PSLP: Padded SLP Automatic Vectorization

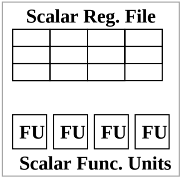
Vasileios Porpodas[†], Alberto Magni[‡] and Timothy M. Jones[†]

University of Cambridge[†]
University of Edinburgh[‡]

EuroLLVM APR 2015

Why SIMD Vectorization?

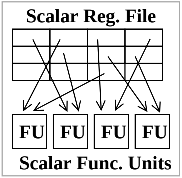
- Scalable parallelism



a. ILP

Why SIMD Vectorization?

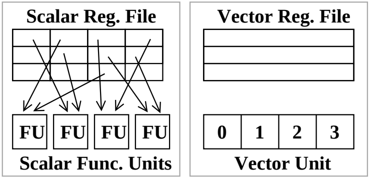
- Scalable parallelism



a. ILP

Why SIMD Vectorization?

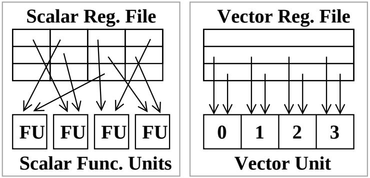
- Scalable parallelism



a. ILP
b. Vector Parallelism

Why SIMD Vectorization?

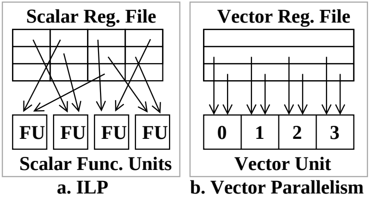
- Scalable parallelism



a. ILP b. Vector Parallelism

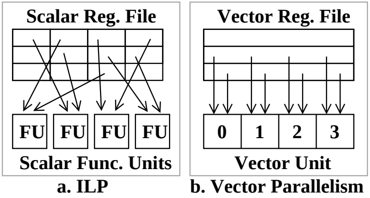
Why SIMD Vectorization?

- Scalable parallelism
- High Performance



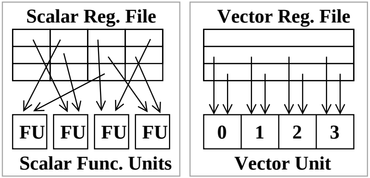
Why SIMD Vectorization?

- Scalable parallelism
- High Performance
- Energy efficiency



Why SIMD Vectorization?

- Scalable parallelism
- High Performance
- Energy efficiency
- Supported since mid 90's
- Frequent updates of vector ISAs



a. ILP

b. Vector Parallelism

AMD K6-2 PROCESSOR 3DNOW!

ARM NEON™

AVX2

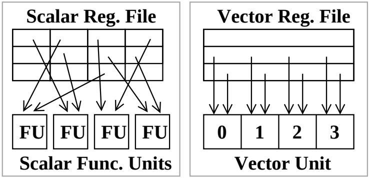
MMX™ intel inside pentium PROCESSOR

altivec™ technology

SSE4

Why SIMD Vectorization?

- Scalable parallelism
- High Performance
- Energy efficiency
- Supported since mid 90's
- Frequent updates of vector ISAs
- Vector generation not done in hardware
- Low-level programming or capable compiler



a. ILP b. Vector Parallelism

AMD K6-2 PROCESSOR 3DNow!

ARM NEON™

AVX2

MMX™ intel inside pentium PROCESSOR

altivec™ technology

SSE4

SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI'00]

SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI'00]
- State-of-the-art straight-line code vectorizer

SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI'00]
- State-of-the-art straight-line code vectorizer
- Implemented in most compilers (including GCC and LLVM)

SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI'00]
- State-of-the-art straight-line code vectorizer
- Implemented in most compilers (including GCC and LLVM)
- In theory it should be a superset of loop-vectorizer

SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI'00]
- State-of-the-art straight-line code vectorizer
- Implemented in most compilers (including GCC and LLVM)
- In theory it should be a superset of loop-vectorizer
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed

SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI'00]
- State-of-the-art straight-line code vectorizer
- Implemented in most compilers (including GCC and LLVM)
- In theory it should be a superset of loop-vectorizer
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed
- In practice it is missing features present in the Loop vectorizer (Interleaved Loads, Predication)

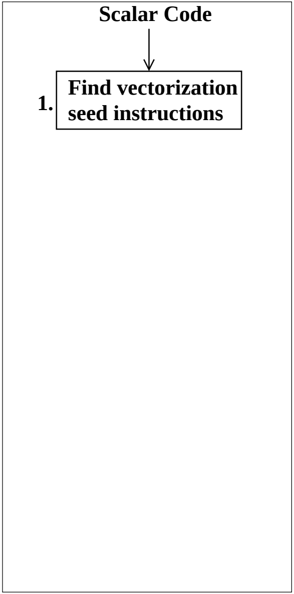
SLP Vectorization Algorithm

- Input is scalar IR

Scalar Code

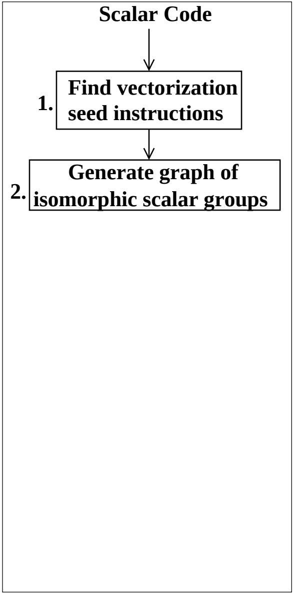
SLP Vectorization Algorithm

- Input is scalar IR
- Seed instructions are:
  1. Consecutive Stores
  2. Reductions



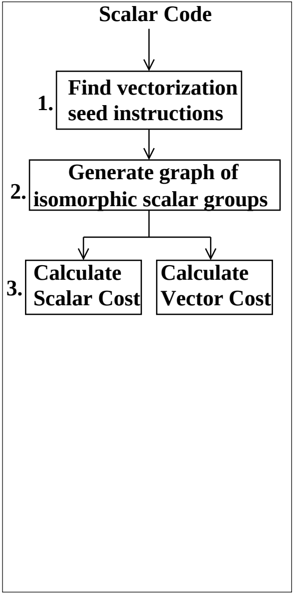
SLP Vectorization Algorithm

- Input is scalar IR
- Seed instructions are:
  1. Consecutive Stores
  2. Reductions
- Graph contains vectorizable isomorphic instructions



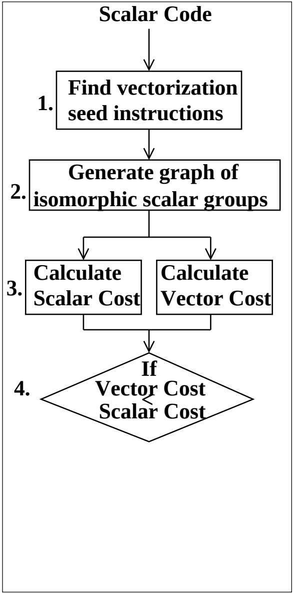
SLP Vectorization Algorithm

- Input is scalar IR
- Seed instructions are:
  1. Consecutive Stores
  2. Reductions
- Graph contains vectorizable isomorphic instructions
- Cost: weighted instr. count



