Look-Ahead SLP: Auto-vectorization in the presence of commutative operations

Vasileios Porpodas\textsuperscript{1}, Rodrigo C. O. Rocha\textsuperscript{2} and Luís F. W. Góes\textsuperscript{3}

\textsuperscript{1}Intel Santa Clara, USA
\textsuperscript{2}University of Edinburgh, UK
\textsuperscript{3}PUC Minas, Brazil

CGO 2018
Legal Disclaimer & Optimization Notice

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.

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.

Copyright 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s 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.

Notice revision #20110804
SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI’00]
SLP Straight-Line Code Vectorizer

- **Superword Level Parallelism** [Larsen PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP
SLP Straight-Line Code Vectorizer

- **Superword Level Parallelism** [Larsen PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP

\[
\begin{align*}
A[i+0] &= B[i+0] \\
A[i+1] &= B[i+1]
\end{align*}
\]
SLP Straight-Line Code Vectorizer

- **Superword Level Parallelism** [Larsen PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP

\[
A[i+0] = B[i+0] \\
A[i+1] = B[i+1] \\
A[i+1:i+0] = B[i+1:i+0]
\]
SLP Straight-Line Code Vectorizer

- Superword Level Parallelism [Larsen PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP

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

- SLP and loop-vectorizer complement each other:
SLP Straight-Line Code Vectorizer

- **Superword Level Parallelism** [Larsen PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP

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

- SLP and loop-vectorizer complement each other:
  - 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]
- GCC and LLVM implementations are based on Bottom-Up SLP

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

- SLP and loop-vectorizer complement each other:
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed
- It is missing features present in the Loop vectorizer (e.g., Interleaved Loads, Predication)
SLP Straight-Line Code Vectorizer

- **Superword Level Parallelism** [Larsen PLDI’00]
- GCC and LLVM implementations are based on Bottom-Up SLP

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

- SLP and loop-vectorizer complement each other:
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed
- It is missing features present in the Loop vectorizer (e.g., Interleaved Loads, Predication)
  - Usually run SLP after the Loop Vectorizer
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub2};
\end{align*}
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub2};
\end{align*}
\]

\[\ldots \quad B[i+0] \quad B[i+1] \quad \ldots \quad \ldots\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} & = \ldots - \ldots; \\
\text{sub2} & = \ldots - \ldots; \\
A[i+0] & = \text{sub1} + B[i+0]; \\
A[i+1] & = B[i+1] + \text{sub2}; \\
\end{align*}
\]

![Naive SLP diagram](http://vporpo.me)
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub}1 &= \ldots - \ldots; \\
\text{sub}2 &= \ldots - \ldots; \\
A[i+0] &= \text{sub}1 + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub}2;
\end{align*}
\]

... ... ... B[i+0] B[i+1] ... ... ...
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub2};
\end{align*}
\]

Naive SLP

\[
\begin{align*}
\text{A}[i+0] & \quad \text{B}[i+0] & \quad \text{B}[i+1] & \quad \text{A}[i+1] & \quad \ldots & \quad \ldots \\
- & \quad + & \quad L & \quad L & \quad - & \quad + & \quad + \\
\end{align*}
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

```
sub1 = \ldots - \ldots;
sub2 = \ldots - \ldots;
A[i+0] = \textcolor{red}{\text{sub1}} + B[i+0];
A[i+1] = \textcolor{red}{B[i+1]} + \text{sub2};
```

![Naive SLP Diagram]

![SLP Diagram]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\text{sub1} = \ldots - \ldots;
\text{sub2} = \ldots - \ldots;
\text{A}[i+0] = \text{sub1} + \text{B}[i+0];
\text{A}[i+1] = \text{B}[i+1] + \text{sub2};
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub2}; \\
\end{align*}
\]

\[
\cdots \quad B[i+0] \quad B[i+1] \quad \cdots \quad \cdots
\]

Naive SLP
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
\text{A}[i+0] &= \text{sub1} + \text{B}[i+0]; \\
\text{A}[i+1] &= \text{B}[i+1] + \text{sub2};
\end{align*}
\]

```
\ldots \quad \ldots \quad \ldots
\ldots \quad \ldots \quad \ldots
```

Vectorization would STOP without reordering!

Naive SLP

```
- +
\text{S} \quad \text{A}[i+0]
```

```
- +
\text{S} \quad \text{A}[i+1]
```

```
+ +
\quad \text{S} \quad \text{S} \quad \text{A}[i:i+1]
```
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub2};
\end{align*}
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= B[i+1] + \text{sub2};
\end{align*}
\]

\[\ldots B[i+0] \quad B[i+1] \quad \ldots \quad \ldots\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= \text{sub2} + B[i+1]; \\
\end{align*}
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= \text{sub2} + B[i+1];
\end{align*}
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= \text{sub2} + B[i+1]; \\
\end{align*}
\]
Why commutative operations matter?

- Operands can be reordered
- It can change the shape of the DAG
- Affects SLP ability to vectorize
- SLP is effective when the immediate operands differ

\[
\begin{align*}
\text{sub1} &= \ldots - \ldots; \\
\text{sub2} &= \ldots - \ldots; \\
A[i+0] &= \text{sub1} + B[i+0]; \\
A[i+1] &= \text{sub2} + B[i+1];
\end{align*}
\]

Vectorization continues!
SLP not effective in more complex cases

- SLP reordering not effective for:
SLP not effective in more complex cases

- SLP reordering not effective for:
  1. Load address mismatch further up the graph
SLP not effective in more complex cases

- SLP reordering not effective for:
  1. Load address mismatch further up the graph
  2. Opcode mismatch further up the graph
SLP not effective in more complex cases

- SLP reordering not effective for:
  1. Load address mismatch further up the graph
  2. Opcode mismatch further up the graph
  3. Reordering across chains of commutative operations
SLP not effective in more complex cases

- SLP reordering not effective for:
  1. Load address mismatch further up the graph
  2. Opcode mismatch further up the graph
  3. Reordering across chains of commutative operations

