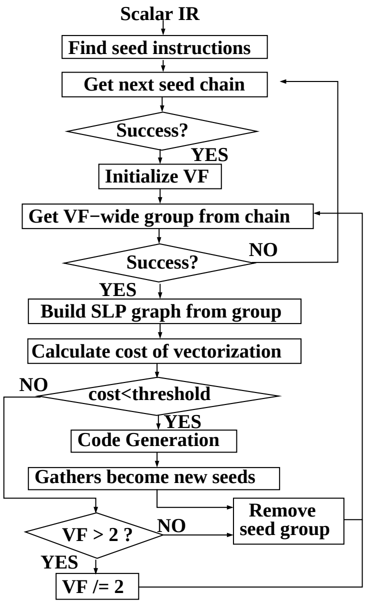


```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

VF=4

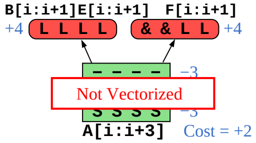


VF=2

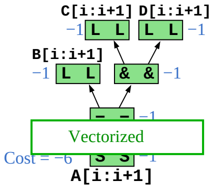


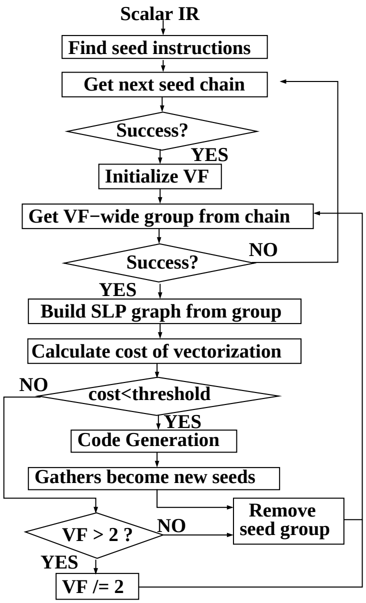
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

VF=4



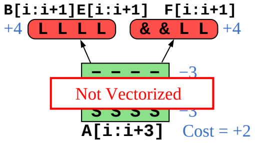
VF=2



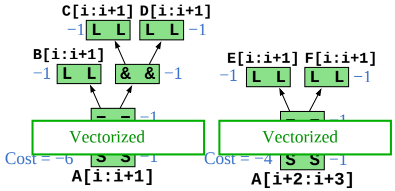


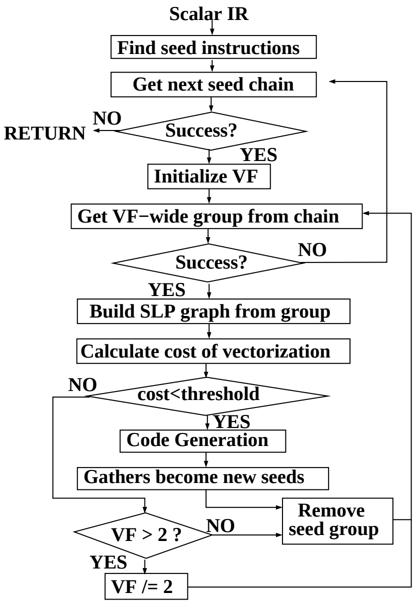
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

VF=4



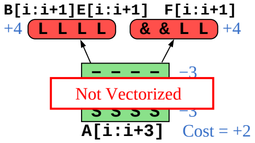
VF=2



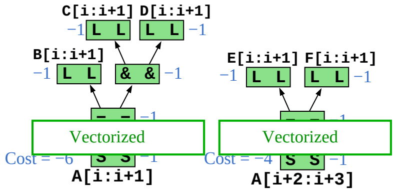


