

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

Vasileios Porpodas¹, **Rodrigo Rocha**²
and Luís Góes³

Intel Santa Clara, USA¹

University of Edinburgh, UK²

PUC Minas, Brazil³

EuroLLVM 2018



THE UNIVERSITY of EDINBURGH
informatics



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

Straight-Line Code Vectorizer

- LLVM and GCC provide SLP
 - Superword Level Parallelism [Larsen PLDI'00]
 - Bottom-Up SLP

Straight-Line Code Vectorizer

- LLVM and GCC provide SLP
 - Superword Level Parallelism [Larsen PLDI'00]
 - Bottom-Up SLP
- SLP and loop-vectorizer complement each other:

Straight-Line Code Vectorizer

- LLVM and GCC provide SLP
 - Superword Level Parallelism [Larsen PLDI'00]
 - Bottom-Up SLP
- SLP and loop-vectorizer complement each other:
 - Even if loop vectorizer fails, SLP could partly succeed

Straight-Line Code Vectorizer

- LLVM and GCC provide SLP
 - Superword Level Parallelism [Larsen PLDI'00]
 - Bottom-Up SLP
- SLP and loop-vectorizer complement each other:
 - Even if loop vectorizer fails, SLP could partly succeed
- Compilers usually run SLP after the Loop Vectorizer

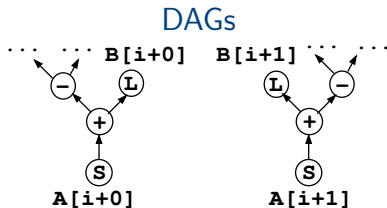
Why commutative operations matter ?

- Operands can be reordered
- It can change the shape of the DAG
- Since SLP operates on DAGs, it affects the SLP's vectorization effectiveness

```

sub1 = ... - ...;
sub2 = ... - ...;
A[i+0] = sub1 + B[i+0];
A[i+1] = B[i+1] + sub2;

```



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 Opcode mismatch further up the graph

SLP not effective in more complex cases

- SLP reordering not effective for:
 - ① Opcode mismatch further up the graph
 - ② Load address mismatch further up the graph

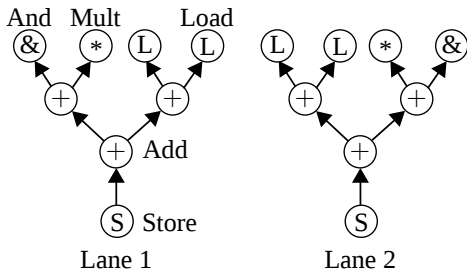
SLP not effective in more complex cases

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

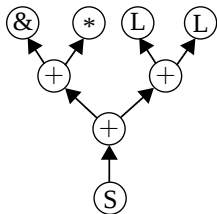
SLP not effective in more complex cases

- SLP reordering not effective for:
 - ① Opcode mismatch further up the graph
 - ② Load address mismatch further up the graph
 - ③ Reordering across chains of commutative operations*
- Look-Ahead SLP (LSLP) provides a solution to all three.

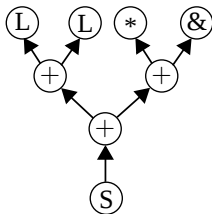
Reordering Chains of Commutative Operations



Reordering Chains of Commutative Operations

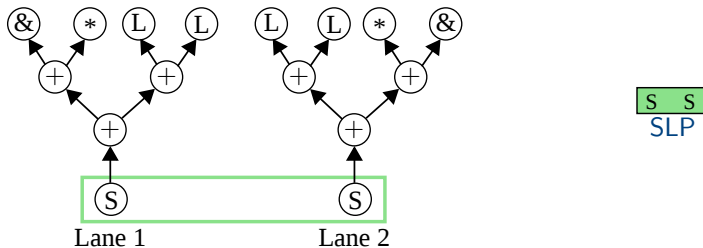


Lane 1



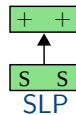
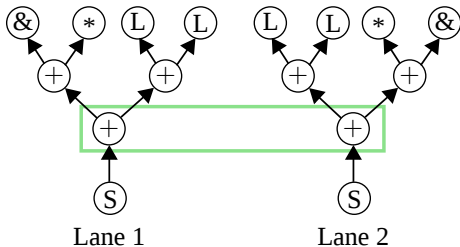
Lane 2