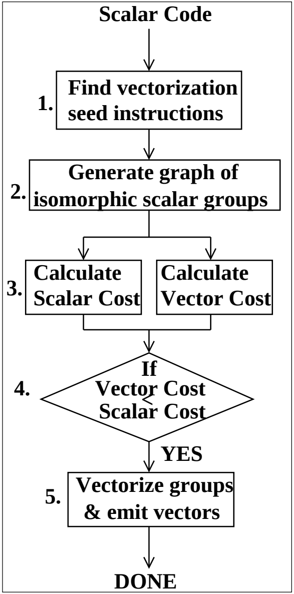
SLP Vectorization Algorithm

- Input is scalar IR
- Seed instructions are:
  1. Consecutive Stores
  2. Reductions
- Graph contains vectorizable isomorphic instructions
- Cost: weighted instr. count
- Check vectorization profitability



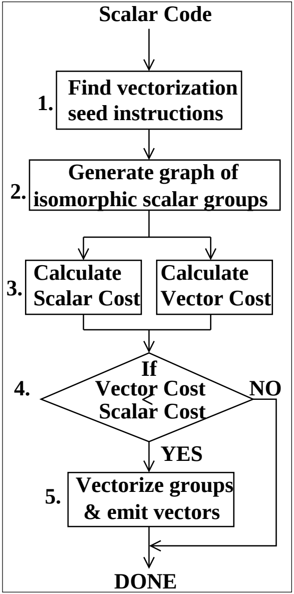
SLP Vectorization Algorithm

- Input is scalar IR
- Seed instructions are:
  1. Consecutive Stores
  2. Reductions
- Graph contains vectorizable isomorphic instructions
- Cost: weighted instr. count
- Check vectorization profitability
- Emit vectors only if profitable



SLP Vectorization Algorithm

- Input is scalar IR
- Seed instructions are:
  1. Consecutive Stores
  2. Reductions
- Graph contains vectorizable isomorphic instructions
- Cost: weighted instr. count
- Check vectorization profitability
- Emit vectors only if profitable



When SLP Fails

1. Data Dependencies

When SLP Fails

1. Data Dependencies
2. Too many gather/scatter instructions. Costs outweigh benefits.

Original	Vectorized
ADD1	Insert1
ADD2	Insert2
ADD3	Insert3
ADD4	Insert4
	ADD1 | ADD2 | ADD3 | ADD4
	Extract1
	Extract2
	Extract3
	Extract4

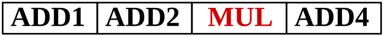
When SLP Fails

1. Data Dependencies

2. Too many gather/scatter instructions. Costs outweigh benefits.

Original	Vectorized
ADD1	Insert1
ADD2	Insert2
ADD3	Insert3
ADD4	Insert4
	ADD1 | ADD2 | ADD3 | ADD4
	Extract1
	Extract2
	Extract3
	Extract4

3. Non-isomorphism



SLP Fails due to non-isomorphism

```
      ...
B[i]   = A[i]   * 7.0 + 1.0;
B[i+1]= A[i+1]        + 5.0;
      ...
```

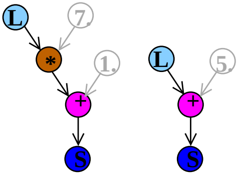
a. Input C code

(X) **Instruction Node or Constant**	→ **Data Flow Edge**

SLP Fails due to non-isomorphism

```
        ...
B[i]   = A[i]   * 7.0 + 1.0;
B[i+1] = A[i+1]       + 5.0;
        ...
```

a. Input C code



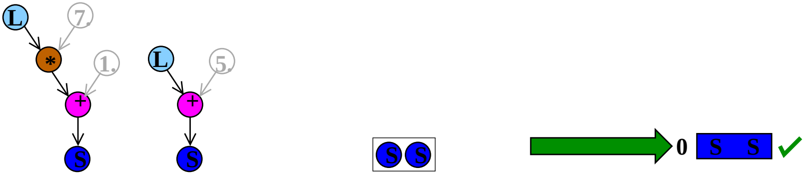
b. DFG

(X) **Instruction Node or Constant**	→ **Data Flow Edge**

SLP Fails due to non-isomorphism

```
    ...
B[i]   = A[i]   * 7.0 + 1.0;
B[i+1] = A[i+1]       + 5.0;
    ...
```

a. Input C code



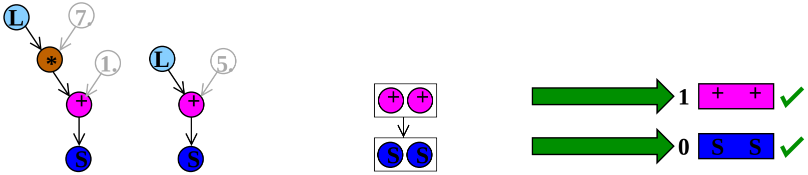
b. DFG **c. SLP internal graph** **d. SLP vectorized groups**

(X) **Instruction Node or Constant**	→ **Data Flow Edge**

SLP Fails due to non-isomorphism

```
      ...
B[i]   = A[i]   * 7.0 + 1.0;
B[i+1] = A[i+1]       + 5.0;
      ...
```

a. Input C code



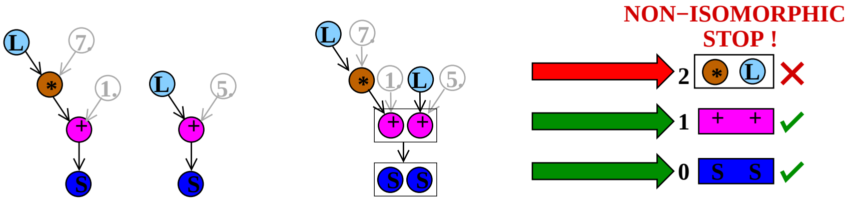
b. DFG **c. SLP internal graph** **d. SLP vectorized groups**

X Instruction Node or Constant → Data Flow Edge

SLP Fails due to non-isomorphism

```
          ...
B[i]   = A[i]   * 7.0 + 1.0;
B[i+1] = A[i+1]       + 5.0;
          ...
```