- Look-Ahead SLP (LSLP) provides a solution to all three.
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
1/3 Load address mismatch

long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

```
B[i+0]  C[i+0]   C[i+1]  B[i+1]
  ↓  1  ↓  2  ↓  3  ↓  4
   &  &  &  &
    ↓  ↓  ↓  ↓
   A[i+0]  A[i+1]
```

Lane 1  Lane 2

SLP

Non-Vectorizable  Vectorizable  +/− Cost
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

![Diagram of load address mismatch](http://vporpo.me)
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

**SLP**

- **B[i+0]**
- **C[i+0]**
- **C[i+1]**
- **B[i+1]**

Lane 1: $A[i+0]$  
Lane 2: $A[i+1]$

- No Reordering
- Non-Vectorizable
- Cost Vectorizable

[Image: http://vporpo.me]
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

![Image of diagram with code snippet]
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

SLP

- **B[i+0] C[i+0]**
  - Lane 1

- **C[i+1] B[i+1]**
  - Lane 2

- **B[i+0] C[i+1]**
  - No Reordering

Non-Vectorizable  Vectorizable  +/- Cost
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

SLP

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

C[i+1] B[i+1]

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

A[i:i+1]

Lane 1

Lane 2

Non-Vectorizable  Vectorizable  +/-#Cost

No Reordering
1/3 Load address mismatch

long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

Lane 1

Lane 2

SLP

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

C[i+1] B[i+1]

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

No Reordering

Non-Vectorizable  Vectorizable  +/- Cost

slide 6 of 18

http://vporpo.me
1/3 Load address mismatch

long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

B[i+0] C[i+0] B[i+1] C[i+0] +2
L L L L

C[i+1] B[i+1] C[i+1] B[i+1] 1 3
L L L L

SLP

No Reordering

& & -1

Cost = 0

Lane 1

Lane 2

Non-Vectorizable  Vectorizable  +/- #Cost
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

**SLP**

- **B[i+0] C[i+0]**
  - Lane 1
- **C[i+1] B[i+1]**
  - Lane 2

**B[i+0] C[i+1] B[i+1] C[i+0]**

- **SLP**
  - No Reordering
  - Not Vectorized!

**Non-Vectorizable**

**Vectorizable**

Cost = 0

slide 6 of 18

http://vporpo.me
1/3 Load address mismatch

long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

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

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

SLP

Cost = 0

Not Vectorized!

LSLP

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

Non-Vectorizable

Vectorizable +/#Cost

http://vporpo.me
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

**SLP**

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

**LSLP**

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

No Reordering

Not Vectorized!

Cost = 0

Non-Vectorizable  Vectorizable  +/- #Cost

http://vporpo.me
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

B[i+0] C[i+0]  B[i+1] C[i+1]
L 1 L 2 
<< & << &
S S
A[i+0]   A[i+1]
Lane 1   Lane 2

SLP:

B[i+0] C[i+1]  B[i+1] C[i+0]
+2 L L +2 
<< & << &
S S
A[i:i+1]  Cost = 0

LSLP:

Not Vectorized!

Non-Vectorizable  Vectorizable +/− #Cost

http://vporpo.me
1/3 Load address mismatch

long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

B[i+0] C[i+0]
L 1 2
<< & S
A[i+0]
Lane 1

B[i+1] C[i+1]
L 4 3
<< & S
A[i+1]
Lane 2

Not Vectorized!
No Reordering
Cost = 0

Vectorizable +/- #Cost
Non-Vectorizable

slide 6 of 18
http://vporpo.me
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

---

**LSLP:**

- **Lane 1:**
  - `B[i+0]` & `C[i+0]` = `A[i+0]`
  - `B[i+1]` & `C[i+1]` = `A[i+1]`

**Lane 2:**

- `B[i+0]` & `C[i+0]` = `A[i+0]`
- `B[i+1]` & `C[i+1]` = `A[i+1]`

---

**SLP:**

- **No Reordering**
- **Cost = 0**
- **Not Vectorized!**

---

**LSLP**

- **Reordering!**
- **Non-Vectorizable**
- **Vectorizable +/#Cost**
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

---

SLP:
- Lane 1: No Reordering
- Lane 2: Not Vectorized!

LSLP:
- Lane 1: Reordering!
- Lane 2: Non-Vectorizable

Cost = 0

---

Non-Vectorizable  Vectorizable  +/- Cost

slide 6 of 18
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

LSLP:

- **Lane 1**
  - ```
    B[i+0] C[i+0]
    &
    <<
    S
    A[i+0]
  ```
- **Lane 2**
  - ```
    B[i+1] C[i+1]
    &
    <<
    S
    A[i+1]
  ```

SLP:

- **No Reordering**
  - ```
    B[i+0] C[i+1] B[i+1] C[i+0]
    & &
    S S
    A[i:i+1]
    Cost = 0
  ```

LSLP Reordering:

- ```
    B[i:i+1]
    &
    <<
    S
    A[i:i+1]
  ```
1/3 Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

LSLP:
- Lane 1: `A[i+0]` (B[i+0] << 1 & C[i+0] << 2)
- Lane 2: `A[i+1]` (C[i+1] << 3 & B[i+1] << 4)

SLP:
- No Reordering
- Cost = 0

**Not Vectorized!**
Load address mismatch

```c
long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);
```

### SLP

- **B[i+0] C[i+0]**
- **B[i+1] C[i+1]**

No Reordering

Cost = 0

### LSLP

- **A[i+0]**
  - Lane 1
  - LSLP: &
  - S
  - A[i+0]

- **A[i+1]**
  - Lane 2
  - LSLP: &
  - S
  - A[i+1]

- **B[i+0] C[i+0]**
- **B[i+1] C[i+1]**

Not Vectorized!