```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
 ...
```

VF=4



VF=2

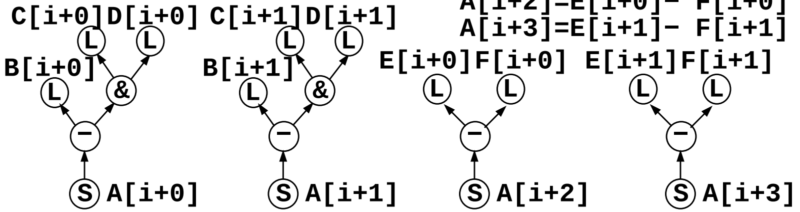


Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

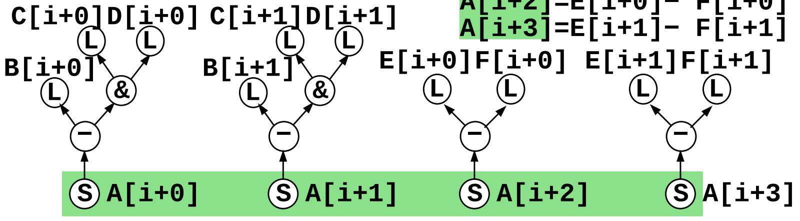
Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

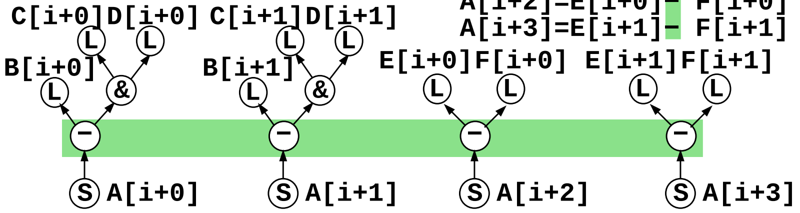


VF=4

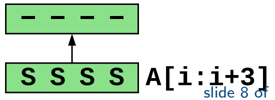
VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



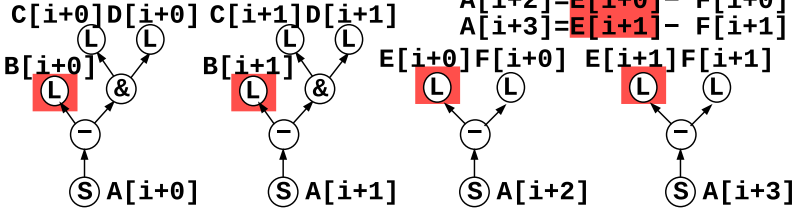
VF=4



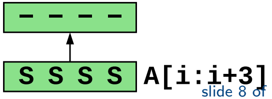
VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



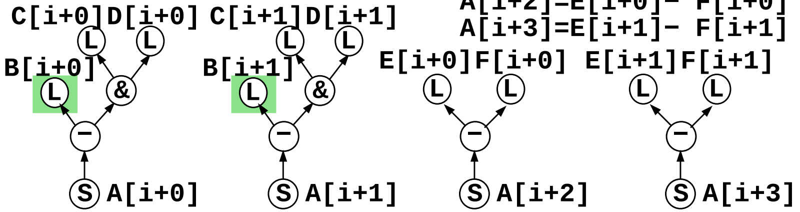
VF=4



VW−SLP

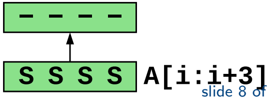
Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



VF=2

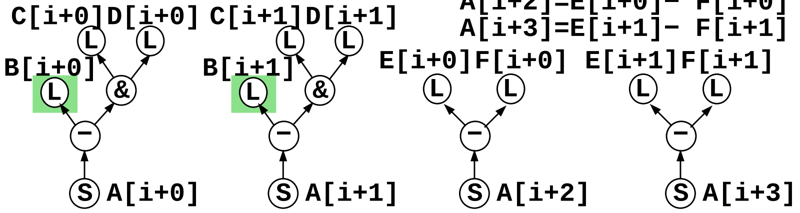
VF=4



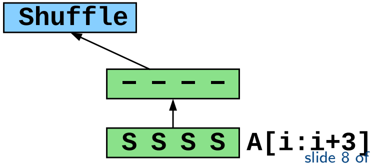
VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



VF=2

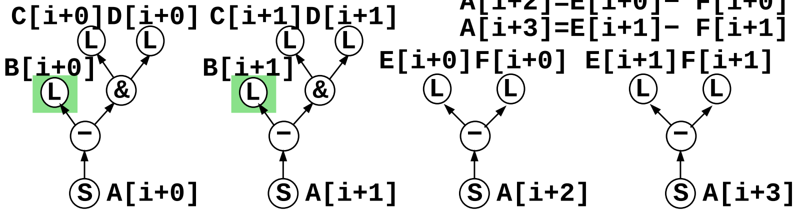


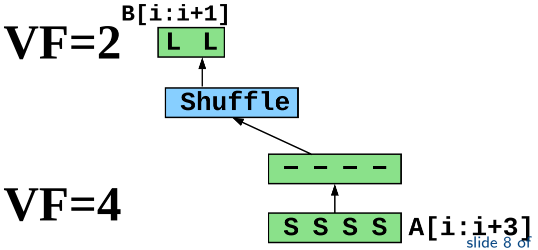
VF=4

VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

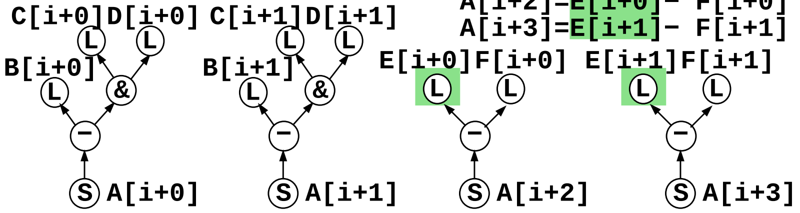


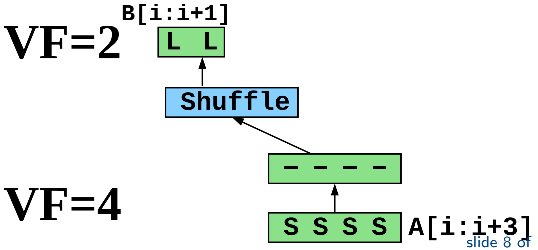


VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

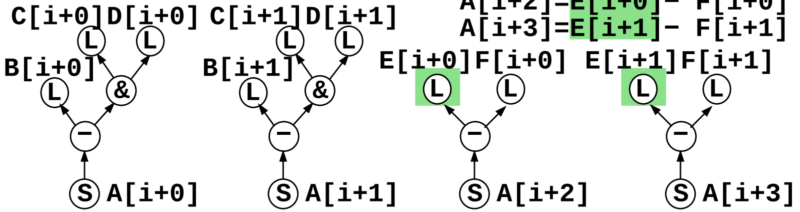


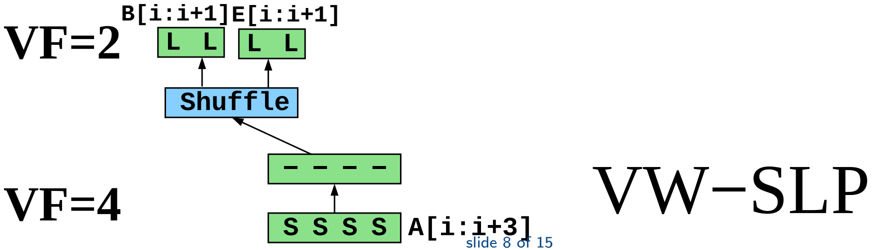


VW−SLP

Variable Width SLP

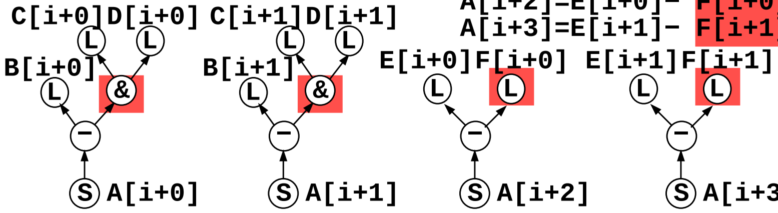
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

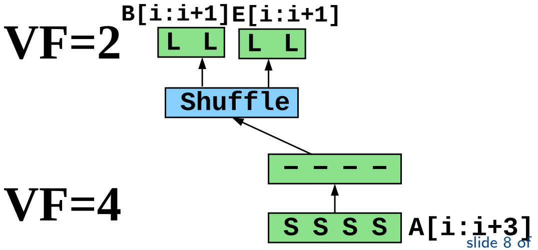




Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

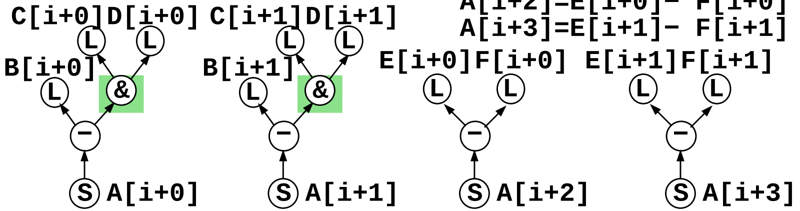


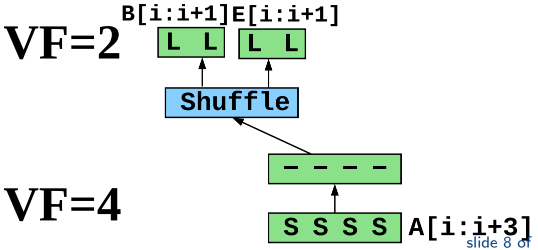


VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

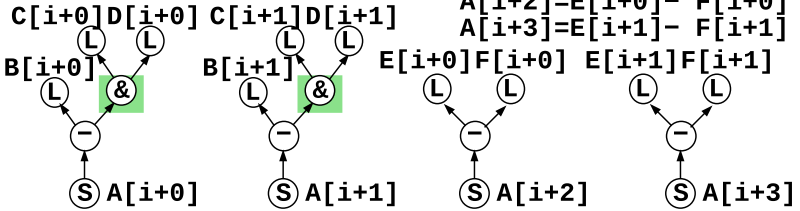


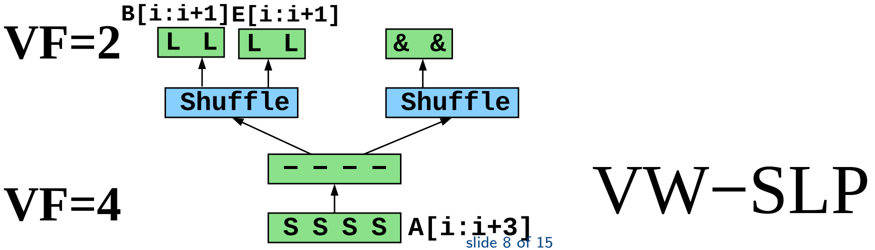


VW−SLP

Variable Width SLP

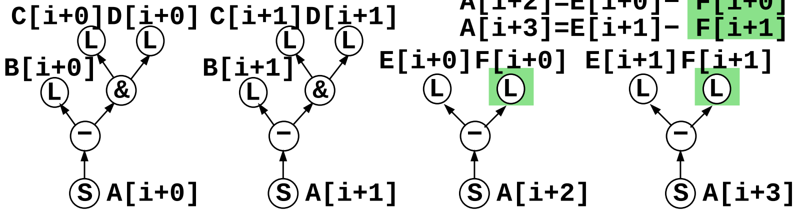
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

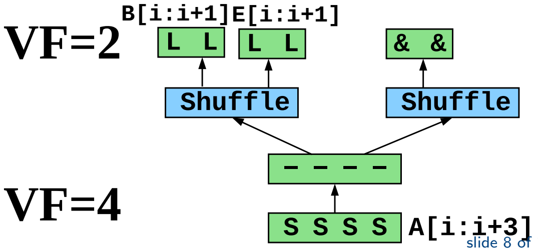




Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

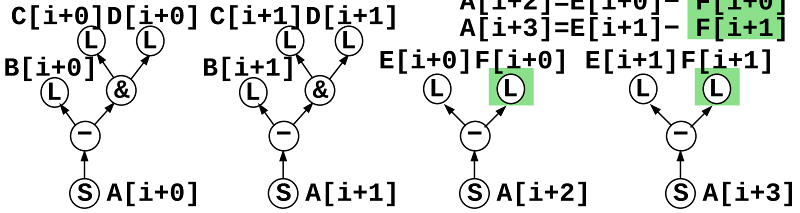


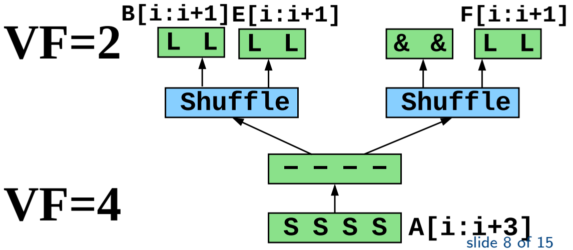


VW−SLP

Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```

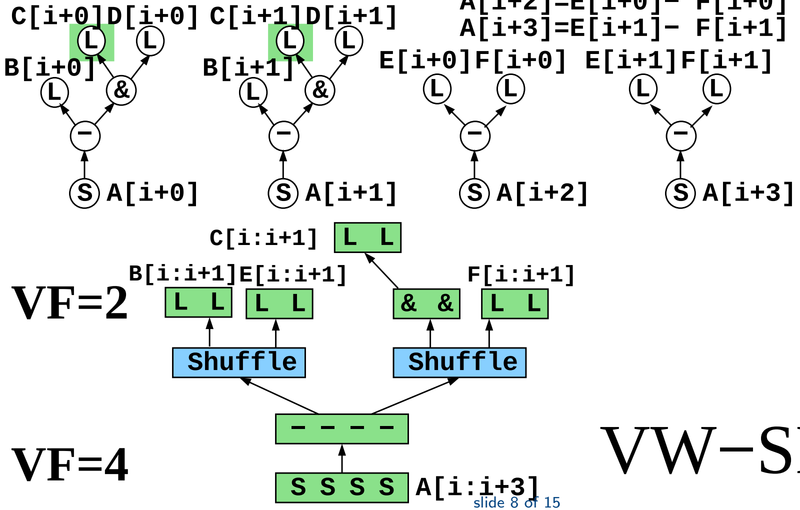




VW−SLP

Variable Width SLP

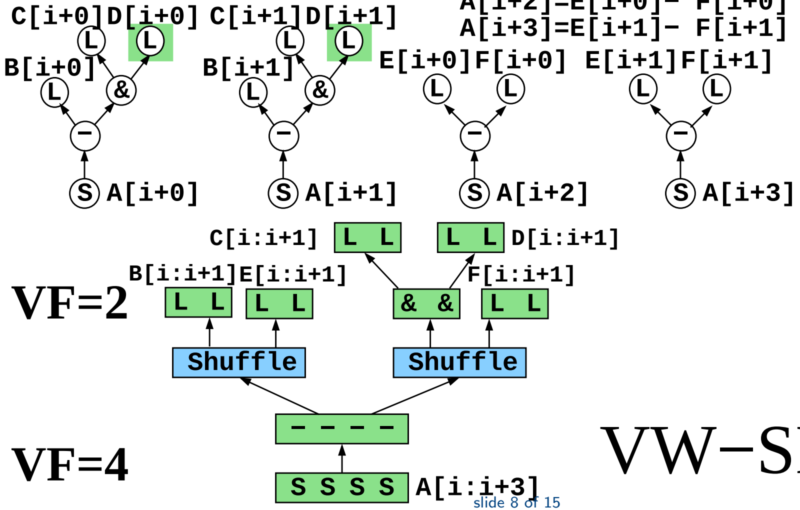
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



VW−SLP

Variable Width SLP

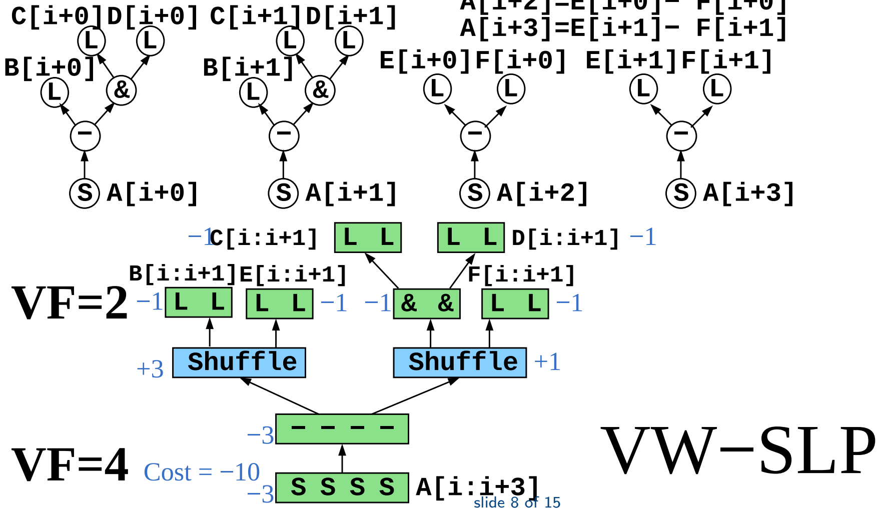
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



VW−SLP

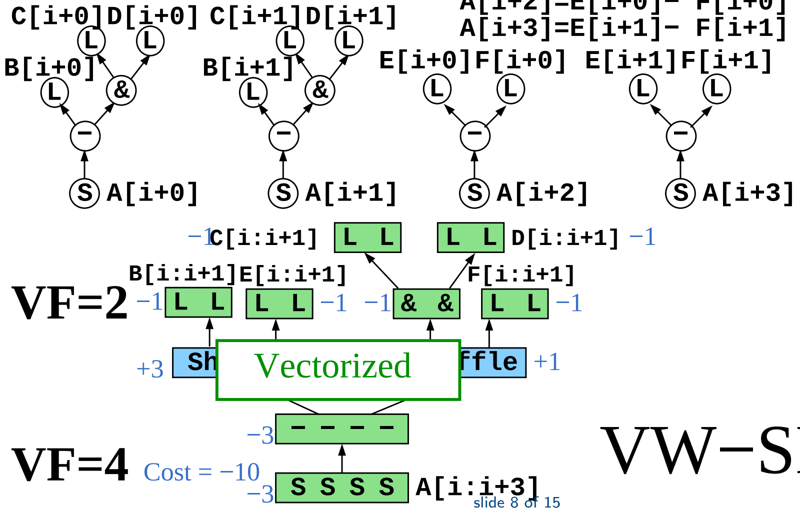
Variable Width SLP

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



Variable Width SLP

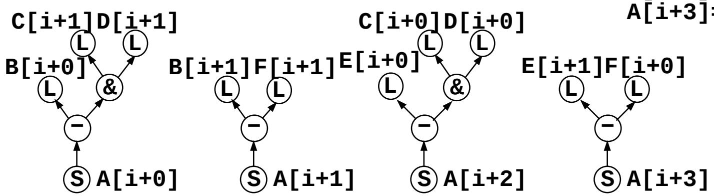
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+0]&D[i+0])
A[i+1]=B[i+1]-(C[i+1]&D[i+1])
A[i+2]=E[i+0]- F[i+0]
A[i+3]=E[i+1]- F[i+1]
```



