PSLP: Padded SLP
Automatic Vectorization

Vasileios Porpodas†, AlbertoMagni‡
and Timothy M. Jones†

University of Cambridge†
University of Edinburgh‡

CGO 2015
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
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

Scalar Code

1. Find vectorization seed instructions
2. Generate graph of isomorphic scalar groups
3. Calculate Scalar Cost  Calculate Vector Cost
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

1. Find vectorization seed instructions
2. Generate graph of isomorphic scalar groups
3. Calculate Scalar Cost Calculate Vector Cost
4. If Vector Cost < Scalar Cost
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

1. Find vectorization seed instructions
2. Generate graph of isomorphic scalar groups
3. Calculate Scalar Cost  Calculate Vector Cost
4. If Vector Cost < Scalar Cost
   YES
5. Vectorize groups & emit vectors
   DONE
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

```
1. Find vectorization seed instructions
2. Generate graph of isomorphic scalar groups
3. Calculate Scalar Cost
   Calculate Vector Cost
4. If Vector Cost < Scalar Cost, Vectorize groups & emit vectors
5. DONE
```
SLP Fails due to non-isomorphism

\[
\begin{align*}
\ldots \\
B[i] &= A[i] * 7.0 + 1.0; \\
B[i+1] &= A[i+1] + 5.0; \\
\ldots \\
\text{a. Input C code}
\end{align*}
\]
SLP Fails due to non-isomorphism

a. Input C code

\[
\begin{align*}
B[i] &= A[i] \times 7.0 + 1.0; \\
B[i+1] &= A[i+1] + 5.0;
\end{align*}
\]

b. DFG

Instruction Node or Constant \rightarrow Data Flow Edge
SLP Fails due to non-isomorphism

\[ B[i] = A[i] \times 7.0 + 1.0; \]
\[ B[i+1] = A[i+1] + 5.0; \]

a. Input C code

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

\( \times \) Instruction Node or Constant \( \rightarrow \) Data Flow Edge

www.cl.cam.ac.uk/~vp331/
SLP Fails due to non-isomorphism

\[
\begin{align*}
B[i] &= A[i] \times 7.0 + 1.0; \\
B[i+1] &= A[i+1] + 5.0;
\end{align*}
\]

a. Input C code

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

Instruction Node or Constant $\rightarrow$ Data Flow Edge
SLP Fails due to non-isomorphism

\[
\begin{align*}
B[i] &= A[i] \times 7.0 + 1.0; \\
B[i+1] &= A[i+1] + 5.0;
\end{align*}
\]

a. Input C code

b. DFG

c. SLP internal graph

d. SLP vectorized groups

× Instruction Node or Constant → Data Flow Edge
SLP Fails due to non-isomorphism

a. Input C code

\[
\begin{align*}
B[i] &= A[i] \times 7.0 + 1.0; \\
B[i+1] &= A[i+1] + 5.0;
\end{align*}
\]

b. DFG

c. SLP internal graph

d. SLP vectorized groups

Scalar Cost

\[
\begin{align*}
&L \\
&* L \\
&+ + S S
\end{align*}
\]

\[
7
\]

NON-ISOMORPHIC STOP!
SLP Fails due to non-isomorphism

a. Input C code

\[ B[i] = A[i] \times 7.0 + 1.0; \]
\[ B[i+1] = A[i+1] + 5.0; \]

b. DFG

c. SLP internal graph

d. SLP vectorized groups

Scalar Cost

\[ L \quad \times \quad L \quad + \quad + \quad S \quad S \]

Vector Cost

\[ L \quad \times \quad i \quad + \quad + \quad S \quad S \]

No Benefit

NON-ISOMORPHIC STOP!

\[ 2 \quad \times \quad L \]

\[ 1 \quad + \quad + \]

\[ 0 \quad S \quad S \]