Cost = −6

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

Non-Vectorizable

Vectorizable +/− #Cost

http://vporpo.me

slide 6 of 18
1/3 Load address mismatch

long A[], B[], C[];
A[i+0] = (B[i+0] << 1) & (C[i+0] << 2);
A[i+1] = (C[i+1] << 3) & (B[i+1] << 4);

---

// Diagram
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = (B[2*i] << 1) & 0x11 + (C[2*i] + 2) & 0x12;
A[i+1] = (D[2*i] + 3) & 0x13 + (E[2*i] << 4) & 0x14;
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
2/3 Opcodes mismatch

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11</td>
<td>2</td>
<td>0x12</td>
<td>3</td>
</tr>
<tr>
<td>&amp;</td>
<td></td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A[i+0]</td>
<td>A[i+1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Lane 1

Lane 2

SLP

Non-Vectorizable Vectorizable +/-# Cost
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1

Lane 2
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1
Lane 2

SLP

No Reordering

++

A[i:i+1]

Non-Vectorizable  Vectorizable  +/-# Cost

slide 7 of 18
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1

Lane 2

Non-Vectorizable  Vectorizable  +/- Cost

SLP

No Reordering

http://vporpo.me
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1

Lane 2

---

Non-Vectorizable  Vectorizable  +/-#Cost
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 2:  

Non-Vectorizable  Vectorizable  +/-# Cost
2/3 Opcodes mismatch

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
```

![Diagram showing the calculation process for A[i+0] and A[i+1].](http://vporpo.me)
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1
Lane 2

SLP
No Reordering

Cost = +4

Non-Vectorizable  Vectorizable  +/-# Cost

http://vporpo.me
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

SLP
No Reordering
Not Vectorized!
Cost = +4

Lane 1
Lane 2

Non-Vectorizable  Vectorizable  +/-#Cost

http://vporpo.me
2/3 Opcodes mismatch

```
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
```
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1
A[i+0]

Lane 2
A[i+1]

+2
0x11 <<
+ 0x12
0x13 +
<< 0x14

SLP
No Reordering
Not Vectorized!

Cost = +4
A[i:i+1]
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1
A[i+0]
Lane 2
A[i+1]

SLP
+2
0x11 << + 0x12
0x13 + << 0x14

No Reordering
Not Vectorized!
A[i:i+1]
Cost = +4

Non-Vectorizable
Vectorizable
+/−# Cost

slide 7 of 18
http://vporpo.me
2/3 Opcodes mismatch

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
```

Lane 1 Lane 2

---

Cost = +4
No Reordering
Not Vectorized!
2/3 Opcodes mismatch

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);
```

---

Slide 7 of 18

http://vporpo.me
unpack_in 2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

|  
|  
|  
2  1  3  4  

Lane 1  Lane 2

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

SLP

Not Vectorized!
Cost = +4
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1
Lane 2

++ 0x11
++ 0x13
++ 0x12
++ 0x14

SLP

Cost = +4

Non-Vectorizable
Vectorizable
+/−# Cost

http://vporpo.me
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

Lane 1 Lane 2
A[i+0] A[i+1]

Cost = +4
No Reordering
Not Vectorized!

SLP Reordering!
Vectorizable +/-# Cost
Non-Vectorizable

http://vporpo.me
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);


Lane 1

Lane 2

Non-Vectorizable  Vectorizable  +/-#Cost
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);


Lane 1 Lane 2

+2 +2 +2 +2
0x11 << + 0x12 0x13 + << 0x14

SLP

No Reordering

Not Vectorized!

A[i:i+1]

-1
& &
+
+
& &

Cost = +4

SLP Reordering!

Non-Vectorizable Vectorizable +/- Cost

http://vporpo.me
2/3 Opcodes mismatch

unsigned long A[], B[], C[], D[], E[];
A[i+0] = (B[2*i] << 1) & 0x11 + (C[2*i] + 2) & 0x12;
A[i+1] = (D[2*i] + 3) & 0x13 + (E[2*i] << 4) & 0x14;

Lane 1
Lane 2

Cost = +4
No Reordering

Cost = -2
LSLP Reordering!
unsigned long A[], B[], C[], D[], E[];
A[i+0] = ((B[2*i] << 1) & 0x11) + ((C[2*i] + 2) & 0x12);
A[i+1] = ((D[2*i] + 3) & 0x13) + ((E[2*i] << 4) & 0x14);

---

SLP

No Reordering

Not Vectorized!

A[i:i+1]

Cost = +4

---

LSLP

Vectorized!

LSLP Reordering!

A[i:i+1]

Cost = -2
3/3 Chains of Commutative Operations

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
```
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
3/3 Chains of Commutative Operations

unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
**3/3 Chains of Commutative Operations**

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
```

![Diagram showing chains of commutative operations with lane 1 and lane 2.]
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
3/3 Chains of Commutative Operations

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
```

Slide 8 of 18

http://vporpo.me
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
3/3 Chains of Commutative Operations

Unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

Cost = -2
No Reordering
Non-Vectorizable
+/#Cost

Slide 8 of 18
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
3/3 Chains of Commutative Operations

unsigned long A[], B[], C[], D[], E[];
A[i+0]=A[i+0]&(B[i+0]+C[i+0])&(D[i+0]+E[i+0]);
A[i+1]=(D[i+1]+E[i+1])&(B[i+1]+C[i+1])&A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

3/3 Chains of Commutative Operations

LSLP:

Vectorized!

Non-Vectorizable
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

SLP

Vectorized!
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

LSLP:

C[i+0] B[i+0]
+ +
& &
A[i+0] E[i+0]
C[i+1] B[i+1]
+ +
& &
A[i+1] E[i+1]

Multi-Node

Lane 1

Lane 2

Vectorized!

