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

Vasileios Porpodas¹, Rodrigo C. O. Rocha²
Evgueni Brevnov¹, Luís F. W. Góes³
and Timothy Mattson¹

Intel¹, University of Edinburgh², PUC Minas³

CGO 2019
Legal Disclaimer & Optimization Notice

Performance results are based on testing as of 10/16/2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright 2018, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

Optimization Notice

Intels compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI’00]
SLP: The Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen et al. PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP [Rosen et al. GCC-DEV’07]
SLP: The Straight-Line Code Vectorizer

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

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

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

- Vectorizes across instructions, *NOT* iterations
SLP compared to Loop Vectorization

- Vectorizes across instructions, NOT iterations

```c
for (i=0; i<N; i+=4)
    A[i] = B[i]
    A[i+1] = B[i+1]
    A[i+2] = B[i+2]
    A[i+3] = B[i+3]
```
SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

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

Loop Vectorization (LV) with VF = 4
for (i=0; i<N; i+=16)
SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

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

**Loop Vectorization (LV) with VF = 4**

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

http://vporpo.me
SLP compared to Loop Vectorization

- 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)
  \[
  \begin{align*}
  A[i] &= B[i] \\
  A[i+1] &= B[i+1] \\
  A[i+3] &= B[i+3]
  \end{align*}
  \]
SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

```
A[i] = B[i]
A[i+1] = B[i+1]
A[i+2] = B[i+2]
A[i+3] = B[i+3]
```

Loop Vectorization (LV) with VF = 4

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

for (i=0; i<N; i+=4)
    A[i] = B[i]
    A[i+1] = B[i+1]
    A[i+2] = B[i+2]
    A[i+3] = B[i+3]
SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

Loop Vectorization (LV) with VF = 4
for (i=0; i<N; i+=16)

- A[i, i+4, i+8, i+12] = B[i, i+4, i+8, i+12]
- A[i+1, i+5, i+9, i+13] = B[i+1, i+5, i+9, i+13]
- A[i+2, i+6, i+10, i+14] = B[i+2, i+6, i+10, i+14]
SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

**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 compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

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

**Loop Vectorization (LV) with VF = 4**

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

http://vporpo.me
SLP compared to Loop Vectorization

- 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]
  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+1] = B[i+1]
  A[i+2] = B[i+2]
  A[i+3] = B[i+3]
SLP compared to Loop Vectorization

- Vectorizes across instructions, *NOT* iterations

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

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

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

![Diagram of Look-Ahead SLP](http://vporpo.me)
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

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

\[
\begin{align*}
C[i+0] & \quad B[i+0] \\
& \quad \text{Lane 1} \\
\begin{array}{c}
+ \\
\& \\
S \\
A[i+0] \\
\end{array} & \quad A[i+0]E[i+0] \\
\begin{array}{c}
+ \\
\& \\
S \\
A[i+1] \\
\end{array} & \quad B[i+1]D[i+1] \\
D[i+0] & \quad C[i+1]E[i+1] \\
& \quad \text{Lane 2} \\
\begin{array}{c}
+ \\
\& \\
S \\
A[i+0] \\
\end{array} & \quad A[i+1] \\
\begin{array}{c}
+ \\
\& \\
S \\
A[i+1] \\
\end{array} & \quad A[i+1]E[i+1] \\
\end{align*}
\]
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

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

\[ A[i+0] \rightarrow B[i+0] \rightarrow C[i+0] \rightarrow D[i+0] \rightarrow E[i+0] \]

Lane 1:
- No Reordering

Lane 2:
- No Reordering

SLP:
- \( C[i:i+1] \rightarrow B[i:i+1] \rightarrow A[i:i+1] \)
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

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

```
C[i+0] B[i+0]

L  L  L  L  L
A[i+0] E[i+0]

D[i+0]

+ + +

& & &

S  S

A[i+0]  Lane 1

B[i+1]  D[i+1]

C[i+1]  E[i+1]

L  L  L  L  L

+ + +

& & &

S  S

A[i+1]  Lane 2
```