Reordering Chains of Commutative Operations



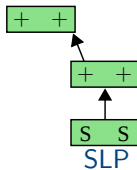
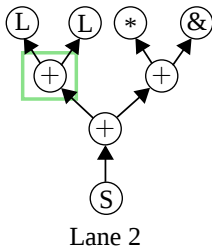
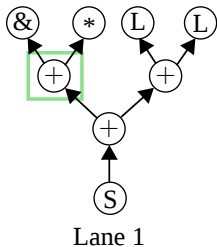
● Non-Vectorizable
 Vectorizable
 +/- #Cost

Reordering Chains of Commutative Operations



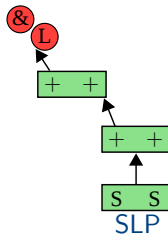
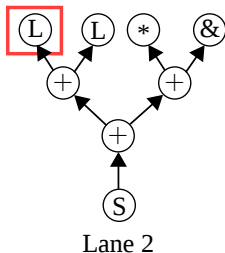
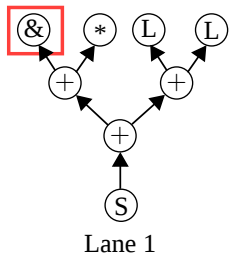
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



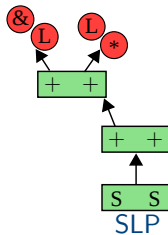
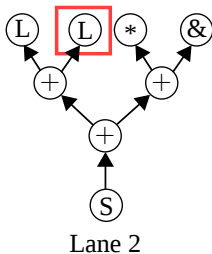
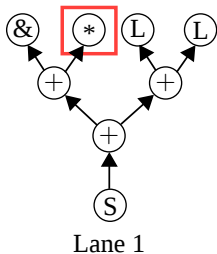
● Non-Vectorizable ■ Vectorizable +/-#Cost

Reordering Chains of Commutative Operations



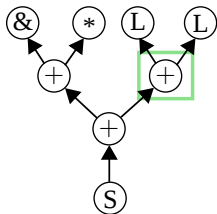
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations

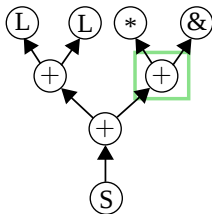


● Non-Vectorizable
 Vectorizable
 +/- #Cost

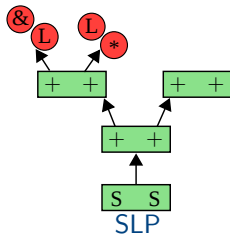
Reordering Chains of Commutative Operations



Lane 1

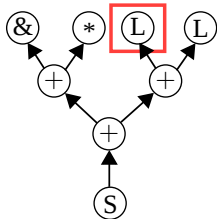


Lane 2

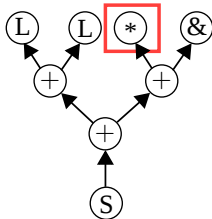


● Non-Vectorizable
 Vectorizable
 +/- #Cost

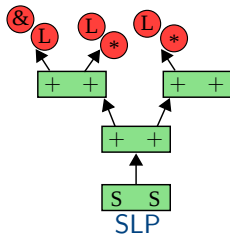
Reordering Chains of Commutative Operations



Lane 1

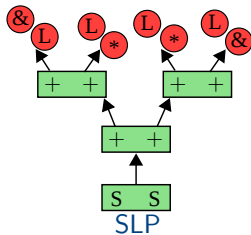
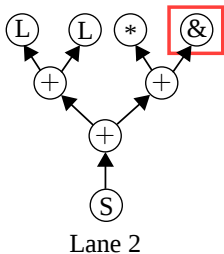
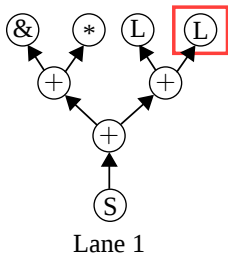


Lane 2



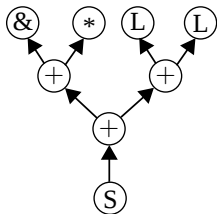
● Non-Vectorizable
 Vectorizable
 +/-#Cost

Reordering Chains of Commutative Operations

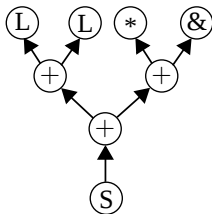


● Non-Vectorizable ■ Vectorizable +/- #Cost

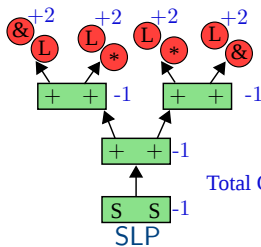
Reordering Chains of Commutative Operations