\( \text{Instruction Node or Constant} \rightarrow \text{Data Flow Edge} \)
PSLP fixes Non-Isomorphism

a. PSLP graphs

Instruction or Constant → Data Flow Edge
PSLP fixes Non-Isomorphism

a. PSLP graphs  
b. PSLP padded graphs

Instruction or Constant → Data Flow Edge
PSLP fixes Non-Isomorphism

a. PSLP graphs

b. PSLP padded graphs

Instruction or Constant → Data Flow Edge
PSLP fixes Non-Isomorphism

a. PSLP graphs

b. PSLP padded graphs

Instruction or Constant Select Instruction → Data Flow Edge
PSLP fixes Non-Isomorphism

a. PSLP graphs

b. PSLP padded graphs

Instruction or Constant  Select Instruction  →Data Flow Edge
PSLP fixes Non-Isomorphism

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

Instruction or Constant  Select Instruction  → Data Flow Edge
PSLP fixes Non-Isomorphism

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

Instruction or Constant Select Instruction → Data Flow Edge
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)

1. Find vectorization seed instructions
2. Generate a graph for each seed
PSLP Algorithm

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

1. Find vectorization seed instructions
2. Generate a graph for each seed
3. Perform minimal Padding of graphs
PSLP Algorithm

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

1. Find vectorization seed instructions
2. Generate a graph for each seed
3. Perform minimal Padding of graphs
4. Calculate Scalar Cost  Calculate Vector Cost  Calculate Padded Vector Cost
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

1. Find vectorization seed instructions
2. Generate a graph for each seed
3. Perform minimal Padding of graphs
4. Calculate Scalar Cost  
   Calculate Vector Cost  
   Calculate Padded Vector Cost
5. If Padded Cost is best
   YES
6. Emit Padded Scalars
PSLP Algorithm

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

1. Find vectorization seed instructions
2. Generate a graph for each seed
3. Perform minimal Padding of graphs
4. Calculate Scalar Cost  Calculate Vector Cost  Calculate Padded Vector Cost
5. If Padded Cost is best
   - NO
   - YES
6. Emit Padded Scalars
7. If Vector Cost
   - Scalar Cost
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

1. Find vectorization seed instructions
2. Generate a graph for each seed
3. Perform minimal Padding of graphs
4. Calculate Scalar Cost, Calculate Vector Cost, Calculate Padded Vector Cost
5. If Padded Cost is best
   - YES: Emit Padded Scalars
   - NO: Go back to step 4
6. Emit Padded Scalars
7. If Vector Cost Scalar Cost
   - YES: Generate SLP graph containing groups of isomorphic scalars
   - NO: Go back to step 4
8. Generate SLP graph containing groups of isomorphic scalars
9. Vectorize groups & emit vectors

Vanilla 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

Non-Isomorphic

Diagram of two graphs with labels $g_1$ and $g_2$.
Minimal Padding Algorithm

Non-Isomorphic

\[ g_1 \quad \text{MCS1} \]
\[ g_2 \quad \text{MCS2} \]
Minimal Padding Algorithm

Non-Isomorphic

MCS1
MCS2

MCS1
MCS2
Minimal Padding Algorithm

Non-Isomorphic

\[ g_1 \quad \text{MCS1} \]
\[ g_2 \quad \text{MCS2} \]

\[ \text{diff1} \]
\[ \text{diff2} \]

\[ g_1 \quad \text{MCS1} \]
\[ g_2 \quad \text{MCS2} \]

\[ \text{MCS1} \]
\[ \text{MCS2} \]

\[ \text{slide 7 of 15} \]

www.cl.cam.ac.uk/~vp331/
Minimal Padding Algorithm

Non-Isomorphic

MinCS1

MinCS2
Minimal Padding Algorithm

Non-Isomorphic

$g_1$ $g_2$

MCS1 MCS2

MinCS1 MinCS2

$\text{diff1}$ $\text{diff2}$

$S$ $L$

$+\quad *\quad 7$

$\text{MCS1}$ $\text{MCS2}$

$\text{MCS1}$ $\text{MCS2}$

$\text{slide 7 of 15}$
Minimal Padding Algorithm

Non-Isomorphic

Isomorphic!
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

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)

\[
\begin{align*}
    b[0].real &= a[0].real \\
    b[0].imag &= -a[0].imag \\
    b[1].real &= a[1].real \\
    b[1].imag &= -a[1].imag
\end{align*}
\]