No Reordering

Cost = -2

Non-Vectorizable

Vectorizable

+/−# Cost
3/3 Chains of Commutative Operations

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
```

Multi-Node Multi-Node

SLP

Vectorized!

Non-Vectorizable +/- #Cost

slide 8 of 18

http://vporpo.me
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

3/3 Chains of Commutative Operations

SLP

Vectorized!
Cost = -2

Non-Vectorizable

SLP Reordering!

LSLP

Multi-Node

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

D[i+0]

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

A[i+1] E[i+1]

Lane 1

Lane 2

LSLP:

Cost = 2

Multi-Node

A[i+0] B[i+1]

D[i+1]

A[i+1] E[i+1]

A[i+1] B[i+1]

Vectorizable!
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

LSLP:

LSLP Reordering!

Non-Vectorizable  Vectorizable  +/- Cost

slide 8 of 18  http://vporpo.me
3/3 Chains of Commutative Operations

```c
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];
```

**SLP:**
- No Reordering
- Cost = \(-2\)
- Non-Vectorizable

**Vectorized!**
- Cost = \(-2\)
- Vectorizable

**LSLP:**
- Reordering
- Non-Vectorizable
- Vectorizable
- \(+/-\# Cost\)
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

Example:

C[i+0] B[i+0]
\[\begin{array}{c}
L \\
& & \text{Multi-Node} \\
A[i+0] E[i+0]
\end{array}\]

Lane 1

C[i+1] B[i+1]
\[\begin{array}{c}
L \ \\
& & \text{Multi-Node} \\
A[i+1] E[i+1]
\end{array}\]

Lane 2

LSLP:

Cost = -2

Vectorized!

Non-Vectorizable Vectorizable +/- Cost

http://vporpo.me
unsigned long A[], B[], C[], D[], E[];
A[i+0] = A[i+0] & (B[i+0] + C[i+0]) & (D[i+0] + E[i+0]);
A[i+1] = (D[i+1] + E[i+1]) & (B[i+1] + C[i+1]) & A[i+1];

LSLP:

C[i+0] B[i+0]
D[i+0]
A[i+0] E[i+0]
S

A[i+0] Lane 1

C[i+1] B[i+1]
D[i+1]
A[i+1] E[i+1]
S

A[i+1] Lane 2

Vectorized!
Cost = -2

Non-Vectorizable Vectorizable +/-#Cost
(L)SLP Algorithm

Seed instructions are usually:

1. Consecutive Stores
2. Reductions
(L)SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Pop next seed group
(L)SLP Algorithm

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

- Graph contains groups of vectorizable instructions
(L)SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Pop next seed group
3. Generate the (L)SLP graph for seed
4. Calculate cost of vectorization
(L)SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Pop next seed group
3. Generate the (L)SLP graph for seed
4. Calculate cost of vectorization
5. If cost < threshold, pop next seed group
(L)SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Pop next seed group
3. Generate the (L)SLP graph for seed
4. Calculate cost of vectorization
5. \( \text{cost} < \text{threshold} \)
   - YES
6. Replace scalars with vectors
(L)SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Pop next seed group
3. Generate the (L)SLP graph for seed
4. Calculate cost of vectorization
5. cost < threshold
   YES
6. Replace scalars with vectors
7. Emit insert/extract instructions
(L)SLP Algorithm

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

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

Scalar IR

1. Find seed instructions for vectorization

2. Pop next seed group

3. Generate the (L)SLP graph for seed

4. Calculate cost of vectorization

5. Cost < threshold

  NO

  6. Replace scalars with vectors

  7. Emit insert/extract instructions

  9. Seeds left?

  YES

http://vporpo.me
(L)SLP Algorithm

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

Scalar IR

1. Find seed instructions for vectorization
2. Pop next seed group
3. Generate the (L)SLP graph for seed
4. Calculate cost of vectorization
5. NO  cost < threshold
6. Replace scalars with vectors
7. Emit insert/extract instructions
8. Seeds left?
9. YES
(L)SLP Algorithm

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

Entry

Group Formation

SLP

A

B

C

A

B

C

A

B

C

A

B

C

A

B

C

A

B
(L)SLP Graph Formation

SLP

Entry

Group Formation

Have operands

Diagram:

\[ (+ \rightarrow C \rightarrow A \rightarrow +) \rightarrow (+ \rightarrow B \rightarrow \ldots) \]
(L)SLP Graph Formation

SLP

Entry

Group Formation

Have operands

Operand Reordering

Group Formation

Operand Reordering

Have operands

SLP

A
B
C
A
B
C

http://vporpo.me
(L)SLP Graph Formation

SLP

Entry

Group Formation

Have operands

Operand Reordering

A
B
C
A
B
C
A

http://vporpo.me
(L)SLP Graph Formation

SLP

Group Formation

Have operands

Operand Reordering

Entry

Diagram:

A + B -> C
C + D -> E
E + F -> G

http://vporpo.me
(L)SLP Graph Formation

SLP

Entry

Group Formation

Done reordering

Have operands

Operand Reordering

Diagram:

+ A B
+ C
+ B

A

http://vporpo.me
(L)SLP Graph Formation

SLP

Entry

Group Formation

Done reordering

Have operands

Operand Reordering

Diagram:

A
B
C
B

A
C

+ +
(L)SLP Graph Formation

SLP

Entry
Group Formation
Done reordering
Have operands
Operand Reordering

A
B
C
B

+ +

C

+ +

A

(L)SLP Graph Formation

SLP

Group Formation

Done reordering

Have operands

Operand Reordering

Entry

Diagram:

A + B → C + A

http://vporpo.me
(L)SLP Graph Formation

Entry

Group Formation

Done reordering

Have operands

Operand Reordering

Already sorted

A
B
C
B

A

C

+ + +
(L)SLP Graph Formation

SLP

Entry

Group Formation

Done reordering

Have operands

Operand Reordering

A  B  C  B

+  +  +

A

http://vporpo.me
(L)SLP Graph Formation

SLP

Entry → Group Formation

Done reordering → Have operands

Operand Reordering

Diagram:

- Entry
- Group Formation
- Done reordering
- Have operands
- Operand Reordering

Graph:

- Nodes: A, B, C, B, +
- Edges: A → +, B → +, C → +, B → +, A

Slide 10 of 18

http://vporpo.me
(L)SLP Graph Formation

SLP

Entry

Group Formation

Done reordering

Have operands

Operand Reordering

Diagram:

[Diagram showing group formation, operand reordering, and entry points with expressions A, B, C, and B with operators +]
(L)SLP Graph Formation

Have operands

Group Formation

Done reordering

Entry

Operand Reordering

SLP

A + B C + A C B + + C A + +
(L)SLP Graph Formation

- LSLP: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- **SLP**: Extended DAG Formation with Multi-Nodes

- **LSLP**: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- **LSLP**: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- LSLP: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- **SLP:**