SLP

No Reordering

No Reordering

http://vporpo.me
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

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

\[ A[i+0] \] \[ B[i+0] \] \[ C[i+1] \] \[ D[i+1] \] \\
\[ L \] \[ L \] \[ L \] \[ L \] \\
\[ A[i+0] \] \[ E[i+0] \] \[ D[i+0] \] \\
\[ L \] \[ L \] \[ L \] \\
\[ \& \] \[ \& \] \[ \& \] \\
\[ S \] \[ S \] \\
Lane 1

\[ C[i:i+1] \] \[ B[i:i+1] \] \\
\[ L \] \[ L \] \[ L \] \\
\[ A[i] \] \[ +2 \] \\
\[ \& \] \[ \& \] \\
\[ \& \] \[ \& \] \\
\[ S \] \[ S \] \\
Lane 2

Cost = -2

No Reordering

Vectorized!
State-of-the-art [Look-Ahead SLP CGO’18]

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

```
  C[i+0] B[i+0]   A[i+0] E[i+0]   D[i+0] E[i+1]  
      L          L          L          L  
        +          +          +          +  
       &          &          &          &  
      S          S          S          S  

Lane 1

  B[i+1] D[i+1]   C[i+1] E[i+1]  
      L          L          L          L  
        +          +          +          +  
       &          &          &          &  
      S          S          S          S  

Lane 2
```

Cost = -2
No Reordering
SLP Vectorized!

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

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

![Diagram showing the process of forming Multi-Nodes and reordering operands using Look-Ahead heuristic.](http://vporpo.me)
State-of-the-art [Look-Ahead SLP CGO’18]

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

SLP: Multi-Node

LSLP: Multi-Node

Lane 1

Lane 2

Cost = -2

No Reordering

Vectorized!
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

- Form Multi-Nodes and reorder operands with Look-Ahead heuristic
State-of-the-art [Look-Ahead SLP CGO’18]

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

**SLP and LSLP**

![Diagram showing SLP and LSLP](http://vporpo.me)
State-of-the-art [Look-Ahead SLP CGO’18]

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

Cost = \(-2\)

SLP Vectorized!

\[C[i:i+1] B[i:i+1]\]

No Reordering

\[\text{SLP Reordering}!\]
State-of-the-art [Look-Ahead SLP CGO’18]

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

SLP:

```
S S
A[i:i+1] L L
B[i:i+1] L L
C[i:i+1] B[i:i+1] L L
D[i:i+1] L L
```

Cost = -2

No Reordering

Vectorized!

SLP: Multi-Node

```
S S
A[i:i+1] L L
B[i:i+1] L L
C[i:i+1] B[i:i+1] L L
D[i:i+1] L L
```

Cost = -2

No Reordering

Vectorized!
State-of-the-art [Look-Ahead SLP CGO’18]

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

LSLP: Multi-Node

\[
\begin{align*}
A[i+0] &\quad B[i+0] \\
C[i+1] &\quad B[i+1] \\
D[i+0] &\quad A[i+0]E[i+0] \\
D[i+1] &\quad A[i+1]E[i+1]
\end{align*}
\]

Cost = $-2$

No Reordering

Vectorized!

\[
\begin{align*}
A[i+1] &\quad C[i:i+1]B[i:i+1]
\end{align*}
\]

Cost = $-10$

LSLP Reordering!
State-of-the-art [Look-Ahead SLP CGO’18]

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

SLP: Multi-Node

LSLP: Multi-Node

Cost = −10

SLP: Fully Vectorized!

Cost = −2

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

- The inverse element of ADD(+) is SUB(-)
Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements
Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements
Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements
Multi-Node (LSLP) VS Super-Node (SuperNode-SLP)

- The inverse element of ADD(+) is SUB(-)
- Multi-Nodes cannot handle inverse elements
- Super-Nodes can reorder across them when legal
LSLP fails in the presence of inverse elements

```plaintext
long A[], B[], C[], D[];
A[i+0] = B[i+0] - C[i+0] + D[i+0];
A[i+1] = D[i+1] - C[i+1] + B[i+1];
```
LSLP fails in the presence of inverse elements

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

![Diagram](http://vporpo.me)
LSLP fails in the presence of inverse elements

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

![Diagram of LSLP](http://vporpo.me)
LSLP fails in the presence of inverse elements