VW−SLP

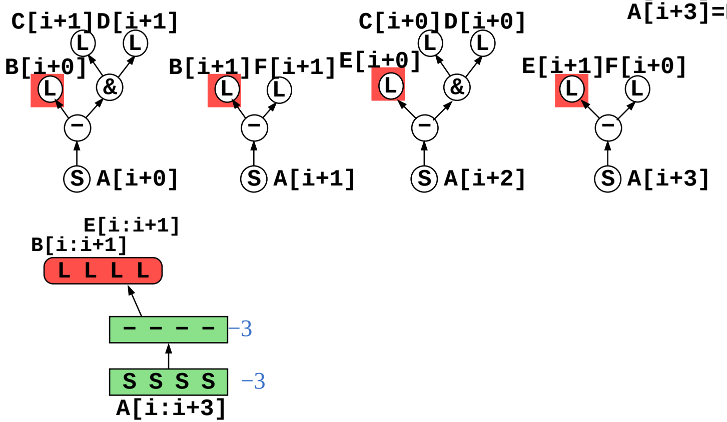
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



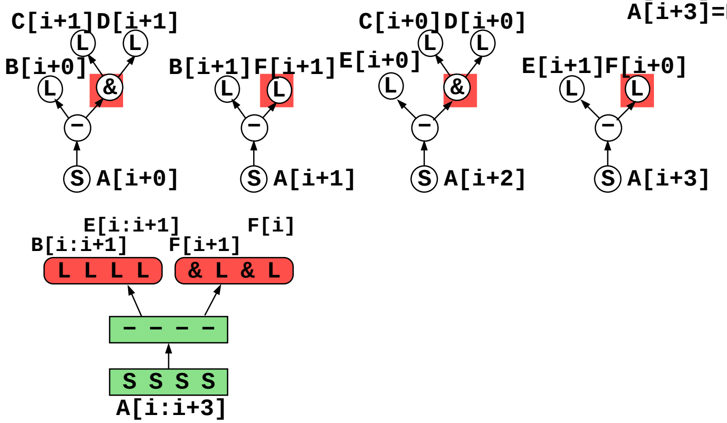
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



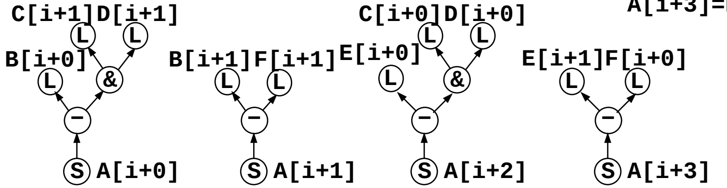
Variable Width + Permutations

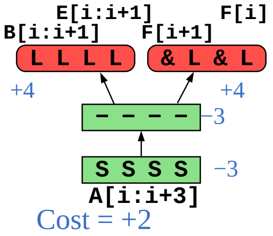
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