Lane 1



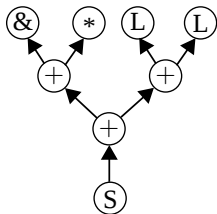
Lane 2



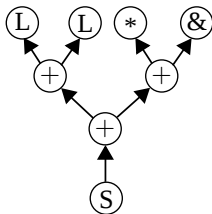
Total Cost = +4

● Non-Vectorizable ■ Vectorizable +/- #Cost

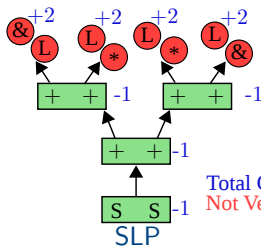
Reordering Chains of Commutative Operations



Lane 1



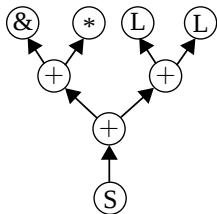
Lane 2



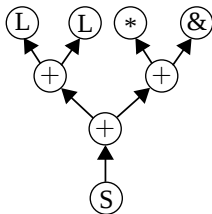
Total Cost = +4
Not Vectorized!

● Non-Vectorizable ■ Vectorizable +/- #Cost

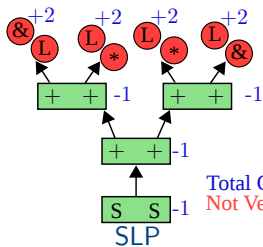
Reordering Chains of Commutative Operations



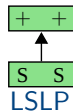
Lane 1



Lane 2

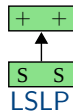
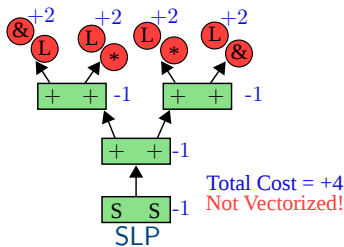
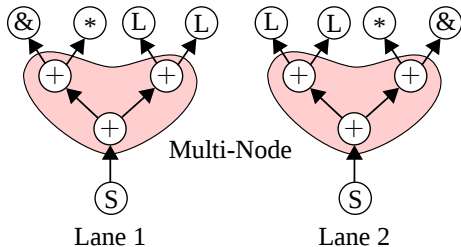


Total Cost = +4
Not Vectorized!



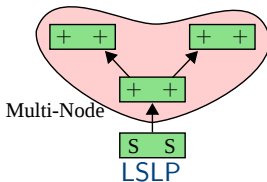
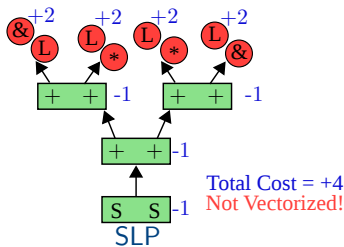
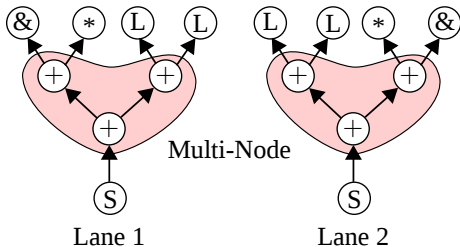
● Non-Vectorizable ■ Vectorizable +/- # Cost

Reordering Chains of Commutative Operations



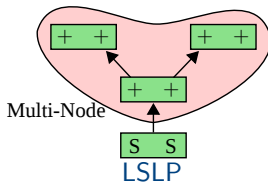
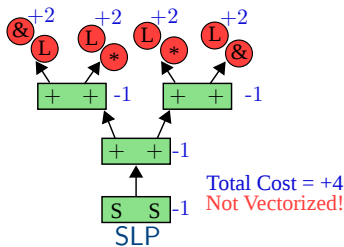
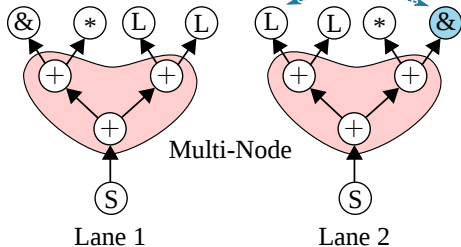
● Non-Vectorizable ■ Vectorizable +/-#Cost