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

![Diagram of LSLP](image)
LSLP fails in the presence of inverse elements

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

![Diagram of LSLP and (L)SLP operations](http://vporpo.me)
LSLP fails in the presence of inverse elements

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

Diagram:

```
B[i+0]  C[i+0]  D[i+1]  C[i+1]
L L L L
D[i+0]  
L L L L
B[i+1]  
L L L L
A[i:i+1] S S S
A[i+0]  
S S S S
A[i+1]  
```
LSLP fails in the presence of inverse elements

long A[], B[], C[], D[];
A[i+0] = B[i+0] - C[i+0] + D[i+0];
A[i+1] = D[i+1] - C[i+1] + B[i+1];
LSLP fails in the presence of inverse elements

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

- $A[i+0] = B[i+0] - C[i+0] + D[i+0]$;

Diagram:

```
(\text{L}) \text{SLP}
```

- $B[i+0] C[i+0]$;
- $D[i+1] C[i+1]$;
- $B[i+0] D[i+1]$;
- $B[i+1] C[i+1]$;
- $A[i:i+1]$;

Slide 7 of 15

http://vporpo.me
LSLP fails in the presence of inverse elements

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

The diagram illustrates the dependencies and operations in the LSLP (Limited Shift Left Prediction) algorithm. The cost of the operation is calculated as 0, indicating it is not vectorized.
LSLP fails in the presence of inverse elements

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

\[ A[i+0] = B[i+0] - C[i+0] + D[i+0]; \]
\[ A[i+1] = D[i+1] - C[i+1] + B[i+1]; \]

Cost = 0
Not Vectorized

\[ A[i:i+1] \]
\[ B[i] D[i+1] C[i:i+1] \]
\[ +2 \]
\[ -1 \]
\[ -1 \]
\[ +2 \]
\[ \]

\[ A[i:i+1] \]
\[ S S \]
\[ -1 \]

\[ A[i+0] \]
\[ A[i+1] \]
LSLP fails in the presence of inverse elements

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

![Diagram showing the operations and costs for LSLP and SN-SLP with inverse elements.](http://vporpo.me)
LSLP fails in the presence of inverse elements

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

**SN-SLP**

- **Cost = 0**
- **Not Vectorized**

**LPM**

- **B[i] D[i+1] C[i:i+1]**
- **Cost = 0**
- **Not Vectorized**

```c
A[i+0] = B[i+0] - C[i+0] + D[i+0];
A[i+1] = D[i+1] - C[i+1] + B[i+1];
```
LSLP fails in the presence of inverse elements

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


B[i] D[i+1] C[i:i+1]
+2 L L L L -1

D[i] B[i+1]
-2 L L +2

Cost = 0 Not Vectorized

A[i:i+1] S S -1

A[i:i+1] S S

Reordering!

SuperNode

SuperNode

SN-SLP

(L) SLP
```
LSLP fails in the presence of inverse elements

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

---

**Cost = 0**

SuperNode

Not Vectorized

---

slide 7 of 15

http://vporpo.me
LSLP fails in the presence of inverse elements

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

![Diagram showing the operations and vectorization benefits of LSLP and SN-SLP](http://vporpo.me)
LSLP fails in the presence of inverse elements

\[
A[i+0] = B[i+0] - C[i+0] + D[i+0];
A[i+1] = D[i+1] - C[i+1] + B[i+1];
\]

---

Cost = 0

Not Vectorized
LSLP fails in the presence of inverse elements

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

Cost = 0

Not Vectorized
LSLP fails in the presence of inverse elements

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

**Diagram:**

- **LSLP**
  - Cost = 0
  - Not Vectorized

- **SN-SLP**
  - Cost = -6
  - Vectorized!
SuperNode internal nodes can be reordered too

```c
long A[], B[], C[], D[];
A[i+0] = B[i+0] - C[i+0] + D[i+0];
A[i+1] = B[i+1] + D[i+1] - C[i+1];
```
SuperNode internal nodes can be reordered too