Variable Width + Permutations

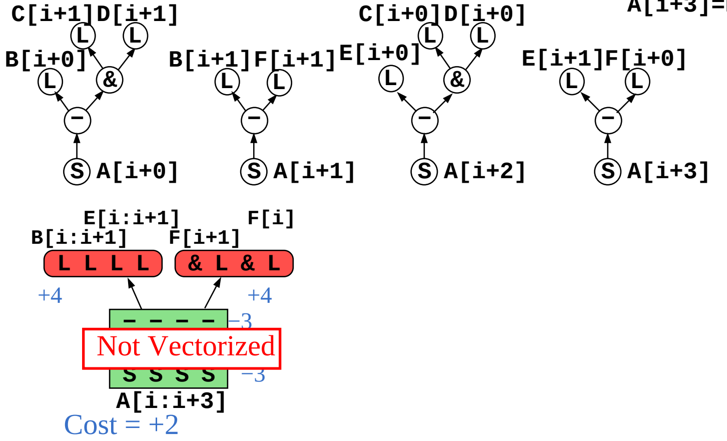
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```





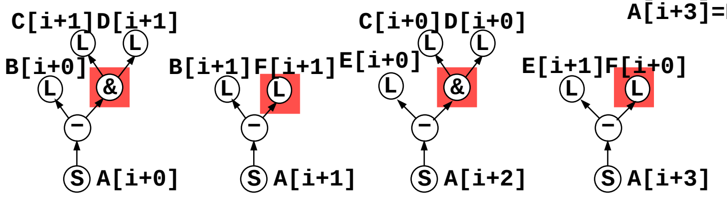
Variable Width + Permutations

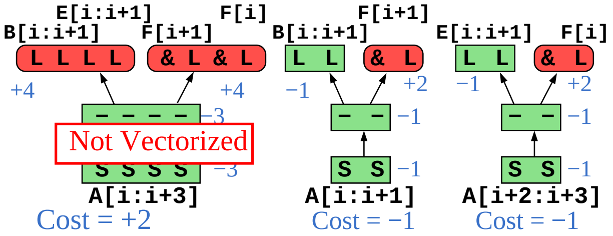
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



Variable Width + Permutations

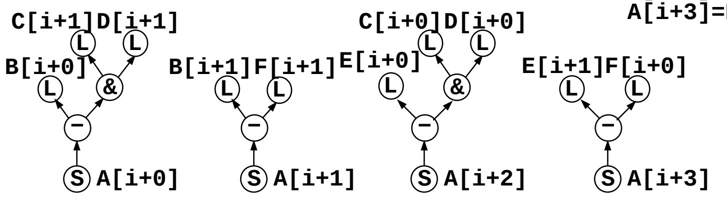
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0])
```

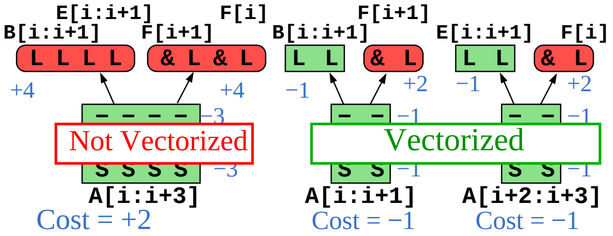




Variable Width + Permutations

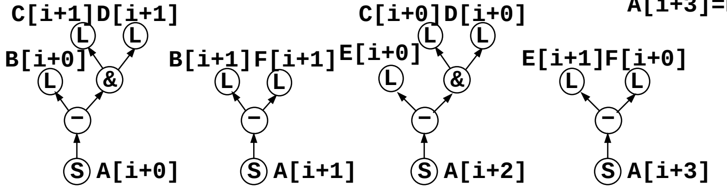
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



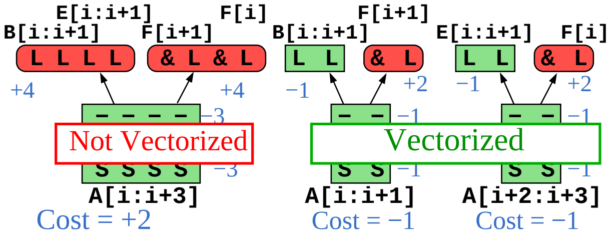


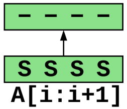
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0])
```



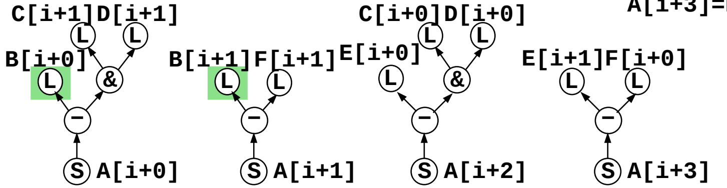
VW−SLP



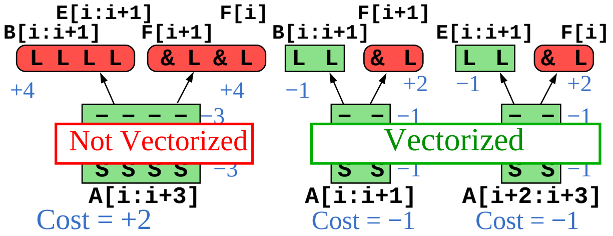


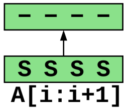
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1)
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



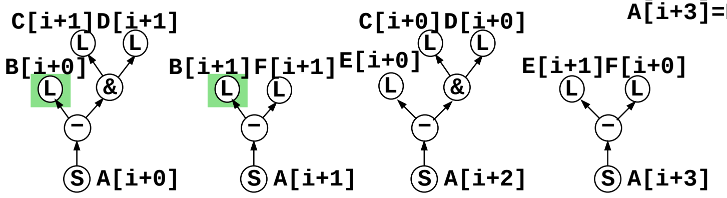
VW−SLP



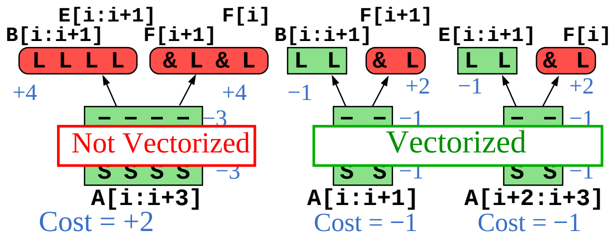


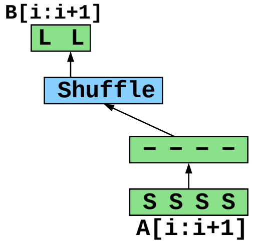
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