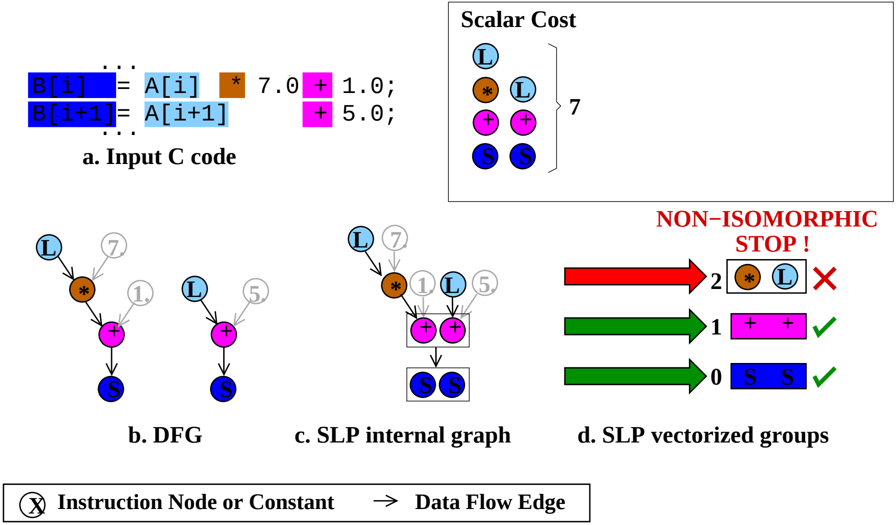
a. Input C code



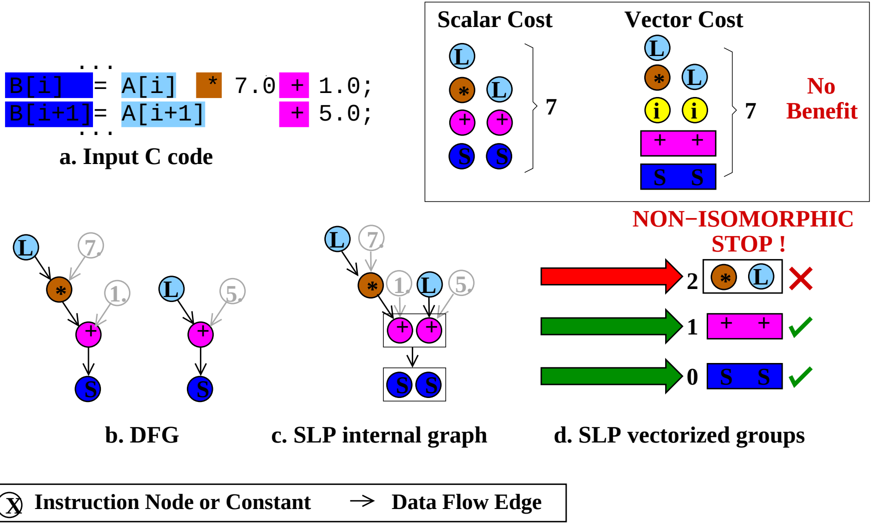
b. DFG **c. SLP internal graph** **d. SLP vectorized groups**

X **Instruction Node or Constant** → **Data Flow Edge**

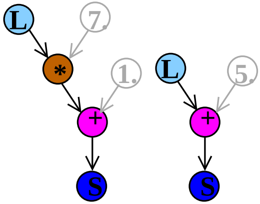
SLP Fails due to non-isomorphism



SLP Fails due to non-isomorphism

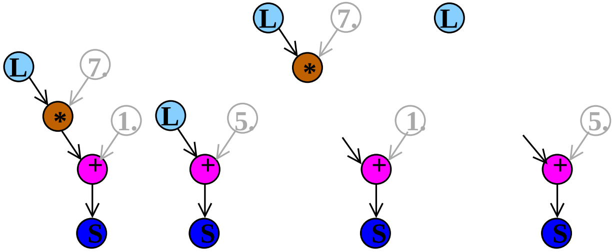


PSLP fixes Non-Isomorphism



a. PSLP graphs

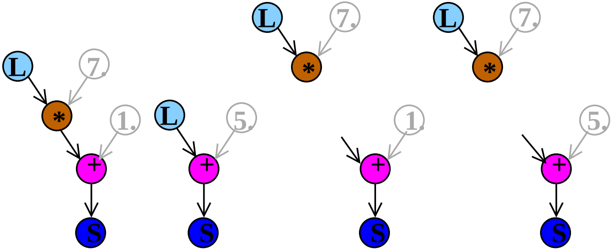
PSLP fixes Non-Isomorphism



a. PSLP graphs

b. PSLP padded graphs

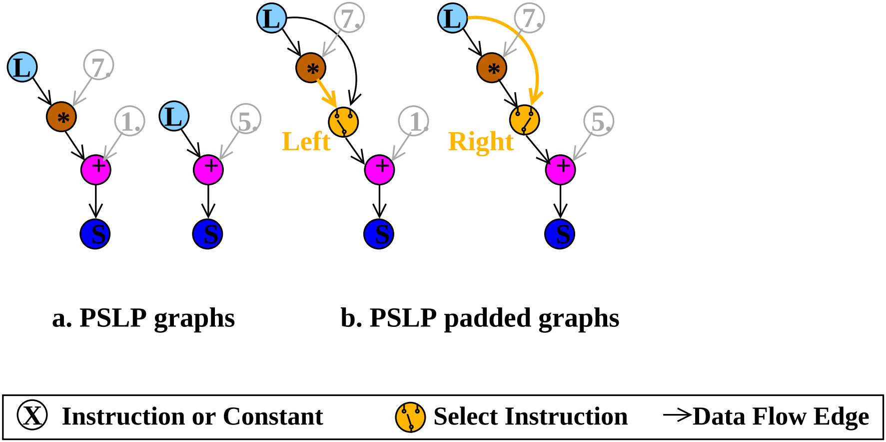
PSLP fixes Non-Isomorphism



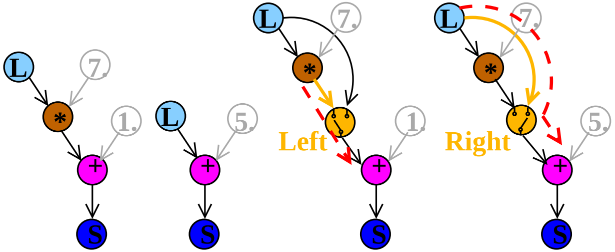
a. PSLP graphs **b. PSLP padded graphs**

Ⓧ **Instruction or Constant**	→ **Data Flow Edge**

PSLP fixes Non-Isomorphism



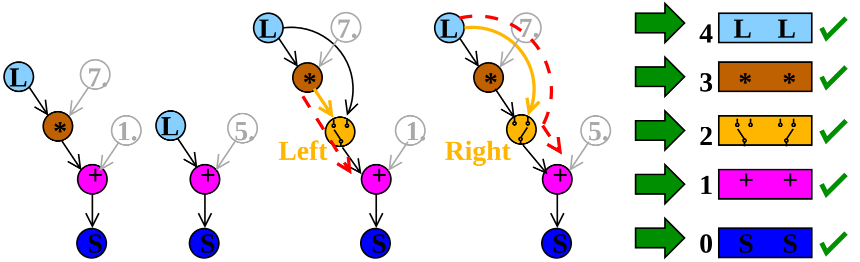
PSLP fixes Non-Isomorphism



a. PSLP graphs

b. PSLP padded graphs

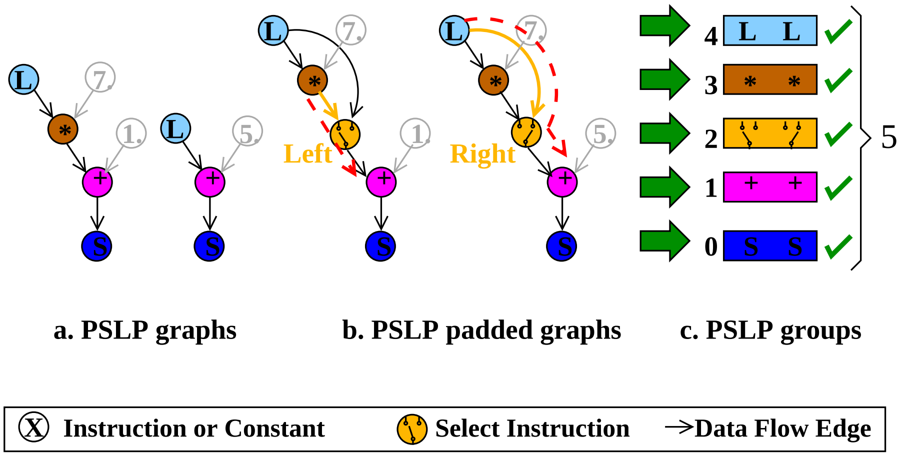
PSLP fixes Non-Isomorphism



a. PSLP graphs b. PSLP padded graphs c. PSLP groups

X Instruction or Constant Select Instruction → Data Flow Edge

PSLP fixes Non-Isomorphism



PSLP Algorithm

- Extension to SLP

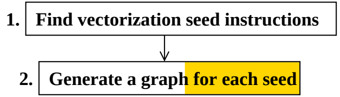
PSLP Algorithm

- Extension to SLP

1. **Find vectorization seed instructions**

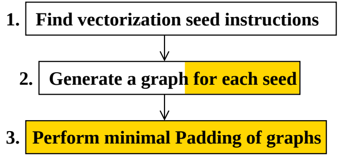
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)