```c
long A[], B[], C[], D[];
A[i+0] = B[i+0] - C[i+0] + D[i+0];
A[i+1] = B[i+1] + D[i+1] - C[i+1];
```
SuperNode internal nodes can be reordered too

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

![Diagram of SuperNode operations](http://vporpo.me)
SuperNode internal nodes can be reordered too

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

\[
\begin{align*}
B[i+0] & \quad C[i+0] \quad D[i+1] \quad B[i+1] \\
L & \quad L & \quad L & \quad L \\
- & \quad + & \quad - & \quad + \\
\quad & \quad & \quad & \\
S & \quad & \quad & \quad \\
A[i+0] & \quad & \quad & \quad \\
\end{align*}
\]
SuperNode internal nodes can be reordered too

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

![SuperNode diagram](http://vporpo.me)
SuperNode internal nodes can be reordered too

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

![Diagram showing the operation of SuperNode internal nodes](http://vporpo.me)
SuperNode internal nodes can be reordered too

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

```
B[i+0] C[i+0] D[i+1] B[i+1]
```

```
A[i+0] A[i+1]
```

```
S
```

```
D[i+0] C[i+1]
```

```
+ −
```

```
B[i:i+1] C[i] D[i+1]
```

```
( L ) SLP
```

```
S S
```

```
A[i:i+1]
```

slide 8 of 15
SuperNode internal nodes can be reordered too

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

```
B[i+0] C[i+0] D[i+1] B[i+1]
  \( - \) \( + \) \( - \) \( + \)
  \( \downarrow \) \( \downarrow \) \( \downarrow \) \( \downarrow \)
A[i+0] A[i+1]
```

```
B[i:i+1] C[i] D[i+1]
  \( - \) \( + \) \( - \)
  \( \downarrow \) \( \downarrow \) \( \downarrow \)
A[i:i+1] S S
```
SuperNode internal nodes can be reordered too

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

\[
\begin{align*}
B[i+0] - C[i+0] + D[i+1] = & \quad B[i+1] + D[i+1] - C[i+1] \\
A[i+0] &= A[i+1] \\
\end{align*}
\]

Cost = +4

Not Vectorized

http://vporpo.me
SuperNode internal nodes can be reordered too

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

Cost = +4

Not Vectorized
SuperNode internal nodes can be reordered too

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

Diagram:
- **BLD**
- **DiAG**
- **SLP**
- **SN-SLP**

Cost: +4

Not Vectorized
SuperNode internal nodes can be reordered too

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

Cost = +4

Not Vectorized

A[i:i+1] $S\ S$ $-1$

B[i:i+1] $C[i]D[i+1]$ $-1$

D[i]C[i+1] $+2$

+1 $- +$

(L) SLP

SN-SLP
SuperNode internal nodes can be reordered too

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

\[
A[i+0] = B[i+0] - C[i+0] + D[i+0];
\]

\[
A[i+1] = B[i+1] + D[i+1] - C[i+1];
\]

\[
\text{Cost} = +4
\]

\[
\text{Not Vectorized}
\]
SuperNode internal nodes can be reordered too

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

Cost = +4

Not Vectorized

SN-SLP

(L)-SLP

B[i:i+1] C[i] D[i+1]

D[i] C[i] B[i:i+1]

A[i:i+1]

http://vporpo.me
SuperNode internal nodes can be reordered too

\[
\begin{align*}
\text{long } & \ A[], B[], C[], D[]; \\
& \ A[i+0] = B[i+0] - C[i+0] + D[i+0]; \\
& \ A[i+1] = B[i+1] + D[i+1] - C[i+1]; \\
\end{align*}
\]

\[
\begin{align*}
& \ B[i+0] \ C[i+0] \ B[i+1] \ C[i+1] \\
& \ L \ L \ L \ L \ L \ L \ L \\
& \ D[i+0] \ D[i+1] \\
& \ L \ L \ L \ L \ L \ L \ L \\
& \ A[i+0] \ A[i+1] \\
& \ + \ - \ + \ - \\
& \ S \ S \ S \ S \ S \ S \ S \\
& \ L \ L \ L \ L \ L \ L \ L \\
\end{align*}
\]

\[
\begin{align*}
& \text{Cost} = +4 \\
& \text{Not Vectorized}
\end{align*}
\]
SuperNode internal nodes can be reordered too

\[
\text{long } A[], B[], C[], D[]; \\
A[i+0] = B[i+0] - C[i+0] + D[i+0]; \\
A[i+1] = B[i+1] + D[i+1] - C[i+1]; \\
\]

Cost = +4
SuperNode internal nodes can be reordered too

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

Cost = +4

Not Vectorized
SuperNode internal nodes can be reordered too

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

**SN-SLP**

**L-SLP**

Cost = +4