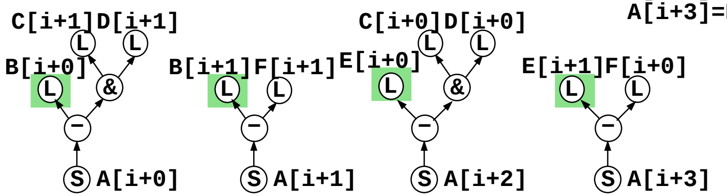
VW−SLP



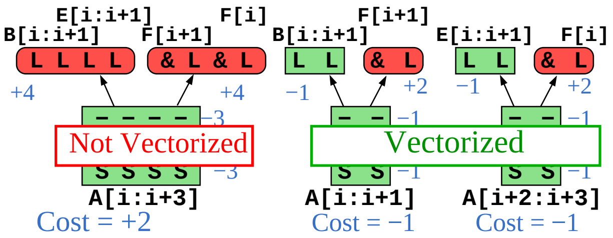


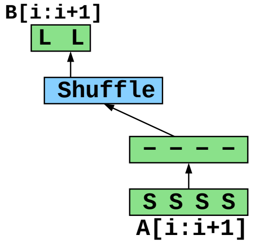
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]−(C[i+1]&D[i+1])
A[i+1]=B[i+1]− F[i+1])
A[i+2]=E[i+0]−(C[i+0]&D[i+0])
A[i+3]=E[i+1]− F[i+0]
```



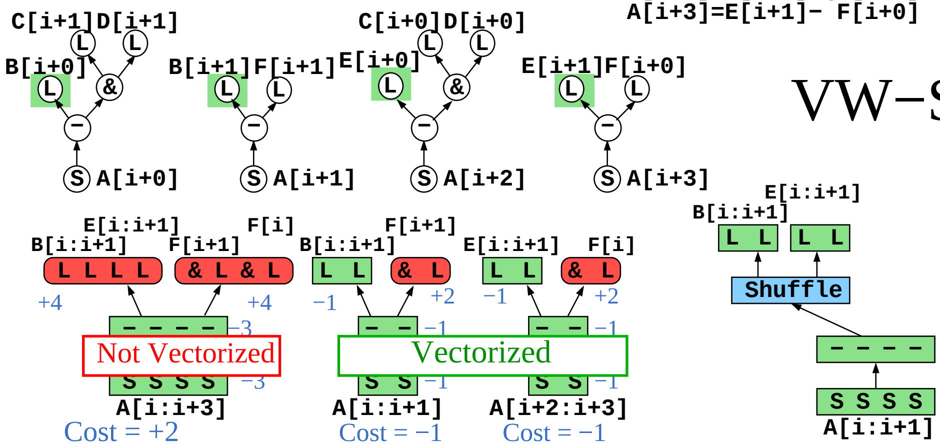
VW−SLP





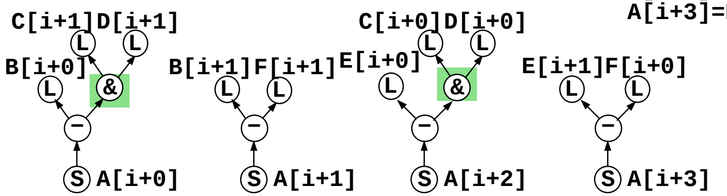
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```

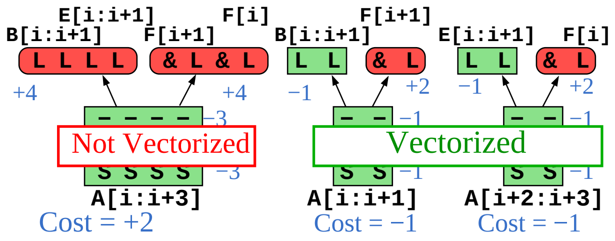


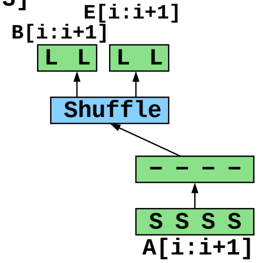
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



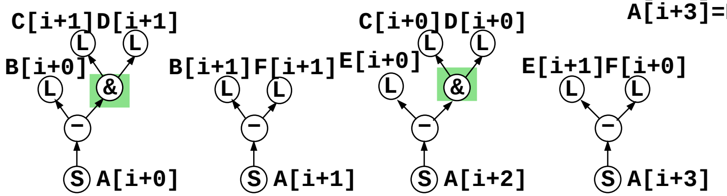
VW−SLP



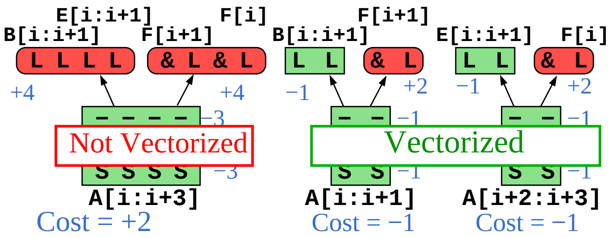


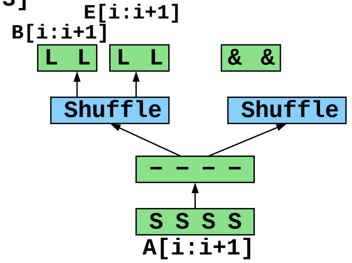
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



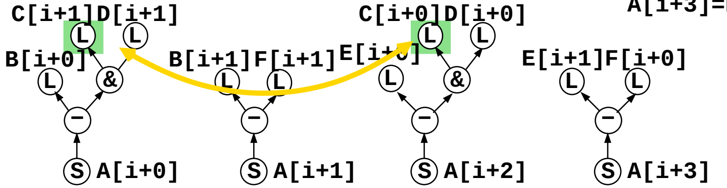
VW−SLP



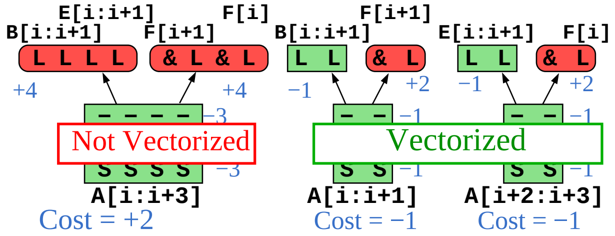


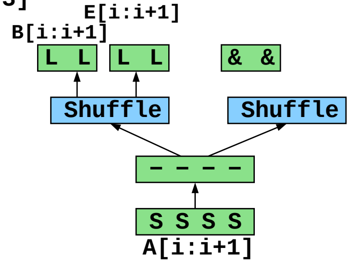
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



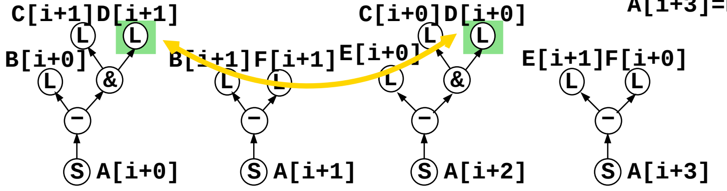
VW–SLP



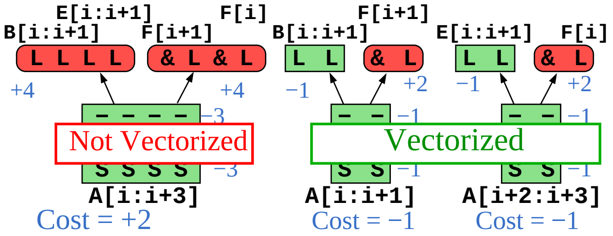


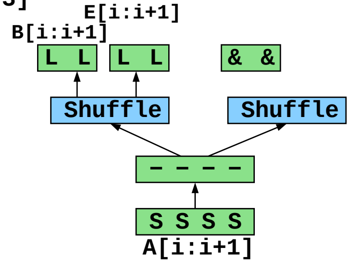
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0])
```