  - Entry
  - Group Formation
  - Done reordering
  - Have operands
  -Operand Reordering

- **LSLP:**
  - Entry
  - Group Formation
  - Have commutative operands of same opcode
  - Have operands
  - Multi-Node Formation
(L)SLP Graph Formation

SLP

Entry

Group Formation

Done reordering

Have operands

Operand Reordering

LSLP: Extended DAG Formation with Multi-Nodes

• LSLP: Extended DAG Formation with Multi-Nodes

Entry

Group Formation

Have commutative operands of same opcode

Have operands

Multi-Node Formation

No more operands to add to multi-node
(L)SLP Graph Formation

- LSLP: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- **SLP**
  - Entry
  - Group Formation
  - Done reordering
  - Have operands
  - Operand Reordering

- **LSLP**
  - Entry
  - Group Formation
  - Have commutative operands of same opcode
  - Look-Ahead Operand Reordering
  - Multi-Node Formation
  - Have operands
  - Multi-Node Formation
  - No more operands to add to multi-node

- **LSLP: Extended DAG Formation with Multi-Nodes**
**(L)SLP** Graph Formation

- **SLP**
  - Entry
  - Group Formation
    - Done reordering
    - Have operands
  - Operand Reordering

- **LSLP**
  - Entry
  - Group Formation
    - Have commutative operands of same opcode
  - Look-Ahead Operand Reordering
  - Have operands
  - Multi-Node Formation
    - No more operands to add to multi-node

- **LSLP**: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- LSLP: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- **SLP**
  - Entry
  - Group Formation
  - Done reordering
  - Have operands
  - Operand Reordering

- **LSLP**
  - Entry
  - Group Formation
  - Done reordering
  - Have operands
  - Look-Ahead Operand Reordering
  - Have commutative operands of same opcode
  - Multi-Node Formation
  - No more operands to add to multi-node

- **LSLP: Extended DAG Formation with Multi-Nodes**
(L)SLP Graph Formation

- LSLP: Extended DAG Formation with Multi-Nodes
(L)SLP Graph Formation

- LSLP: Extended DAG Formation with Multi-Nodes
Operand Reordering with Look-Ahead
Operand Reordering with Look-Ahead

Lane 0 Lane 1 Lane 2 Lane 3

D[i+0] D[i+1] D[i+2] E[i] D[i+3]

Node
S S S S

Multi-Node

< L 1 << L 1 << L 1 L 1 slide 11 of 18 http://vporpo.me
Operand Reordering with Look-Ahead

Reordering

Multi-Node

Node

\[ A[i+0] \]
Lane 0

\[ A[i+1] \]
Lane 1

\[ A[i+2] \]
Lane 2

\[ A[i+3] \]
Lane 3

http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Multi-Node

Node

Lane 0    Lane 1    Lane 2    Lane 3
Operand Reordering with Look-Ahead

Look-Ahead Reordering

Multi-Node

Node

Lane 0  Lane 1  Lane 2  Lane 3

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

slide 11 of 18  http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead
Reordering

Multi-Node
Node

Lane 0
Lane 1
Lane 2
Lane 3

Reordering

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operand Slots

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;&lt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&lt;&lt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Operand Reordering with Look-Ahead

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt; &lt;</td>
<td>&lt; &lt;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>&lt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Slide 11 of 18
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Multi-Node

Node

Lane 0 Lane 1 Lane 2 Lane 3

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>☂:1 ☂:2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operand Slots
Operand Reordering with Look-Ahead

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td><img src="slide11.png" alt="Operand Slots" /></td>
<td>1</td>
<td><img src="slide11.png" alt="Operand Slots" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="slide11.png" alt="Operand Slots" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><img src="slide11.png" alt="Operand Slots" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><img src="slide11.png" alt="Operand Slots" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multi-Node

Node

Lane 0

Lane 1

Lane 2

Lane 3

Look-Ahead score

Operand Slots

slide 11 of 18
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Look-Ahead score

Lane 0 | Lane 1 | Lane 2 | Lane 3
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt;&lt;:1 &lt;&lt;:2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>L :N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 1 :N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operand Slots

Lane 0 Lane 1 Lane 2 Lane 3

Node

Multi-Node

Slide 11 of 18

http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead Reordering

Multi-Node

Node

Lane 0: $A[i+0]$ (Lane 0)
Lane 1: $A[i+1]$ (Lane 1)
Lane 2: $A[i+2]$ (Lane 2)
Lane 3: $A[i+3]$ (Lane 3)

Look-Ahead Score

Operands Slots

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>$&lt;$</td>
<td>$&lt;$:1</td>
<td>$&lt;$:2</td>
</tr>
<tr>
<td>1</td>
<td>$&lt;$</td>
<td>$&lt;$:2</td>
<td>$&lt;$:1</td>
</tr>
<tr>
<td>2</td>
<td>$&lt;$</td>
<td>$&lt;$:2</td>
<td>$&lt;$:1</td>
</tr>
<tr>
<td>3</td>
<td>$&lt;$</td>
<td>$&lt;$:2</td>
<td>$&lt;$:1</td>
</tr>
</tbody>
</table>

Slide 11 of 18

http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead Reordering

Multi-Node

Node

Lane 0 Lane 1 Lane 2 Lane 3


Lane 0 Lane 1 Lane 2 Lane 3

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt; &lt;</td>
<td>&lt; :1</td>
<td>&lt; :2</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>&lt; &lt;</td>
<td>&lt; :2</td>
<td>&lt; :1</td>
</tr>
</tbody>
</table>

http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Multi-Node

Node

Table:

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;:1</td>
<td>&lt;&lt;:2</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L: N/A</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1: N/A</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;:2</td>
<td>&lt;&lt;:1</td>
</tr>
</tbody>
</table>

Operand Slots

http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Multi-Nodes

Node

Operand Slots

Lane 0 | Lane 1 | Lane 2 | Lane 3
---|---|---|---
final_order | Look-Ahead | final_order | Look-Ahead | final_order | Look-Ahead | final_order
0 | | | | | | |
1 | | | | :N/A | | |
2 | 1 | 1 | :N/A | 1 | | |
3 | | | | | :2 | :1 |