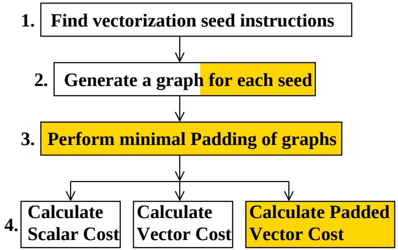
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding



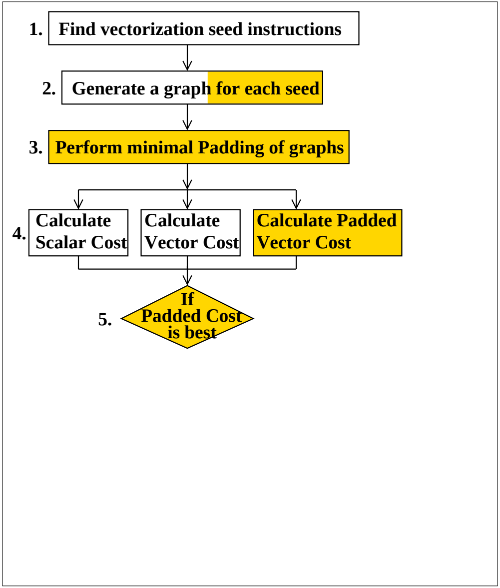
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding
- Cost estimation



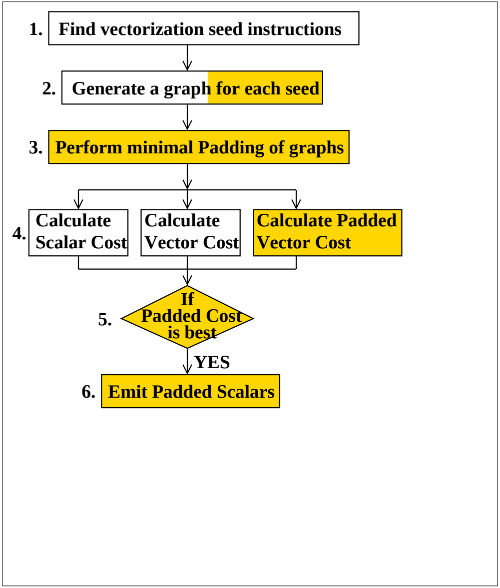
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding
- Cost estimation



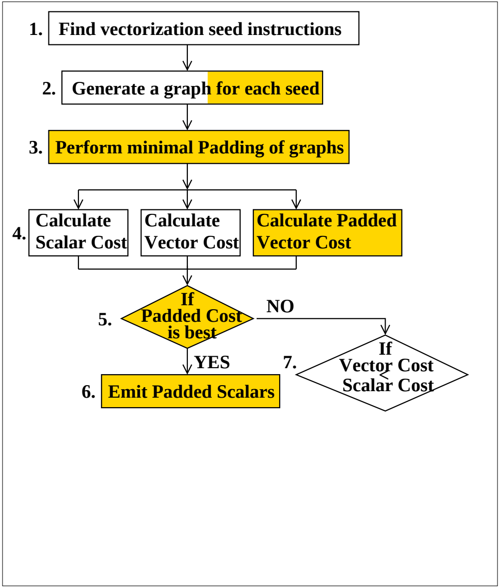
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding
- Cost estimation
- Emit redundant code to create isomorphism



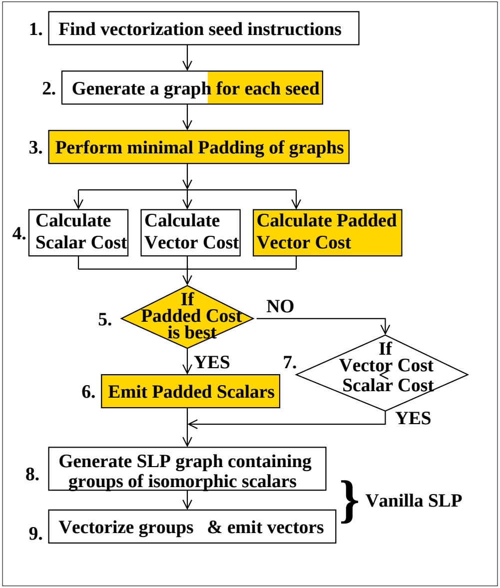
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding
- Cost estimation
- Emit redundant code to create isomorphism



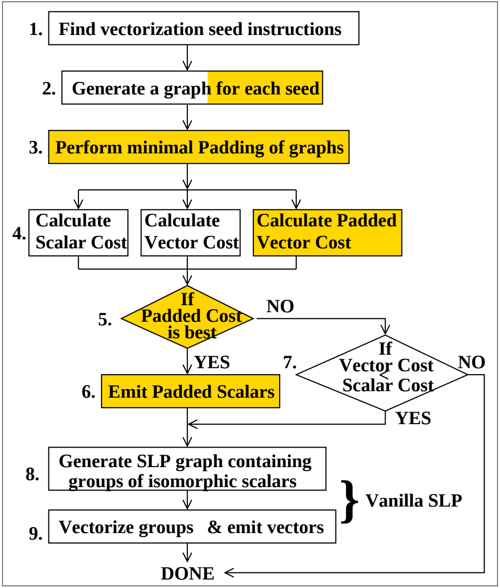
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding
- Cost estimation
- Emit redundant code to create isomorphism
- Code vectorized by original SLP

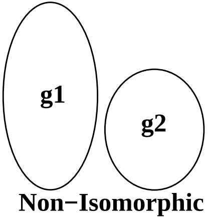


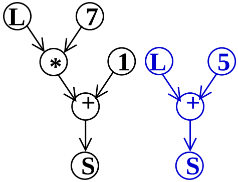
PSLP Algorithm

- Extension to SLP
- Generate multiple graphs (unlike SLP)
- Minimal Padding
- Cost estimation
- Emit redundant code to create isomorphism
- Code vectorized by original SLP