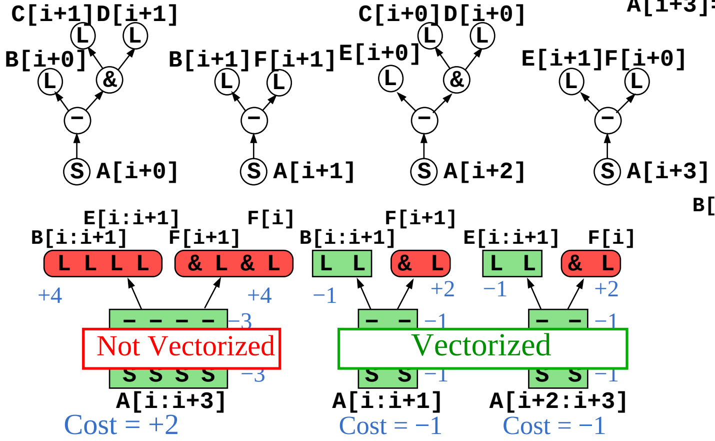
VW−SLP



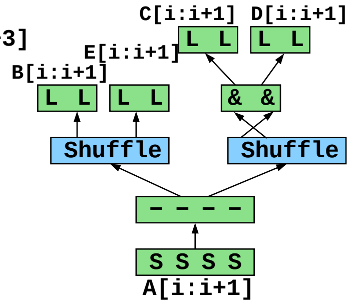


Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0])
```

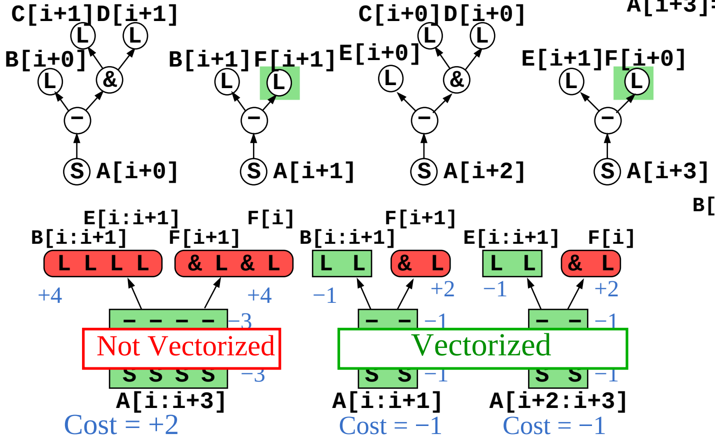


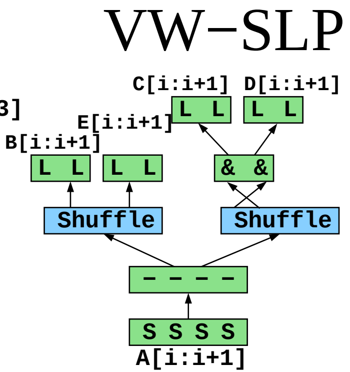
VW−SLP



Variable Width + Permutations

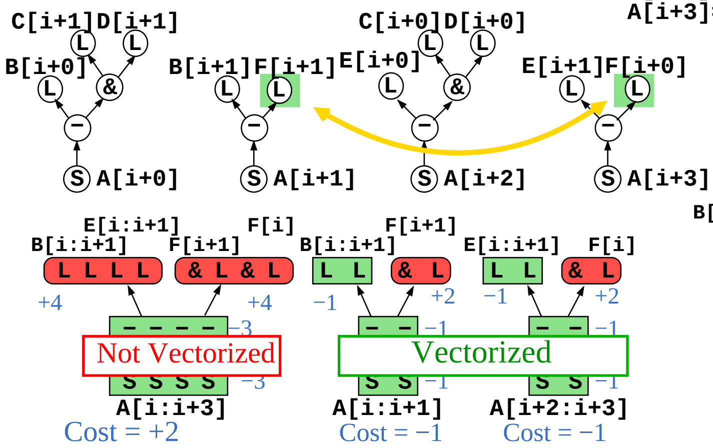
```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0])
```



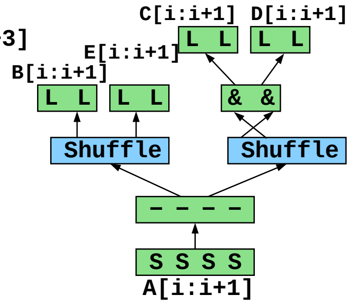


Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```

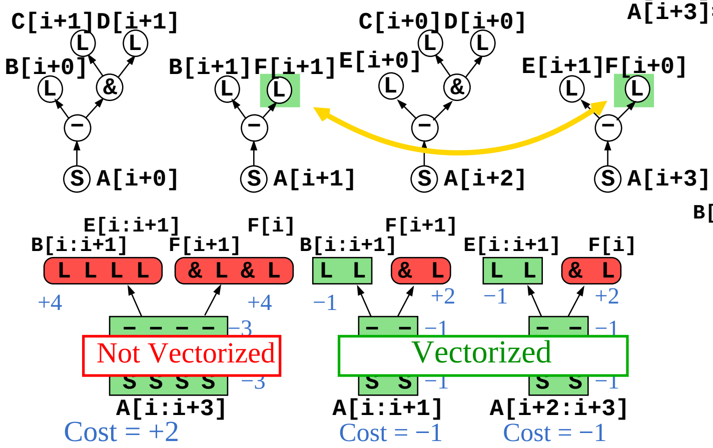


VW–SLP

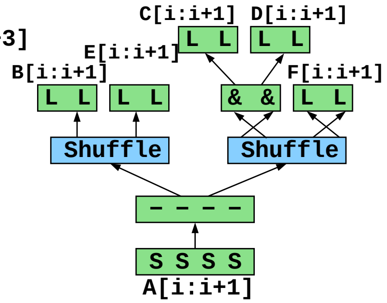


Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```

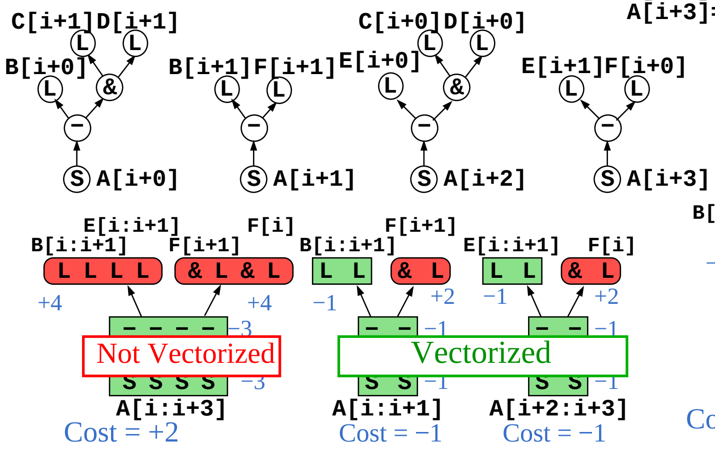


VW−SLP

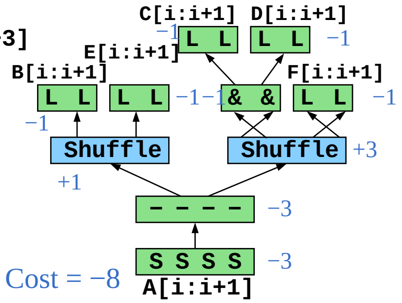


Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```

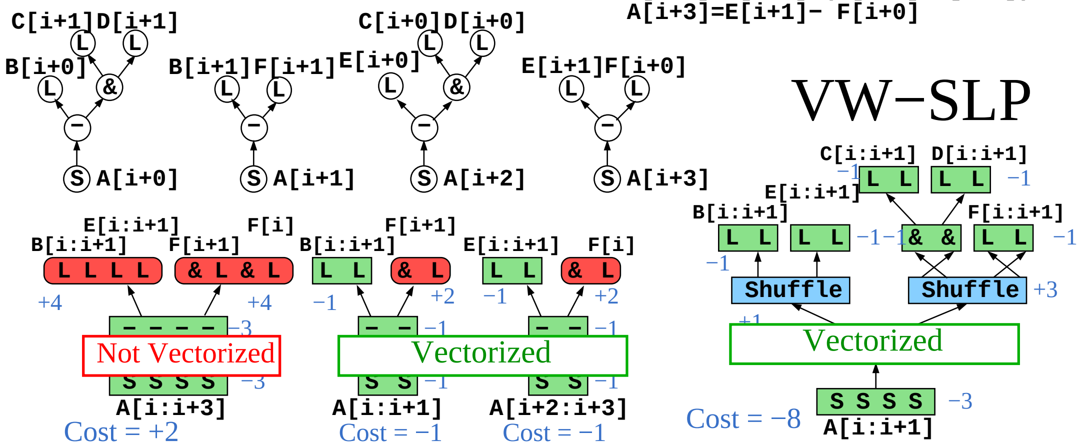


VW–SLP



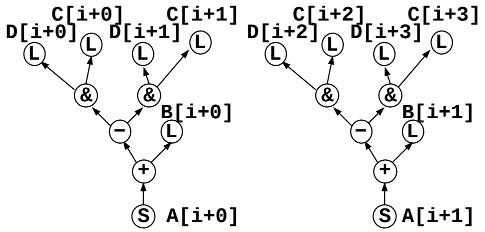
Variable Width + Permutations

```
uint64_t A[],B[],C[],D[],E[],F[]
A[i+0]=B[i+0]-(C[i+1]&D[i+1])
A[i+1]=B[i+1]- F[i+1])
A[i+2]=E[i+0]-(C[i+0]&D[i+0])
A[i+3]=E[i+1]- F[i+0]
```



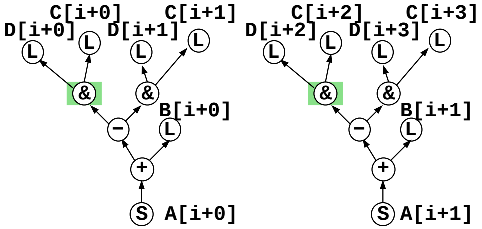
Widening the Vector Width

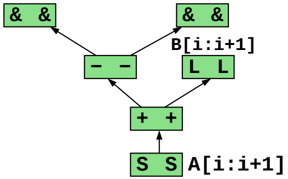
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```



Widening the Vector Width

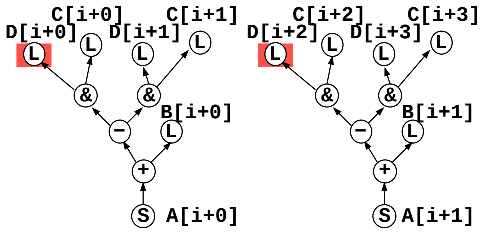
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

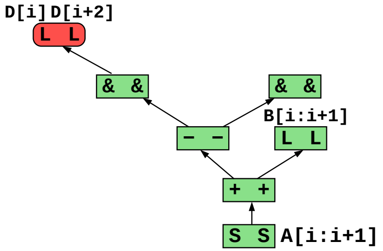




Widening the Vector Width

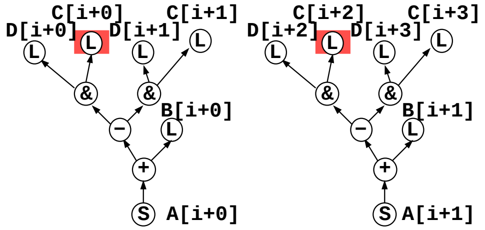
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

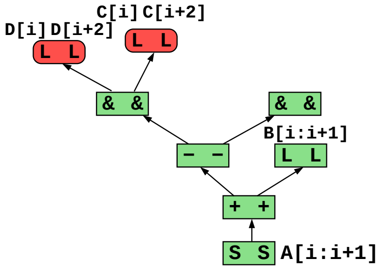




Widening the Vector Width

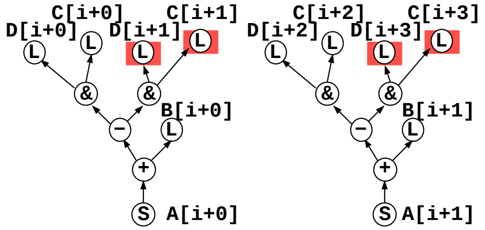
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

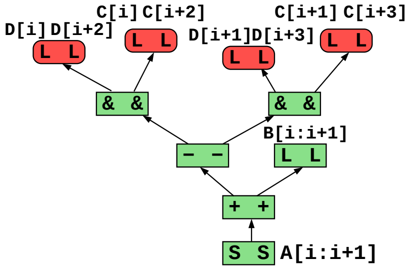




Widening the Vector Width

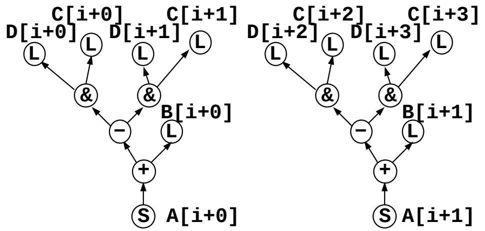
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0-tmp1)
A[i+1]=B[i+1]+(tmp2-tmp3)
```