Reordering Chains of Commutative Operations



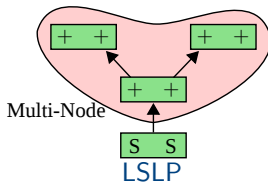
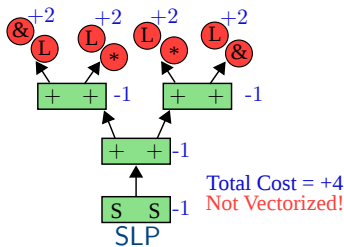
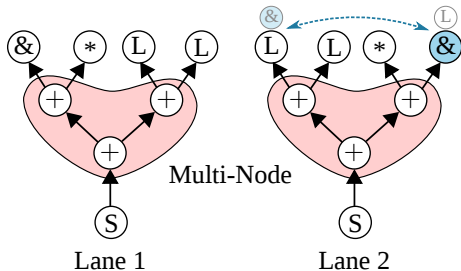
● Non-Vectorizable ■ Vectorizable +/- # Cost

Reordering Chains of Commutative Operations



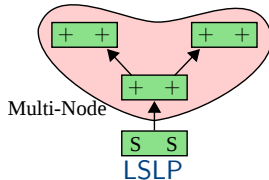
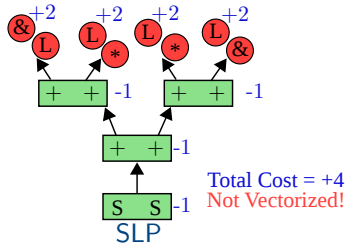
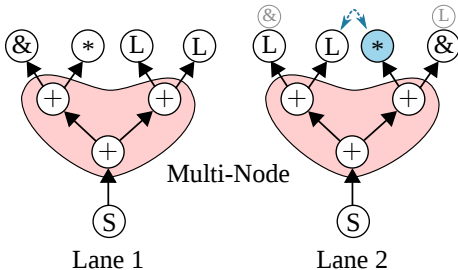
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



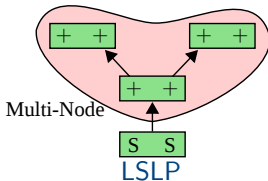
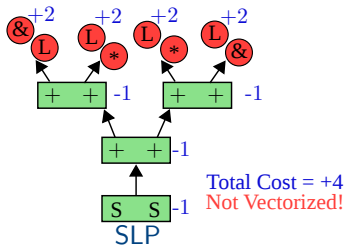
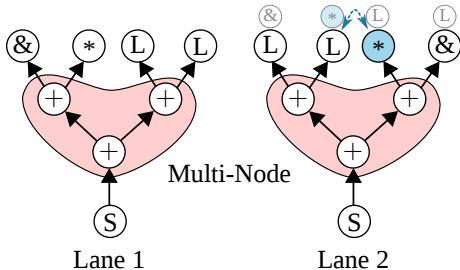
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



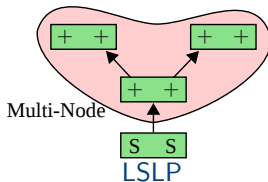
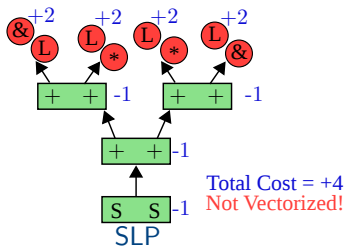
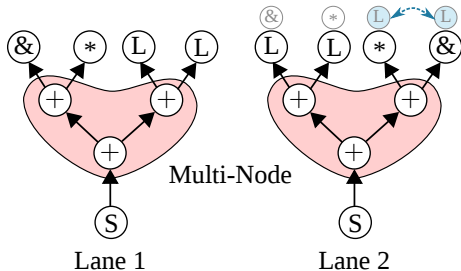
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



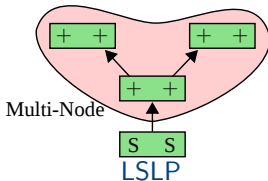
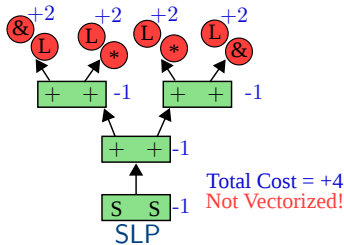
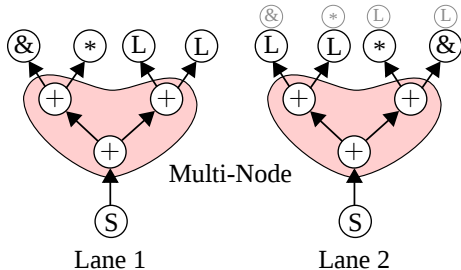
● Non-Vectorizable ■ Vectorizable +/-#Cost