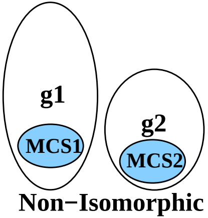


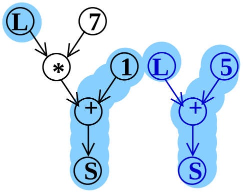
Minimal Padding Algorithm



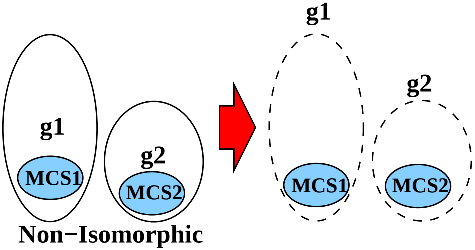


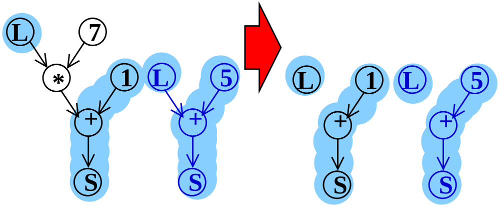
Minimal Padding Algorithm



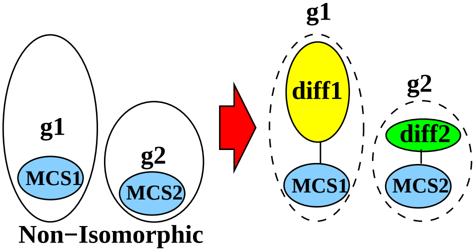


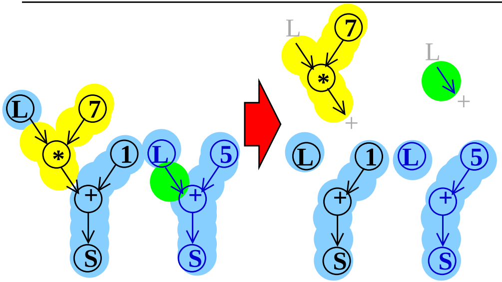
Minimal Padding Algorithm



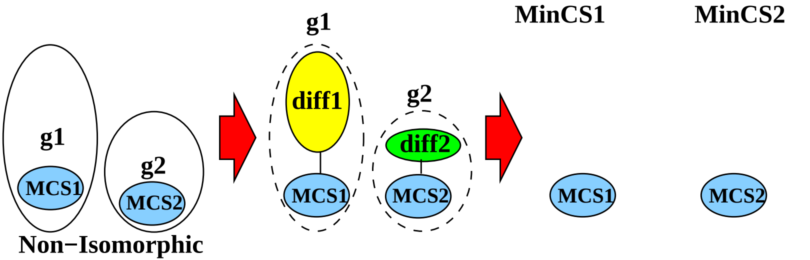


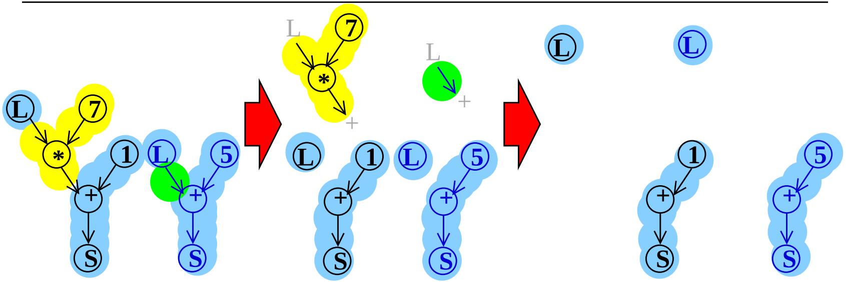
Minimal Padding Algorithm



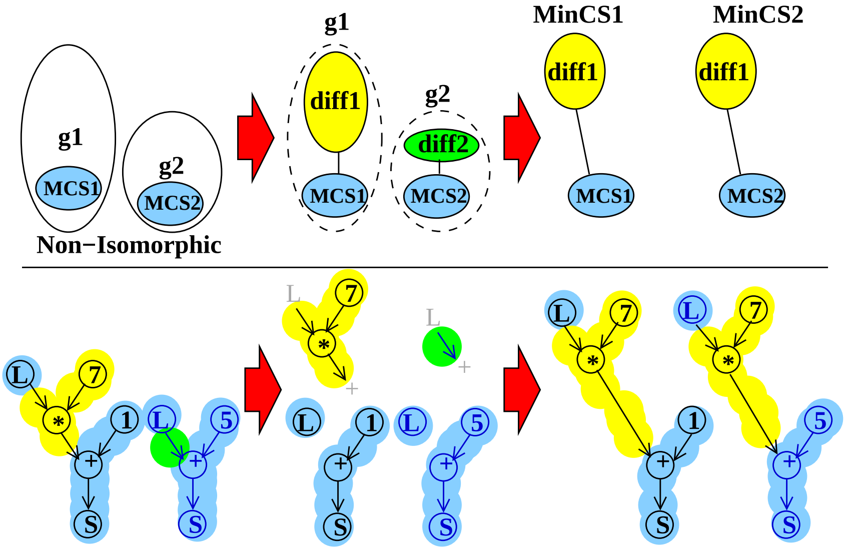


Minimal Padding Algorithm

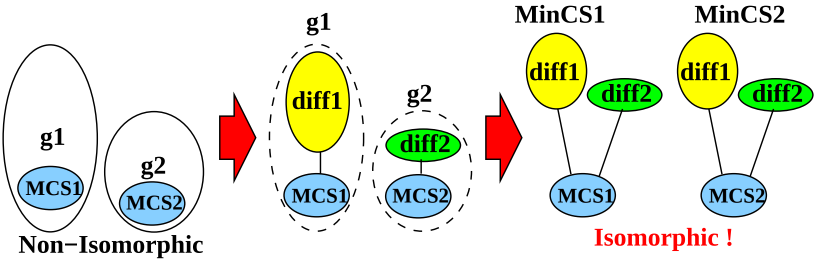


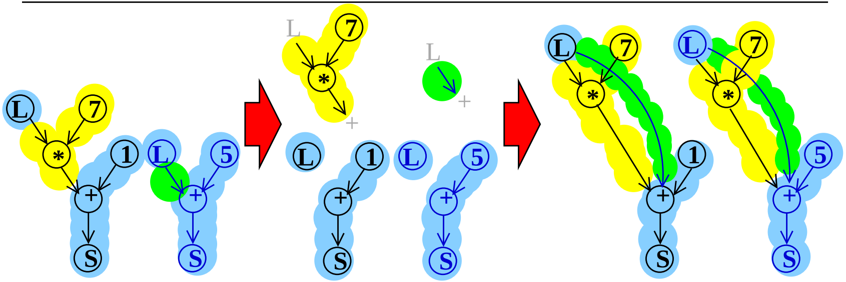


Minimal Padding Algorithm

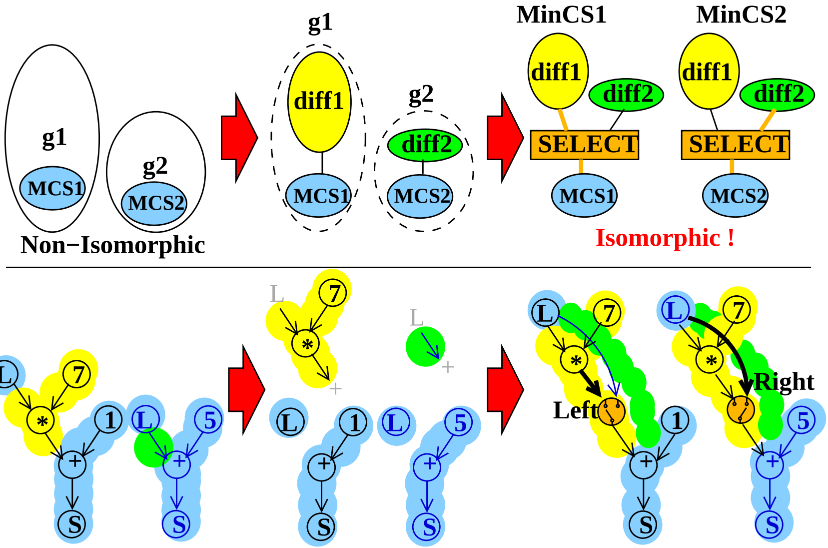


Minimal Padding Algorithm

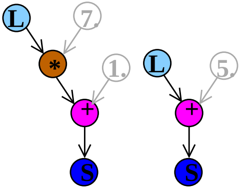




Minimal Padding Algorithm

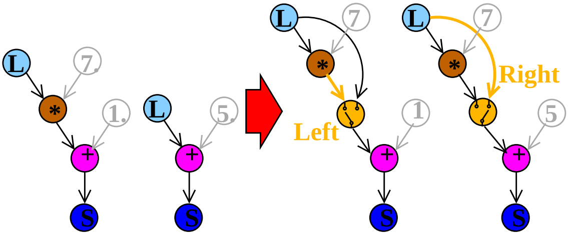


We can do better: Remove redundant Selects



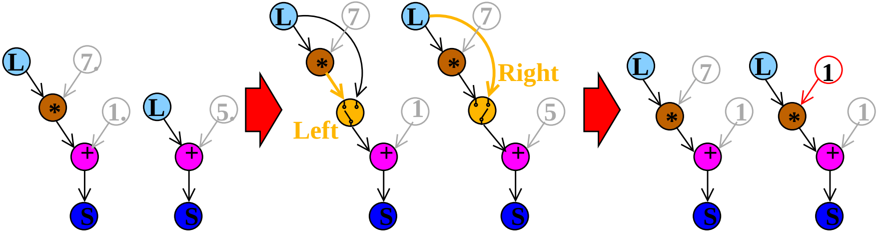
EXAMPLE: Instruction acting as Select

We can do better: Remove redundant Selects



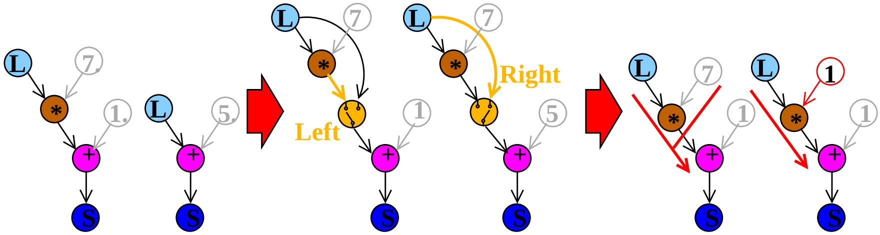
EXAMPLE: Instruction acting as Select

We can do better: Remove redundant Selects



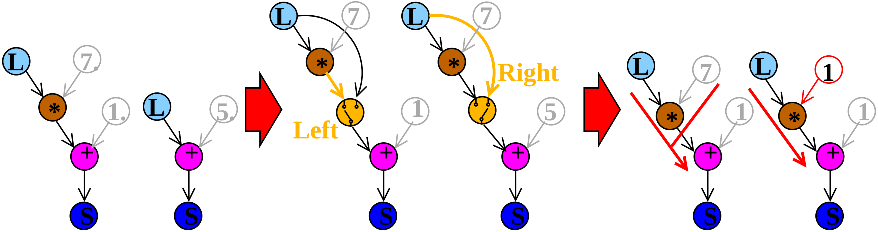
EXAMPLE: Instruction acting as Select

We can do better: Remove redundant Selects

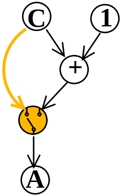


EXAMPLE: Instruction acting as Select

We can do better: Remove redundant Selects

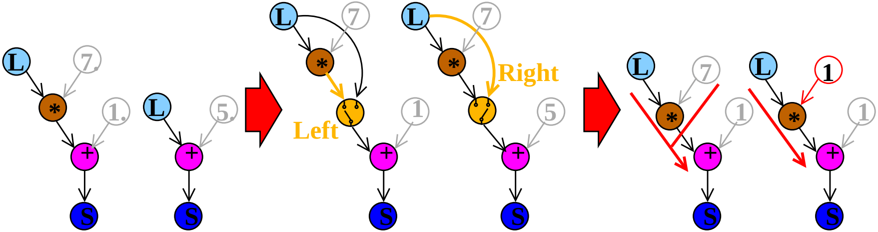


EXAMPLE: Instruction acting as Select

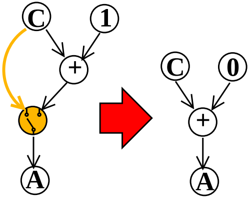


a. Instruction acting as Select

We can do better: Remove redundant Selects

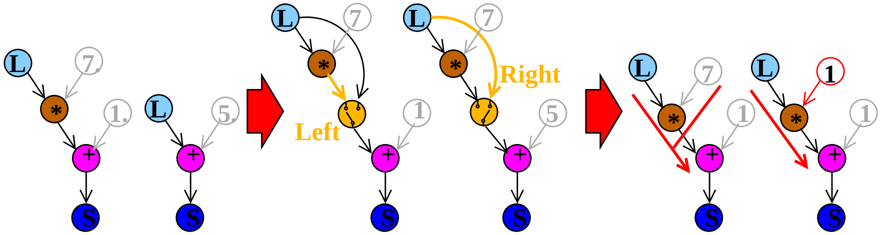


EXAMPLE: Instruction acting as Select

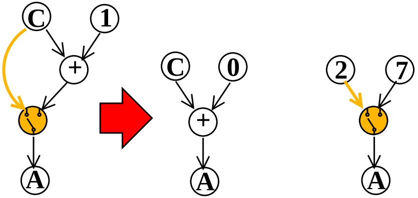


a. Instruction acting as Select

We can do better: Remove redundant Selects



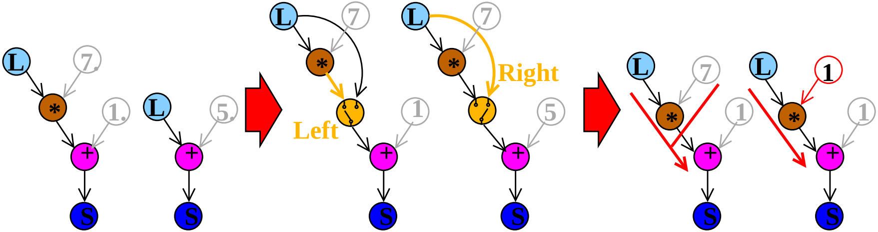
EXAMPLE: Instruction acting as Select



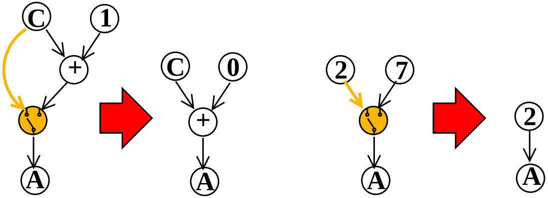
a. Instruction acting as Select

b. Select constants

We can do better: Remove redundant Selects

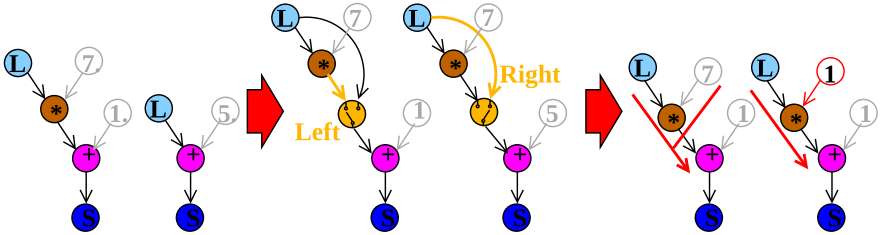