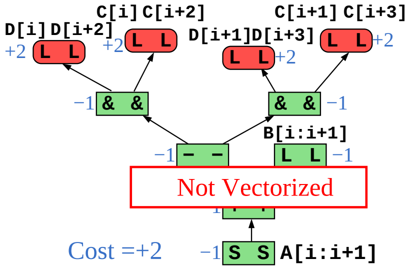




Widening the Vector Width

```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

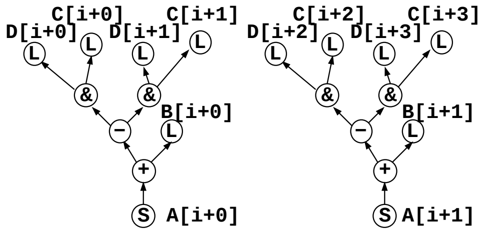


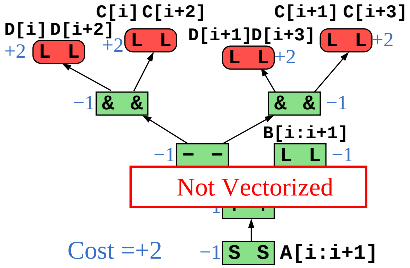


Cost =+2

Widening the Vector Width

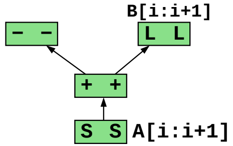
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```





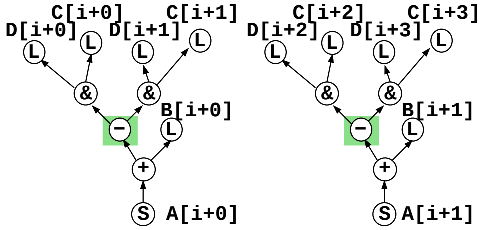
VW-SLP

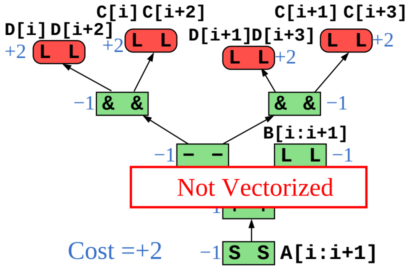
VF=2



Widening the Vector Width

```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0-tmp1)
A[i+1]=B[i+1]+(tmp2-tmp3)
```

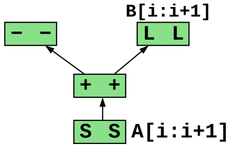




Cost =+2

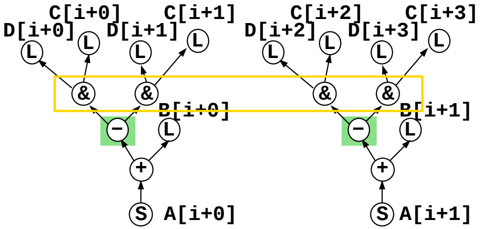
VW-SLP

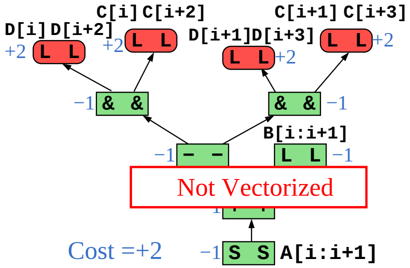
VF=2



Widening the Vector Width

```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0-tmp1)
A[i+1]=B[i+1]+(tmp2-tmp3)
```

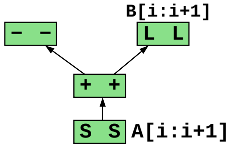




Cost =+2

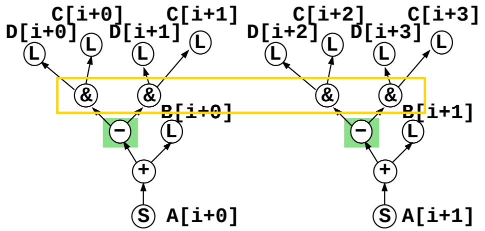
VW-SLP

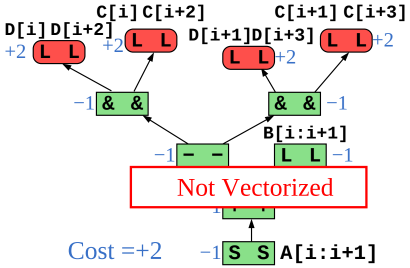
VF=2



Widening the Vector Width

```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0-tmp1)
A[i+1]=B[i+1]+(tmp2-tmp3)
```



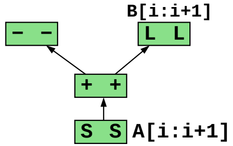


Cost =+2

VW-SLP

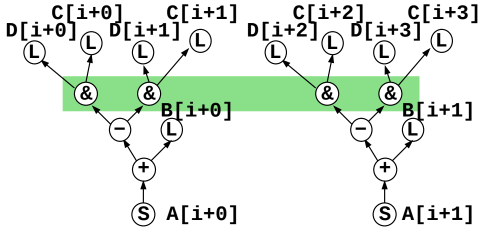
VF=4

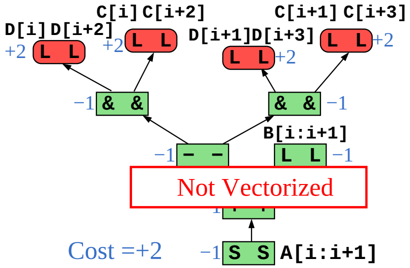
VF=2



Widening the Vector Width

```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

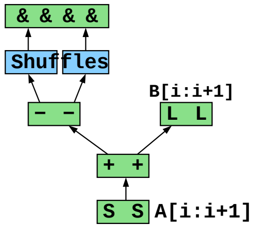




VW-SLP

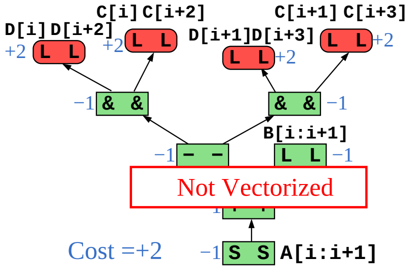
VF=4

VF=2



Widening the Vector Width

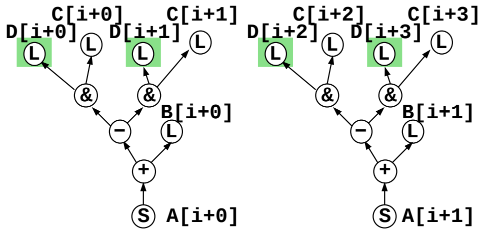
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