Reordering Chains of Commutative Operations



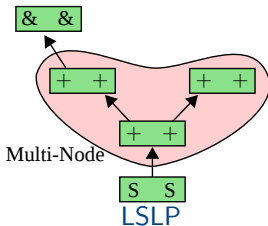
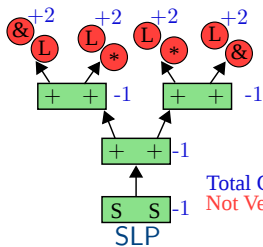
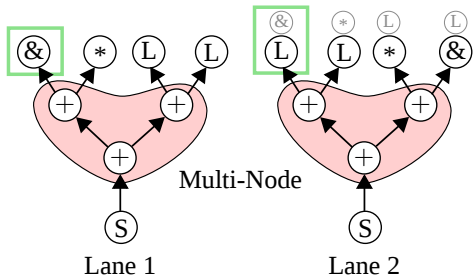
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



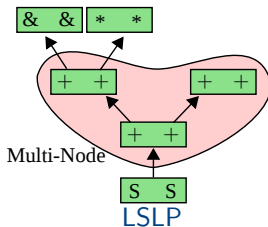
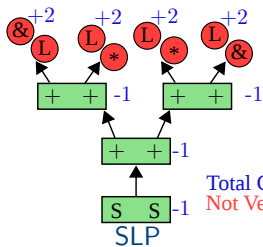
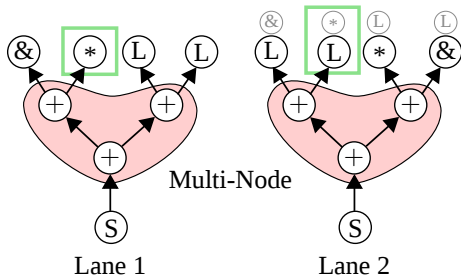
● Non-Vectorizable ■ Vectorizable +/- # Cost

Reordering Chains of Commutative Operations



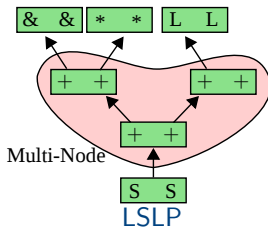
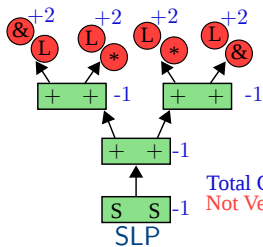
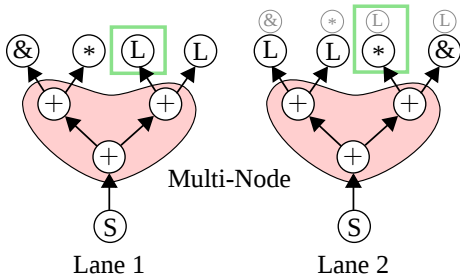
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



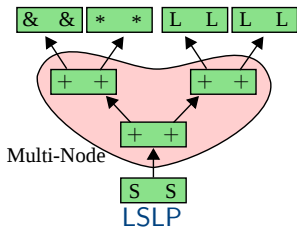
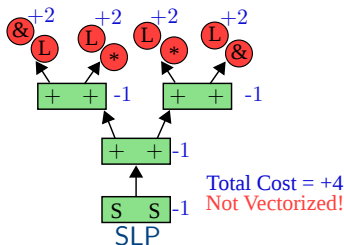
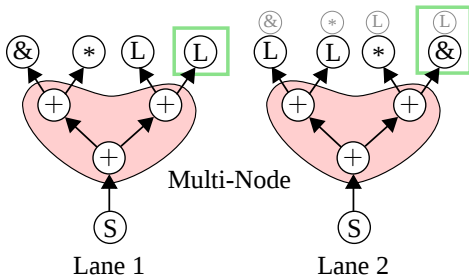
● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