EXAMPLE: Instruction acting as Select

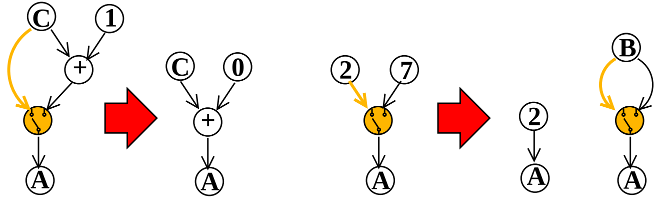


a. Instruction acting as Select **b. Select constants**

We can do better: Remove redundant Selects

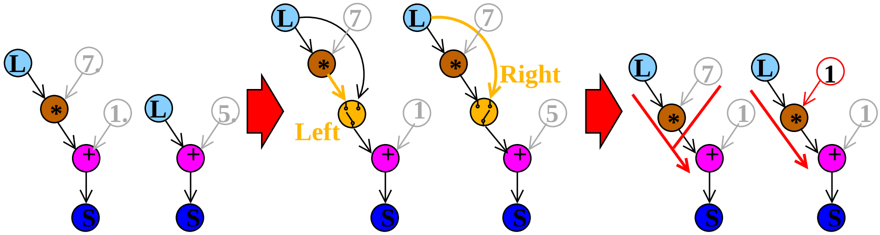


EXAMPLE: Instruction acting as Select

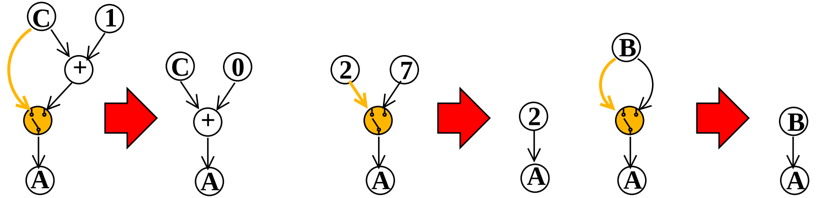


a. Instruction acting as Select b. Select constants c. Select same node

We can do better: Remove redundant Selects



EXAMPLE: Instruction acting as Select



a. Instruction acting as Select **b. Select constants** **c. Select same node**

Opportunities for PSLP in real-life applications

1. Non-isomorphic source code (e.g. computing conjugates in 433.milc)

```
b[0].real =   a[0].real
b[0].imag = - a[0].imag
b[1].real =   a[1].real
b[1].imag = - a[1].imag
```

Memory
...
a[0].real
a[0].imag
a[1].real
a[1].imag
...

Opportunities for PSLP in real-life applications

1. Non-isomorphic source code (e.g. computing conjugates in 433.milc)

```
b[0].real =   a[0].real
b[0].imag = - a[0].imag
b[1].real =   a[1].real
b[1].imag = - a[1].imag
```

Memory
...
a[0].real
a[0].imag
a[1].real
a[1].imag
...

Opportunities for PSLP in real-life applications

1. Non-isomorphic source code (e.g. computing conjugates in 433.milc)

```
b[0].real =   a[0].real
b[0].imag = - a[0].imag
b[1].real =   a[1].real
b[1].imag = - a[1].imag
```

Memory
...
a[0].real
a[0].imag
a[1].real
a[1].imag
...

2. Isomorphic source code but non-isomorphic IR due to high-level optimizations (jdct of cjpeg)

```
tmp1 = quantval[0]*16384
tmp2 = quantval[1]*22725
tmp3 = quantval[2]*21407
tmp4 = quantval[3]*19266
```

Opportunities for PSLP in real-life applications

1. Non-isomorphic source code (e.g. computing conjugates in 433.milc)

```
b[0].real =   a[0].real
b[0].imag = - a[0].imag
b[1].real =   a[1].real
b[1].imag = - a[1].imag
```

Memory
...
a[0].real
a[0].imag
a[1].real
a[1].imag
...

2. Isomorphic source code but non-isomorphic IR due to high-level optimizations (jdct of cjpeg)

```
tmp1 = quantval[0]*16384          tmp1 = quantval[0]<<14