```
B[i:i+1] C[i:i+1] D[i+1] C[i+1]
```

Cost = −6

```
B[i:i+1] C[i] D[i+1]
```

Vectorized!

Not Vectorized
Legality

Accumulated Path Operation (APO)

Linearized

A

B C D

G + H − E F

I + J − K

http://vporpo.me
Legality

Accumulated Path Operation (APO)  Linearized

A  B  C  D
G +  H -  E  F
I +  J -  K

slide 9 of 15
http://vporpo.me
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Linearized</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>+</td>
<td>A</td>
</tr>
</tbody>
</table>

Slide 9 of 15

http://vporpo.me
Legality

### Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

### Linearized

A

+
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Linearized</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>+</td>
</tr>
</tbody>
</table>

Slide 9 of 15
Legality

### Sum

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
<th>Linearized</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>+</td>
</tr>
</tbody>
</table>

### Diagram

- **A**: $+$
- **B**: $+$
- **C**: $-$
- **D**: $+$
- **E**: $+$
- **F**: $-$
- **G**: $+$
- **H**: $-$
- **I**: $+$
- **J**: $-$
- **K**: $-$

[Slide 9 of 15](http://vporpo.me)
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0 +</td>
</tr>
<tr>
<td>B</td>
<td>0 +</td>
</tr>
<tr>
<td>C</td>
<td>0 +</td>
</tr>
</tbody>
</table>

Linearized

A
B
C

Sum=0
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
</tbody>
</table>

Linearized

A

+ B

+ C

slide 9 of 15

http://vporpo.me
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

Linearized

A → +B → +C → +D

Slide 9 of 15
Legality

```
+    +    +    -
A    B    C    D

G    +    H    -    E    +    F

I    +    J    -

K    -
```

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
</tr>
</tbody>
</table>