slide 11 of 18
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Operand Slots

Lane 0 Lane 1 Lane 2 Lane 3

<table>
<thead>
<tr>
<th>final_order</th>
<th>Look-Ahead score</th>
<th>final_order</th>
<th>Look-Ahead score</th>
<th>final_order</th>
<th>Look-Ahead score</th>
<th>final_order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;   :1</td>
<td>&lt;   :2</td>
<td>&lt;</td>
<td>&lt;   :2</td>
<td>&lt;</td>
<td>:1</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L :N/A</td>
<td>L</td>
<td>L :N/A</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1 :N/A</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td>&lt; :2</td>
<td>&lt; :1</td>
<td>&lt;</td>
<td>&lt;</td>
<td></td>
</tr>
</tbody>
</table>

Multi-Node

Node

slide 11 of 18
## Operand Reordering with Look-Ahead

### Look-Ahead Reordering

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>🔄</td>
<td>🔄:1</td>
<td>🔄:2</td>
</tr>
</tbody>
</table>

### Multi-Node

<table>
<thead>
<tr>
<th>Node</th>
<th>A[i+0] (Lane 0)</th>
<th>A[i+1] (Lane 1)</th>
<th>A[i+2] (Lane 2)</th>
<th>A[i+3] (Lane 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D[i+0]</td>
<td>D[i+1]</td>
<td>D[i+2]</td>
<td>D[i+3]</td>
</tr>
</tbody>
</table>

### Reordering

- A[i+0] B[i+0] C[i+0] D[i+0] E[i+0]
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Multi-Node

Node

Lane 0 Lane 1 Lane 2 Lane 3

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&amp; &amp; &amp; &amp;</td>
<td>&amp; &amp; &amp;</td>
<td>&amp; &amp; &amp;</td>
</tr>
<tr>
<td>1</td>
<td>L L :N/A</td>
<td>L L :N/A</td>
<td>L L :N/A</td>
</tr>
<tr>
<td>2</td>
<td>L L :N/A</td>
<td>L L :N/A</td>
<td>L L :N/A</td>
</tr>
<tr>
<td>3</td>
<td>&amp; &amp; :1</td>
<td>&amp; &amp; :1</td>
<td>&amp; &amp; :1</td>
</tr>
</tbody>
</table>

Operand Slots

slide 11 of 18

http://vporpo.me
Operand Reordering with Look-Ahead

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Operand Slots

- Lane 0: 0, 1
- Lane 1: 0, 1
- Lane 2: 0, 1
- Lane 3: 0, 1

Look-Ahead score

- Lane 0: N/A
- Lane 1: N/A
- Lane 2: N/A
- Lane 3: N/A

final_order

- Lane 0: 0, 0, 0, 0
- Lane 1: 0, 0, 0, 0
- Lane 2: 0, 0, 0, 0
- Lane 3: 0, 0, 0, 0

Slide 11 of 18

http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead Reordering

Lane 0  Lane 1  Lane 2  Lane 3
<table>
<thead>
<tr>
<th>final_order</th>
<th>Look-Ahead score</th>
<th>final_order</th>
<th>Look-Ahead score</th>
<th>final_order</th>
<th>Look-Ahead score</th>
<th>final_order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L</td>
<td>N/A</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Operand Slots

Node

Multi-Node

Operand Reordering with Look-Ahead

Look-Ahead Reordering

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
Operand Reordering with Look-Ahead

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt;&lt;</td>
<td>&lt;=:1</td>
<td>&lt;=:2</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L: N/A</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1: N/A</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt;&lt;</td>
<td>&lt;=:2</td>
<td>&lt;=:1</td>
</tr>
</tbody>
</table>

Operand Slots

Node
Multi−Node
Reordering
Look−Ahead

Operands:
- A[i+0]
- A[i+1]
- A[i+2]
- A[i+3]

Look−Ahead score final_order

Look−Ahead score  Look−Ahead score final_order final_order

Lane 0: N/A
Lane 1: N/A
Lane 2: N/A
Lane 3: N/A

slide 11 of 18
http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Multi-Node

Node

A[i+0]
Lane 0

A[i+1]
Lane 1

A[i+2]
Lane 2

A[i+3]
Lane 3

<table>
<thead>
<tr>
<th>Operand Slots</th>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
</tr>
<tr>
<td>0</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;:1</td>
<td>&lt;&lt;:2</td>
<td>&lt;&lt;</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L: N/A</td>
<td>L</td>
<td>L: N/A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1: N/A</td>
<td>1</td>
<td>L: N/A</td>
</tr>
<tr>
<td>3</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;:2</td>
<td>&lt;&lt;:1</td>
<td>&lt;&lt;</td>
</tr>
</tbody>
</table>

FAILED
## Operand Reordering with Look-Ahead

### Diagram Description

- **Look-Ahead Reordering**: Diagram illustrates operand reordering with look-ahead, showing how operands are reordered across lanes.

### Table: Look-Ahead Score and Final Order

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Operand Slots

- **Lane 0**: '1' indicates a slot with a look-ahead score of 2.
- **Lane 1**: '1' indicates a slot with a look-ahead score of 2.
- **Lane 2**: '1' indicates a slot with a look-ahead score of 2.
- **Lane 3**: '1' indicates a slot with a look-ahead score of 2.

### Multi-Node

- **Node**: Diagram shows nodes labeled 'S' indicating possible reordering points.

### Lane Specifics

- **Lane 0**: Final order indicates a successful reordering process.
- **Lane 1**: Final order indicates a successful reordering process.
- **Lane 2**: Final order indicates a successful reordering process.
- **Lane 3**: Final order indicates a successful reordering process.

### Look-Ahead Score

- **Lane 0**: Look-ahead score varies between lanes, indicating different levels of reordering efficiency.
- **Lane 1**: Look-ahead score varies between lanes, indicating different levels of reordering efficiency.
- **Lane 2**: Look-ahead score varies between lanes, indicating different levels of reordering efficiency.
- **Lane 3**: Look-ahead score varies between lanes, indicating different levels of reordering efficiency.

### Failed Case

- **Lane 3**: Indicated with 'FAILED' to highlight an unsuccessful reordering attempt.

---

*slide 11 of 18*
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Operand Reordering with Look-Ahead

Look-Ahead score final_order

Look-Ahead score final_order

Operand Slots

Lane 0 Lane 1 Lane 2 Lane 3

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>&lt; &lt;</td>
<td>&lt; &lt; :1</td>
<td>&lt; &lt; :2</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>L :N/A</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1 :N/A</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt; &lt;</td>
<td>&lt; &lt; :2</td>
<td>&lt; &lt; :1</td>
</tr>
</tbody>
</table>

Slide 11 of 18

http://vporpo.me
Operand Reordering with Look-Ahead

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operand Slots