<table>
<thead>
<tr>
<th>Memory</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0].real</td>
<td>a[0].imag</td>
</tr>
<tr>
<td>a[1].real</td>
<td>a[1].imag</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Opportunities for PSLP in real-life applications

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

\[
\begin{align*}
\text{b}[0].\text{real} &= \text{a}[0].\text{real} \\
\text{b}[0].\text{imag} &= -\text{a}[0].\text{imag} \\
\text{b}[1].\text{real} &= \text{a}[1].\text{real} \\
\text{b}[1].\text{imag} &= -\text{a}[1].\text{imag}
\end{align*}
\]

<table>
<thead>
<tr>
<th>...</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0].real</td>
<td>...</td>
</tr>
<tr>
<td>a[0].imag</td>
<td>...</td>
</tr>
<tr>
<td>a[1].real</td>
<td>...</td>
</tr>
<tr>
<td>a[1].imag</td>
<td>...</td>
</tr>
</tbody>
</table>
Opportunities for PSLP in real-life applications

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

   \[
   \begin{align*}
   b[0].real &= a[0].real \\
   b[0].imag &= a[0].imag \\
   b[1].real &= a[1].real \\
   b[1].imag &= a[1].imag
   \end{align*}
   \]

   ...  

   Memory

   \[
   \begin{array}{l}
   a[0].real \\
   a[0].imag \\
   a[1].real \\
   a[1].imag \\
   \ldots
   \end{array}
   \]

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

   \[
   \begin{align*}
   tmp1 &= \text{quantval}[0] \times 16384 \\
   tmp2 &= \text{quantval}[1] \times 22725 \\
   tmp3 &= \text{quantval}[2] \times 21407 \\
   tmp4 &= \text{quantval}[3] \times 19266
   \end{align*}
   \]
Opportunities for PSLP in real-life applications

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

\[
\begin{align*}
  b[0].real &= a[0].real \\
  b[0].imag &= -a[0].imag \\
  b[1].real &= a[1].real \\
  b[1].imag &= -a[1].imag
\end{align*}
\]

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0].real</td>
</tr>
<tr>
<td>a[0].imag</td>
</tr>
<tr>
<td>a[1].real</td>
</tr>
<tr>
<td>a[1].imag</td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
  \text{tmp1} &= \text{quantval}[0]*16384 \\
  \text{tmp2} &= \text{quantval}[1]*22725 \\
  \text{tmp3} &= \text{quantval}[2]*21407 \\
  \text{tmp4} &= \text{quantval}[3]*19266
\end{align*}
\]

\[
\begin{align*}
  \text{tmp1} &= \text{quantval}[0]<<14 \\
  \text{tmp2} &= \text{quantval}[1]*22725 \\
  \text{tmp3} &= \text{quantval}[2]*21407 \\
  \text{tmp4} &= \text{quantval}[3]*19266
\end{align*}
\]
Oppunities for PSLP in real-life applications

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

\[
\begin{align*}
b[0].real &= a[0].real \\
b[0].imag &= -a[0].imag \\
b[1].real &= a[1].real \\
b[1].imag &= -a[1].imag
\end{align*}
\]

<table>
<thead>
<tr>
<th>...</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0].real</td>
<td></td>
</tr>
<tr>
<td>a[0].imag</td>
<td></td>
</tr>
<tr>
<td>a[1].real</td>
<td></td>
</tr>
<tr>
<td>a[1].imag</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
tmp1 &= quantval[0] \times 16384 \\
tmp2 &= quantval[1] \times 22725 \\
tmp3 &= quantval[2] \times 21407 \\
tmp4 &= quantval[3] \times 19266
\end{align*}
\]

```
opt
\[
\begin{align*}
tmp1 &= quantval[0] \ll 14 \\
tmp2 &= quantval[1] \times 22725 \\
tmp3 &= quantval[2] \times 21407 \\
tmp4 &= quantval[3] \times 19266
\end{align*}
```
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

Performance of Kernels (Execution Time)

Whole Benchmarks (Execution Time)
PSLP enables or extends vectorization

Vectorization Coverage Breakdown

- 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