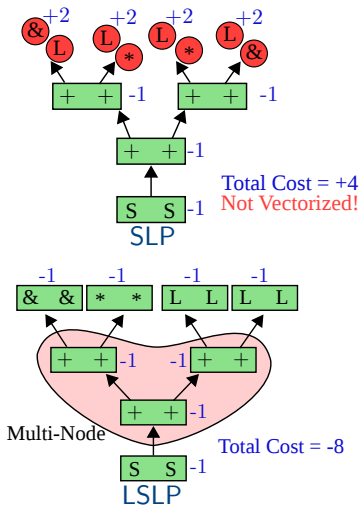
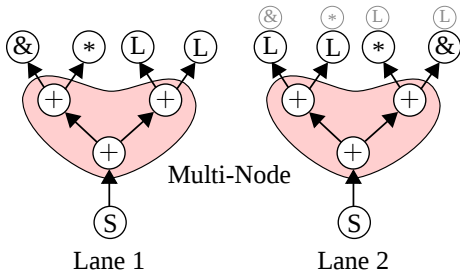
● Non-Vectorizable
 Vectorizable
 +/- # Cost

Reordering Chains of Commutative Operations



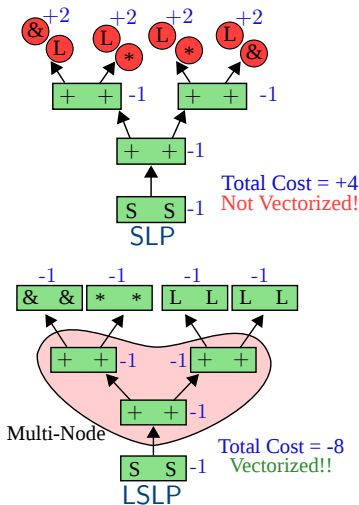
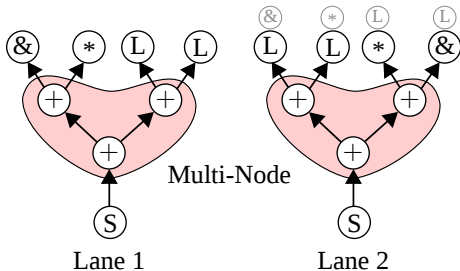
● Non-Vectorizable ■ Vectorizable +/-#Cost

Reordering Chains of Commutative Operations



● Non-Vectorizable ■ Vectorizable +/- #Cost

Reordering Chains of Commutative Operations



● Non-Vectorizable ■ Vectorizable +/- #Cost

Experimental Setup

- Target: Intel Core i5-6440HQ Skylake CPU
- Kernels from SPEC CPU2006
- We evaluated the following cases:

Experimental Setup

- Target: Intel Core i5-6440HQ Skylake CPU
- Kernels from SPEC CPU2006
- We evaluated the following cases:
 - ① All loop, SLP and LSLP vectorizers disabled (O3)

Experimental Setup

- Target: Intel Core i5-6440HQ Skylake CPU
- 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)

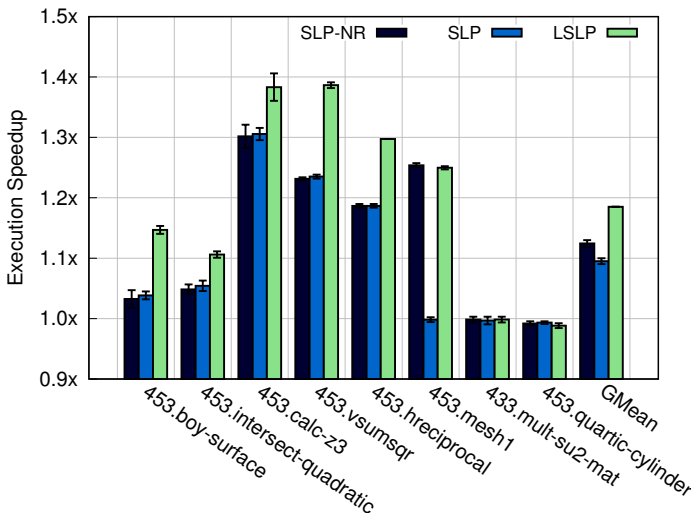
Experimental Setup

- Target: Intel Core i5-6440HQ Skylake CPU
- 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

- Target: Intel Core i5-6440HQ Skylake CPU
- 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)



Conclusion

- LSLP introduces an effective scheme for dealing with commutative operations.
- Look-Ahead SLP: Auto-vectorization in the presence of commutative operations (CGO 2018)