Slide 11 of 18
http://vporpo.me
Operand Reordering with Look-Ahead

Look-Ahead

Reordering

Operand Slots

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>1</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>2</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>3</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Node

Multi-NODE

Operand Slots

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_order</td>
<td>Look-Ahead score</td>
<td>final_order</td>
<td>Look-Ahead score</td>
</tr>
<tr>
<td>0</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>1</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>2</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>3</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Operands:

- A[i+0]
- A[i+1]
- A[i+2]
- A[i+3]
Score Calculation with Look-Ahead

Lane 0

Lane 1 Candidates

B[i+0]  \[L\] 1

C[i+1]  \[L\] 2

B[i+1]  \[L\] 3

slot 0
## Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>L 1</td>
<td>C[i+1]</td>
<td>L 2</td>
</tr>
<tr>
<td>B[i+1]</td>
<td>L 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lane 0 Candidates: Less than or equal to 1
Lane 1 Candidates: Less than or equal to 2
B[i+1]: Less than or equal to 3
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td>B[i+1]</td>
</tr>
<tr>
<td>![Operand 1]</td>
<td>![Operand 2]</td>
<td>![Operand 3]</td>
</tr>
</tbody>
</table>

Operands:

- B[i+0]
- C[i+1]
- B[i+1]
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th></th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>slot 0</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Lane 0</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[i+0]</td>
<td><img src="lane0" alt="Operand 1" /></td>
<td>C[i+1]</td>
<td><img src="lane1" alt="Operand 2" /></td>
</tr>
<tr>
<td><img src="lane0" alt="Lane 0" /></td>
<td><img src="lane1" alt="Lane 1" /></td>
<td><img src="lane2" alt="Lane 2" /></td>
<td><img src="lane3" alt="Lane 3" /></td>
</tr>
<tr>
<td><strong>Lane 1 Candidates</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[i+1]</td>
<td><img src="lane0" alt="Operand 3" /></td>
<td></td>
<td><img src="lane1" alt="Operand 4" /></td>
</tr>
<tr>
<td><img src="lane0" alt="Lane 0" /></td>
<td><img src="lane1" alt="Lane 1" /></td>
<td><img src="lane2" alt="Lane 2" /></td>
<td><img src="lane3" alt="Lane 3" /></td>
</tr>
</tbody>
</table>

**Operands**

- B[i+0]
- C[i+1]
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td>B[i+1]</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Operand 1" /></td>
<td><img src="image" alt="Operand 2" /></td>
<td><img src="image" alt="Operand 3" /></td>
</tr>
<tr>
<td>Score</td>
<td></td>
<td>Score</td>
<td></td>
</tr>
</tbody>
</table>

Operands:
- B[i+0]
- C[i+1]
- B[i+1]
### Score Calculation with Look-Ahead

#### Slot 0

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td></td>
</tr>
<tr>
<td>B[i+1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Operands

- **B[i+0]**: Lane 0
- **C[i+1]**: Lane 1
- **B[i+1]**: Lane 1

**Not Consecutive**

<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
## Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td>B[i+1]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Score</td>
<td>Score</td>
</tr>
<tr>
<td></td>
<td>Not Consecutive</td>
<td>Different Opcodes</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Slote 0**
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th></th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>slot 0</td>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td>B[i+1]</td>
</tr>
<tr>
<td>Operands</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Score</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Score</td>
</tr>
<tr>
<td></td>
<td>C[i+1]</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not Consecutive
Different Opcodes
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B[i+0]</td>
<td>C[i+1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B[i+1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>C[i+1]</td>
<td>Not Consecutive</td>
</tr>
<tr>
<td></td>
<td>Different Opcodes</td>
</tr>
<tr>
<td></td>
<td>Different Opcodes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

B[i+0] 

C[i+1] 

Different Opcodes
## Score Calculation with Look-Ahead

### Table: Score Calculation

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram: Score Calculation with Look-Ahead

- **Operands**:
  - B[i+0]
  - C[i+1]

- **Scores**:
  - Not Consecutive
  - Different Opcodes
  - Both Constants

- **Scores**:
  - Slide 12 of 18
  - http://vporpo.me
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B[i+0]</td>
<td>C[i+1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Score</td>
</tr>
<tr>
<td>Operands</td>
<td></td>
<td>Not Consecutive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Different Opcodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Different Opcodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both Constants</td>
</tr>
<tr>
<td>Look-Ahead score:</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>slot 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lane 0</strong></td>
</tr>
<tr>
<td><strong>B[i+0]</strong></td>
</tr>
</tbody>
</table>

- **Operands**
  - **B[i+0]**
    - Consecutive: 1
  - **C[i+1]**
    - Not Consecutive: 2
    - Different Opcodes: 0
    - Both Constants: +1
  - **B[i+1]**
    - Consecutive: 3

**Look-Ahead score:** 1
### Score Calculation with Look-Ahead

#### slot 0

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
<th>Lane 0</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td></td>
<td>B[i+1]</td>
<td></td>
</tr>
<tr>
<td>L 1</td>
<td>L 2</td>
<td></td>
<td>L 3</td>
<td></td>
</tr>
</tbody>
</table>

**Operands**

- B[i+0] 1
- C[i+1] 2

**Score Calculation**

- Not Consecutive
- Different Opcodes
- Different Opcodes
- Both Constants

- Look-Ahead score: 1

---

Not Consecutive: 0
Different Opcodes: 0
Both Constants: +1
Consecutive Different Opcodes: 1

---

http://vporpo.me
# Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B[i+0]</strong></td>
<td></td>
<td><strong>B[i+1]</strong></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>C[i+1]</strong></td>
<td>Not Consecutive</td>
<td>Consecutive</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Different Opcodes</td>