tmp2 = quantval[1]*22725   opt    tmp2 = quantval[1]*22725
tmp3 = quantval[2]*21407   ==>    tmp3 = quantval[2]*21407
tmp4 = quantval[3]*19266          tmp4 = quantval[3]*19266
```

Opportunities for PSLP in real-life applications

1. Non-isomorphic source code (e.g. computing conjugates in 433.milc)

```
b[0].real =   a[0].real
b[0].imag = - a[0].imag
b[1].real =   a[1].real
b[1].imag = - a[1].imag
```

Memory
...
a[0].real
a[0].imag
a[1].real
a[1].imag
...

2. Isomorphic source code but non-isomorphic IR due to high-level optimizations (jdct of cjpeg)

```
tmp1 = quantval[0]*16384
tmp2 = quantval[1]*22725
tmp3 = quantval[2]*21407
tmp4 = quantval[3]*19266
```

opt ⇒

```
tmp1 = quantval[0]<<14
tmp2 = quantval[1]*22725
tmp3 = quantval[2]*21407
tmp4 = quantval[3]*19266
```

Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.

Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.
- Target: Intel Core i5-4570 @ 3.2Ghz

Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.
- Target: Intel Core i5-4570 @ 3.2Ghz
- Compiler flags: -O3 -allow-partial-unroll -march=core-avx2 -mtune-core-i7 -ffast-math

Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.
- Target: Intel Core i5-4570 @ 3.2Ghz
- Compiler flags: -O3 -allow-partial-unroll -march=core-avx2 -mtune-core-i7 -ffast-math
- Kernels, SPEC 2006 and Mediabench II
- We evaluated the following cases:

Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.
- Target: Intel Core i5-4570 @ 3.2Ghz
- Compiler flags: -O3 -allow-partial-unroll -march=core-avx2 -mtune-core-i7 -ffast-math
- Kernels, SPEC 2006 and Mediabench II
- We evaluated the following cases:
  1. All loop, SLP and PSLP vectorizers disabled (O3)

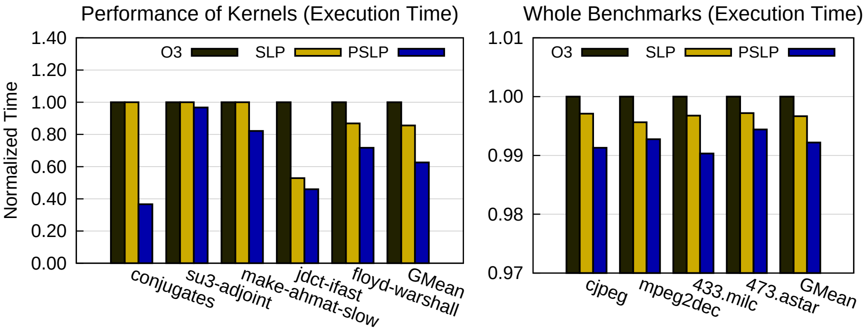
Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.
- Target: Intel Core i5-4570 @ 3.2Ghz
- Compiler flags: -O3 -allow-partial-unroll -march=core-avx2 -mtune-core-i7 -ffast-math
- Kernels, SPEC 2006 and Mediabench II
- We evaluated the following cases:
  1. All loop, SLP and PSLP vectorizers disabled (O3)
  2. O3 + SLP enabled (SLP)