```

Linearized

A
  +
  B
  +
  C
  +
  D
  +
  E
  +
  F
  +
  G
  +
  H
  +
  I
  +
  J
  +
  K
```
### Legality

#### Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Linearized

- A → B → C → D
- E → F
- G → H → I → J → K

---

[Source](http://vporpo.me)
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Linearized</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+</td>
</tr>
<tr>
<td>B</td>
<td>+</td>
</tr>
<tr>
<td>C</td>
<td>+</td>
</tr>
<tr>
<td>D</td>
<td>−</td>
</tr>
<tr>
<td>E</td>
<td>−</td>
</tr>
<tr>
<td>F</td>
<td>+</td>
</tr>
<tr>
<td>G</td>
<td>+</td>
</tr>
<tr>
<td>H</td>
<td>+</td>
</tr>
<tr>
<td>I</td>
<td>+</td>
</tr>
<tr>
<td>J</td>
<td>−</td>
</tr>
<tr>
<td>K</td>
<td>+</td>
</tr>
</tbody>
</table>
Legality

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

slide 9 of 15
http://vporpo.me
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
</tr>
</tbody>
</table>
### Legality

#### Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0 +</td>
</tr>
<tr>
<td>B</td>
<td>0 +</td>
</tr>
<tr>
<td>C</td>
<td>0 +</td>
</tr>
<tr>
<td>D</td>
<td>1 −</td>
</tr>
<tr>
<td>E</td>
<td>1 −</td>
</tr>
<tr>
<td>F</td>
<td>2 +</td>
</tr>
<tr>
<td>G</td>
<td>0 +</td>
</tr>
<tr>
<td>H</td>
<td>0 +</td>
</tr>
<tr>
<td>I</td>
<td>0 +</td>
</tr>
<tr>
<td>J</td>
<td>1 −</td>
</tr>
<tr>
<td>K</td>
<td>0 +</td>
</tr>
</tbody>
</table>

#### Linearized

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J
- K
Legality

![Diagram showing a network of nodes and arrows with labels A to K and operations (+ or -)]

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
</tr>
</tbody>
</table>

Linearized:

A → B → C → D → E → F → G → H → I → J → K

http://vporpo.me
### Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0 +</td>
</tr>
<tr>
<td>B</td>
<td>0 +</td>
</tr>
<tr>
<td>C</td>
<td>0 +</td>
</tr>
<tr>
<td>D</td>
<td>1 −</td>
</tr>
<tr>
<td>E</td>
<td>1 −</td>
</tr>
<tr>
<td>F</td>
<td>2 +</td>
</tr>
<tr>
<td>G</td>
<td>0 +</td>
</tr>
<tr>
<td>H</td>
<td>0 +</td>
</tr>
<tr>
<td>I</td>
<td>0 +</td>
</tr>
<tr>
<td>J</td>
<td>1 −</td>
</tr>
<tr>
<td>K</td>
<td>0 +</td>
</tr>
</tbody>
</table>

### Linearized Operations

- A
  - B
  - C
  - D
  - E
  - F
  - G
  - H
  - I
  - J
  - K
Legality

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
</tr>
</tbody>
</table>

Linearized:

A → B → C → D → E → F → G → H → I → J → K
Legality

Accumulated Path Operation (APO)

<table>
<thead>
<tr>
<th>Sum</th>
<th>Accumulated Path Operation (APO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+</td>
</tr>
<tr>
<td>B</td>
<td>+</td>
</tr>
<tr>
<td>C</td>
<td>+</td>
</tr>
<tr>
<td>D</td>
<td>1 −</td>
</tr>
<tr>
<td>E</td>
<td>1 −</td>
</tr>
<tr>
<td>F</td>
<td>2 +</td>
</tr>
<tr>
<td>G</td>
<td>0 +</td>
</tr>
<tr>
<td>H</td>
<td>+</td>
</tr>
<tr>
<td>I</td>
<td>0 +</td>
</tr>
<tr>
<td>J</td>
<td>0 −</td>
</tr>
<tr>
<td>K</td>
<td>0 +</td>
</tr>
</tbody>
</table>

Linearized

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J
- K

slide 9 of 15

http://vporpo.me
SN-SLP Algorithm

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

Scalar IR
1. Find seed instructions for vectorization
SN-SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Get next seed group
SN-SLP Algorithm

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

• Graph contains groups of vectorizable instructions
SN-SLP Algorithm

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

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

Scalar IR

1. Find seed instructions for vectorization
2. Get next seed group
3. Generate the SLP graph
4. Calculate cost of vectorization
5. cost < threshold
SN-SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Get next seed group
3. Generate the SLP graph
4. Calculate cost of vectorization
5. cost<threshold
   - NO
   - cost>threshold
     6. Generation of vector code

http://vporpo.me
SN-SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Get next seed group
3. Generate the SLP graph
4. Calculate cost of vectorization

- NO
  - cost < threshold
- YES
  - Generation of vector code
SN-SLP Algorithm

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

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

Scalar IR
1. Find seed instructions for vectorization
2. Get next seed group
3. Generate the SLP graph
4. Calculate cost of vectorization
5. NO, cost < threshold
6. YES, Generation of vector code
7. Remove current seed group
8. Seeds left?
SN-SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Get next seed group
3. Generate the SLP graph
4. Calculate cost of vectorization
5. NO \( \text{cost} < \text{threshold} \)
6. YES Generation of vector code
7. Remove current seed group
8. Seeds left? YES
SN-SLP Algorithm

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

Diagram:

1. Find seed instructions for vectorization
2. Get next seed group
3. Generate the SLP graph
4. Calculate cost of vectorization
5. If cost < threshold then YES, if not NO
6. Generation of vector code
7. Remove current seed group
8. If seeds left? then YES, if not NO, DONE
Experimental Setup

- Implemented in LLVM trunk
Experimental Setup

- Implemented in LLVM trunk
- Target: Intel® Core™ i5-6440HQ CPU
Experimental Setup

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

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

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

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

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

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

slide 12 of 15
Performance (Full Benchmarks)

GMean

LSLP  SNSLP

433.milc  464.h264ref  435.gromacs  447.dealII  453.povray  454.calculix

http://vporpo.me
Conclusion

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

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

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

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

• Better at identifying isomorphism

• Implemented in LLVM as an extension of SLP
Conclusion

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