<td>Different Opcodes</td>
<td>Different Opcodes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>C[i+1]</strong></td>
<td>Different Opcodes</td>
<td>Both Constants</td>
<td>+1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Both Constants</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Look-Ahead score:** 1
Score Calculation with Look-Ahead

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B[i+0]</td>
<td>C[i+1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Not Consecutive</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[i+1]</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Consecutive</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Different Opcodes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[i+1]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Different Opcodes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[i+1]</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Both Constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Both Constants</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Look-Ahead score: 1
## Score Calculation with Look-Ahead

### slot 0

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[i+1]</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[i+1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Operands

<table>
<thead>
<tr>
<th></th>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[i+1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Look-Ahead score:

- Lane 0: 1
- Lane 1: 2
## Score Calculation with Look-Ahead

### Slot 0

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>Lane 1 Candidates</th>
<th>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[i+0]</td>
<td>C[i+1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[i+1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Operands

- **B[i+0]**: 1
- **C[i+1]**: 2
- **B[i+1]**: 3

#### Score Calculation

- **Not Consecutive**:
  - Score: 0

- **Different Opcodes**:
  - Score: 0

- **Both Constants**:
  - Score: +1

- **Consecutive**:
  - Score: +1

### Look-Ahead Score

- Lane 0: 1
- Lane 1: 2

**Total Look-Ahead Score:** 2
Experimental Setup

- Implemented LSLP in LLVM 4.0
Experimental Setup

- Implemented LSLP in LLVM 4.0
- Target: Intel Core i5-6440HQ Skylake CPU
Experimental Setup

- Implemented LSLP in LLVM 4.0
- Target: Intel Core i5-6440HQ Skylake CPU
- Compiler flags: -O3 -ffast-math -mavx2 -march=skylake -mtune=skylake
Experimental Setup

• Implemented LSLP in LLVM 4.0
• Target: Intel Core i5-6440HQ Skylake CPU
• Compiler flags: -O3 -ffast-math -mavx2
  -march=skylake -mtune=skylake
• Kernels from SPEC CPU2006
• We evaluated the following cases:
Experimental Setup

- Implemented LSLP in LLVM 4.0
- Target: Intel Core i5-6440HQ Skylake CPU
- Compiler flags: -O3 -ffast-math -mavx2 -march=skylake -mtune=skylake
- Kernels from SPEC CPU2006
- We evaluated the following cases:
  1. All loop, SLP and LSLP vectorizers disabled (O3)
Experimental Setup

- Implemented LSLP in LLVM 4.0
- Target: Intel Core i5-6440HQ Skylake CPU
- Compiler flags: -O3 -ffast-math -mavx2 -march=skylake -mtune=skylake
- Kernels from SPEC CPU2006
- We evaluated the following cases:
  1. All loop, SLP and LSLP vectorizers disabled (O3)
  2. O3 + SLP enabled but No Reordering (SLP-NR)
Experimental Setup

- Implemented LSLP in LLVM 4.0
- Target: Intel Core i5-6440HQ Skylake CPU
- Compiler flags: -O3 -ffast-math -mavx2 -march=skylake -mtune=skylake
- Kernels from SPEC CPU2006
- We evaluated the following cases:
  1. All loop, SLP and LSLP vectorizers disabled (O3)
  2. O3 + SLP enabled but No Reordering (SLP-NR)
  3. O3 + SLP enabled (SLP)
Experimental Setup

- Implemented LSLP in LLVM 4.0
- Target: Intel Core i5-6440HQ Skylake CPU
- Compiler flags: -O3 -ffast-math -mavx2 -march=skylake -mtune=skylake
- Kernels from SPEC CPU2006
- We evaluated the following cases:
  1. All loop, SLP and LSLP vectorizers disabled (O3)
  2. O3 + SLP enabled but No Reordering (SLP-NR)
  3. O3 + SLP enabled (SLP)
  4. O3 + LSLP enabled (LSLP)
Performance (normalized to O3)

- Execution Speedup
  - SLP-NR
  - SLP
  - LSLP

Tests:
- 453.boy-surface
- 453.intersect-quadratic
- 453.calc-z3
- 453.vsumsqr
- 453.hreciprocal
- 453.mesh1
- 433.mult-su2-mat
- 453.quartic-cylinder
- GMean

Tests:
- motivation-loads
- motivation-opcodes
- motiv-multi
Static Cost (the higher the better)

-50 -45 -40 -35 -30 -25 -20 -15 -10 -5 0

SLP-NR  SLP  LSLP

453.boy-surface  453.intersect-quadratic  453.calc-z3  453.vsumsqr  453.hreciprocal  433.mult-su2-mat  453.quartic-cylinder  motivation-loads  motivation-opcodes  Mean
Performance (Full Benchmarks)

- About 1% speedup in 453.povray and 435.gromacs
Total Compilation Time

- No significant difference in compilation time
Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations. It is based on:
Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations. It is based on:
  1. Smarter operand reordering with Look-Ahead score
  2. Forming Multi-Nodes of commutative operations and reordering across them
Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations. It is based on:
  1. Smarter operand reordering with Look-Ahead score
  2. Forming Multi-Nodes of commutative operations and reordering across them
- Better at identifying isomorphism
Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations. It is based on:
  1. Smarter operand reordering with Look-Ahead score
  2. Forming Multi-Nodes of commutative operations and reordering across them
- Better at identifying isomorphism
- Implemented in LLVM and evaluated on a real machine
Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations. It is based on:
  1. Smarter operand reordering with Look-Ahead score
  2. Forming Multi-Nodes of commutative operations and reordering across them
- Better at identifying isomorphism
- Implemented in LLVM and evaluated on a real machine
- Improves performance and coverage
Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations. It is based on:
  1. Smarter operand reordering with Look-Ahead score
  2. Forming Multi-Nodes of commutative operations and reordering across them
- Better at identifying isomorphism
- Implemented in LLVM and evaluated on a real machine
- Improves performance and coverage
- Similar compilation time