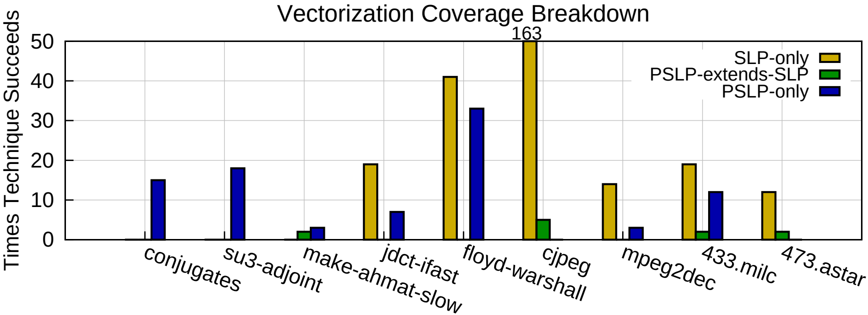
Experimental Setup

- Implemented PSLP in the trunk version of the LLVM 3.6 compiler.
- Target: Intel Core i5-4570 @ 3.2Ghz
- Compiler flags: -O3 -allow-partial-unroll -march=core-avx2 -mtune-core-i7 -ffast-math
- Kernels, SPEC 2006 and Mediabench II
- We evaluated the following cases:
  1. All loop, SLP and PSLP vectorizers disabled (O3)
  2. O3 + SLP enabled (SLP)
  3. O3 + PSLP enabled (PSLP)

PSLP increases performance

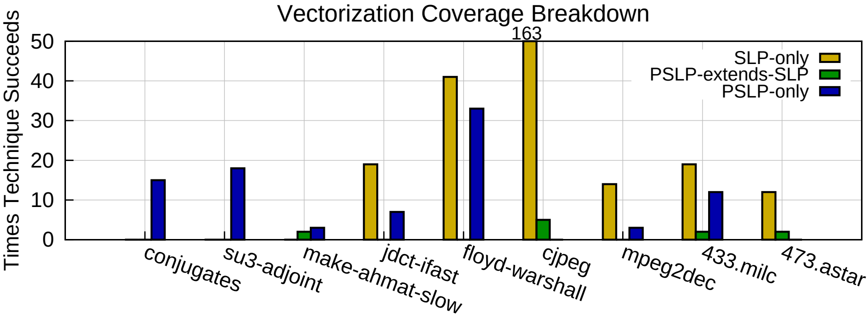


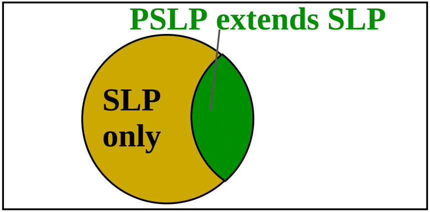
PSLP enables or extends vectorization



- SLP is adequate

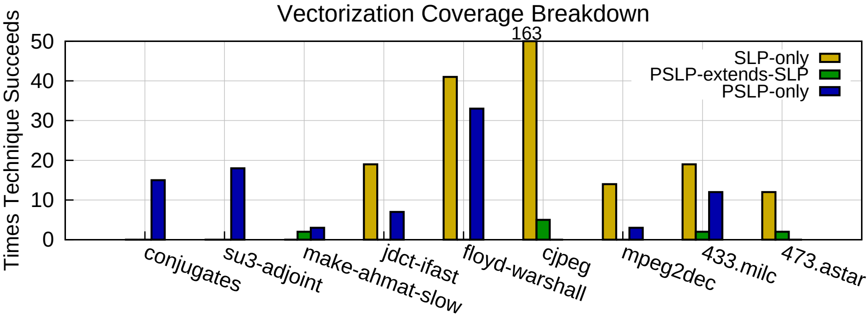
PSLP enables or extends vectorization

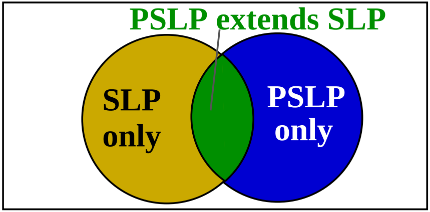




- SLP is adequate
- SLP stops at non-isomorphic code. PSLP extends it.

PSLP enables or extends vectorization

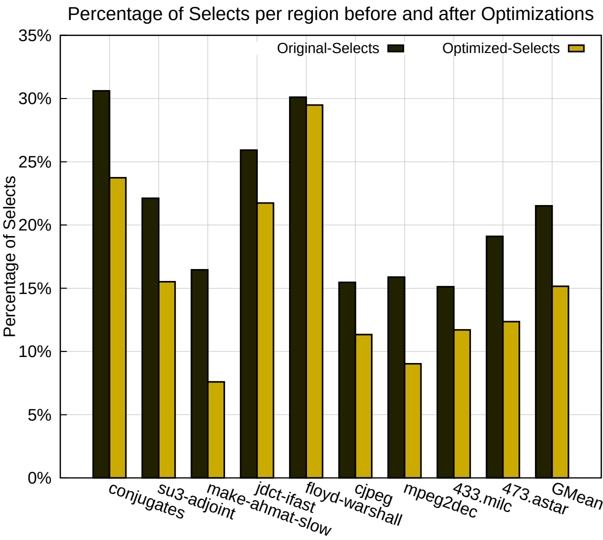




- SLP is adequate
- SLP stops at non-isomorphic code. PSLP extends it.
- SLP fails completely. PSLP succeeds.

Optimizing away redundant Selects

- Select-removal optimizations remove about 21% of the Selects



Conclusion

- PSLP improves vectorization coverage compared to the state-of-the-art

Conclusion

- PSLP improves vectorization coverage compared to the state-of-the-art
- Converts non-isomorphic code into isomorphic by:

Conclusion

- PSLP improves vectorization coverage compared to the state-of-the-art
- Converts non-isomorphic code into isomorphic by:
  - Relying on the Min Common Supergraph for minimal injection of redundant code

Conclusion

- PSLP improves vectorization coverage compared to the state-of-the-art
- Converts non-isomorphic code into isomorphic by:
  - Relying on the Min Common Supergraph for minimal injection of redundant code
  - Emitting Select instructions to guarantee correctness

Conclusion

- PSLP improves vectorization coverage compared to the state-of-the-art
- Converts non-isomorphic code into isomorphic by:
  - Relying on the Min Common Supergraph for minimal injection of redundant code
  - Emitting Select instructions to guarantee correctness
  - Optimizing away redundant Selects

Conclusion

- PSLP improves vectorization coverage compared to the state-of-the-art
- Converts non-isomorphic code into isomorphic by:
  - Relying on the Min Common Supergraph for minimal injection of redundant code
  - Emitting Select instructions to guarantee correctness
  - Optimizing away redundant Selects
- PSLP performs better compared to SLP on commodity SIMD-capable hardware