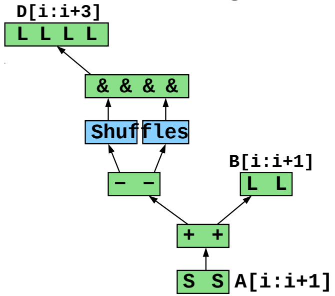


VW-SLP

VF=4

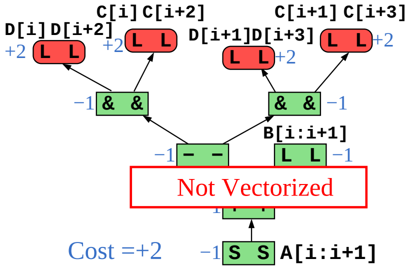
VF=2

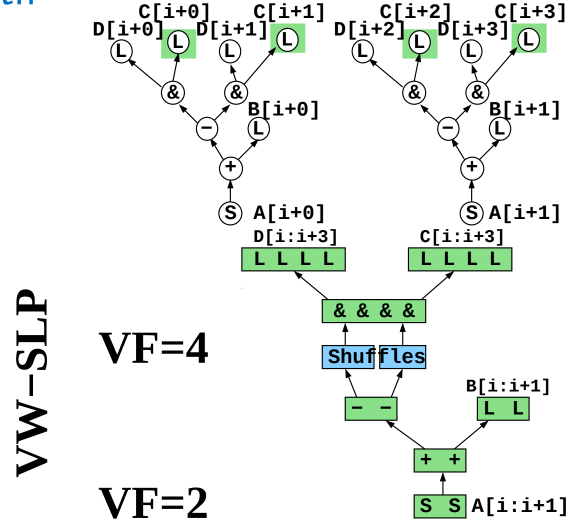




Widening the Vector Width

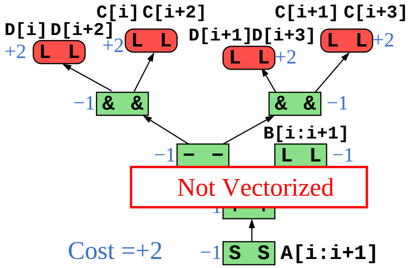
```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```

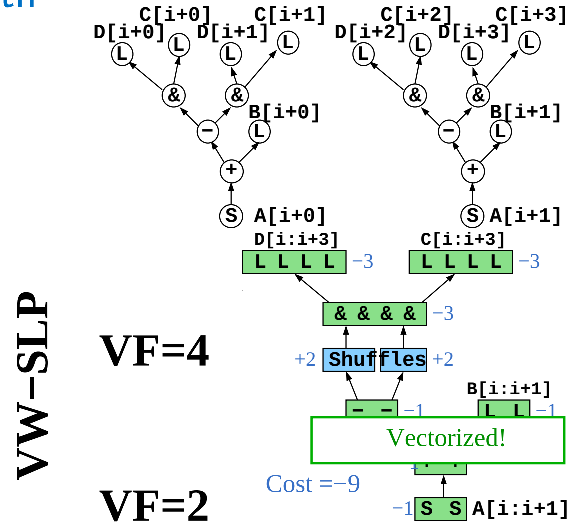




Widening the Vector Width

```
uint64_t A[],B[],C[],D[]
uint64_t tmp0=C[i+0]&D[i+0]
uint64_t tmp1=C[i+1]&D[i+1]
uint64_t tmp2=C[i+2]&D[i+2]
uint64_t tmp3=C[i+3]&D[i+3]
A[i+0]=B[i+0]+(tmp0−tmp1)
A[i+1]=B[i+1]+(tmp2−tmp3)
```





Implementation

- Modified `buildTree_rec()` to handle variable-width

Implementation

- Modified `buildTree_rec()` to handle variable-width
- Spawn localized `buildTree_rec(NewOp)` searches with shorter/wider/permuted `NewOp`

Implementation

- Modified `buildTree_rec()` to handle variable-width
- Spawn localized `buildTree_rec(NewOp)` searches with shorter/wider/permuted `NewOp`
- Cap the `buildTree_rec()` to a maximum exploration depth.

Implementation

- Modified `buildTree_rec()` to handle variable-width
- Spawn localized `buildTree_rec(NewOp)` searches with shorter/wider/permuted `NewOp`
- Cap the `buildTree_rec()` to a maximum exploration depth.
- Evaluate the cost of each new sub-tree using `getEntryCost()` and pick the `NewOp` with the best cost

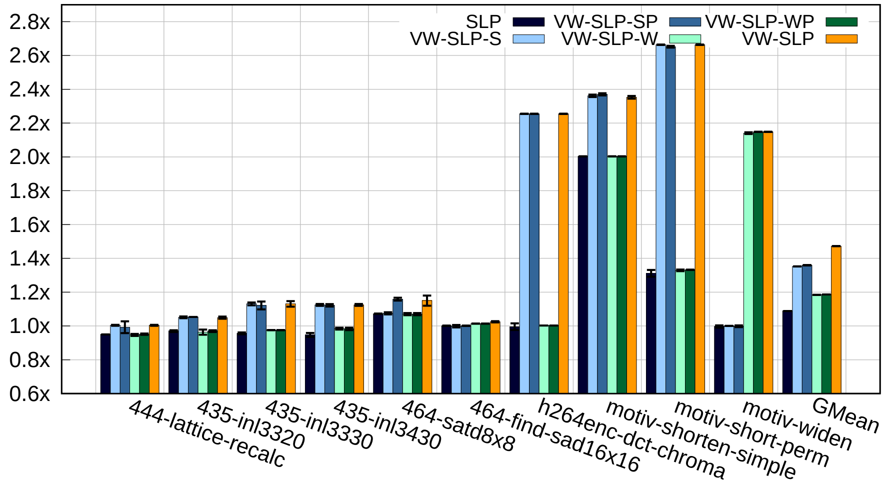
Implementation

- Modified `buildTree_rec()` to handle variable-width
- Spawn localized `buildTree_rec(NewOp)` searches with shorter/wider/permuted `NewOp`
- Cap the `buildTree_rec()` to a maximum exploration depth.
- Evaluate the cost of each new sub-tree using `getEntryCost()` and pick the `NewOp` with the best cost
- Added scheduler support for roll-back and replay

Implementation

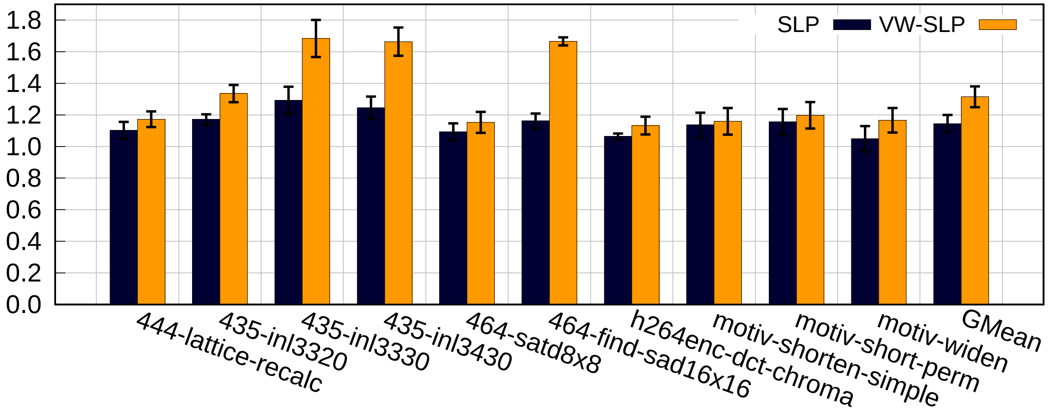
- Modified `buildTree_rec()` to handle variable-width
- Spawn localized `buildTree_rec(NewOp)` searches with shorter/wider/permuted `NewOp`
- Cap the `buildTree_rec()` to a maximum exploration depth.
- Evaluate the cost of each new sub-tree using `getEntryCost()` and pick the `NewOp` with the best cost
- Added scheduler support for roll-back and replay
- Code generation changes in `vectorizeTree()` for variable-width support and the `shufflevector` instructions

Performance



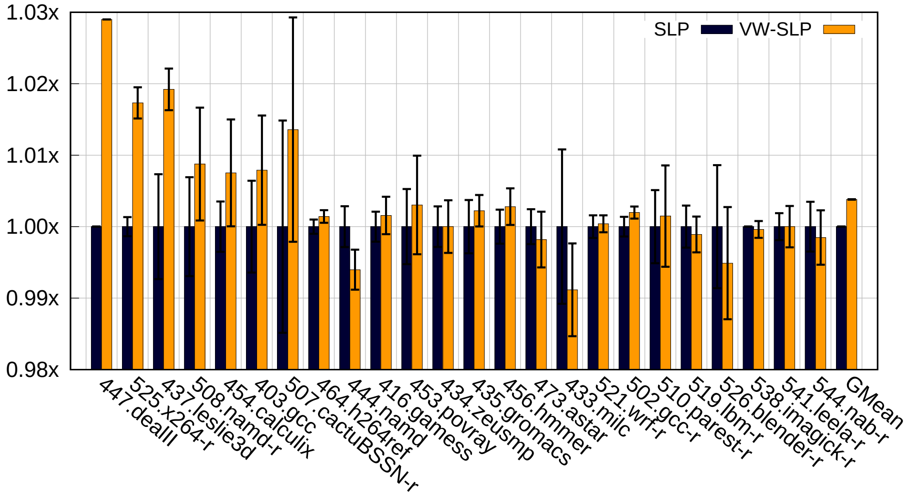
- Functions extracted from SPEC'06 and MediabenchII, Normalized to O3 and tested on a Core™ i5-6440HQ

Compilation Time



- Core^{TM} i5-6440HQ

Performance on SPEC



- SPEC 2006/2017 SLP(No LV), VW-SLP(No LV), on Core™ i5-6440HQ

Conclusion

- Presented SLP with variable vector length

Conclusion

- Presented SLP with variable vector length
- More effective SLP graph

Conclusion

- Presented SLP with variable vector length
- More effective SLP graph
- Better performance and coverage

Conclusion

- Presented SLP with variable vector length
- More effective SLP graph
- Better performance and coverage
- Very small increase in compilation time