# Improvements to LoopInterchange to Accelerate Vectorization

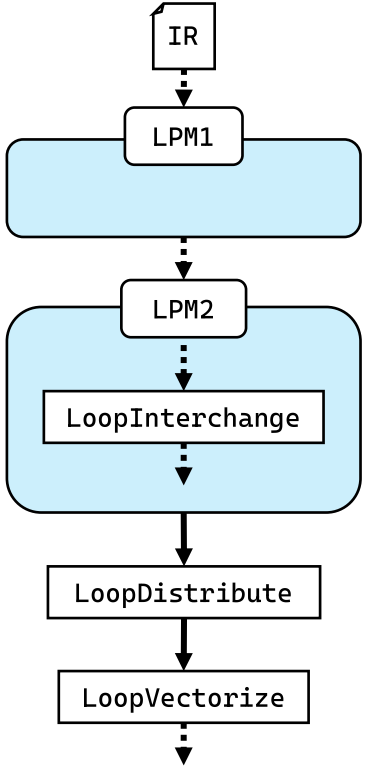
Ryotaro Kasuga (@kasuga-fj)

2025 AsiaLLVM Developers’ Meeting

June 2025

# Background

A part of loop passes pipeline
(LPM: LoopPassManager)



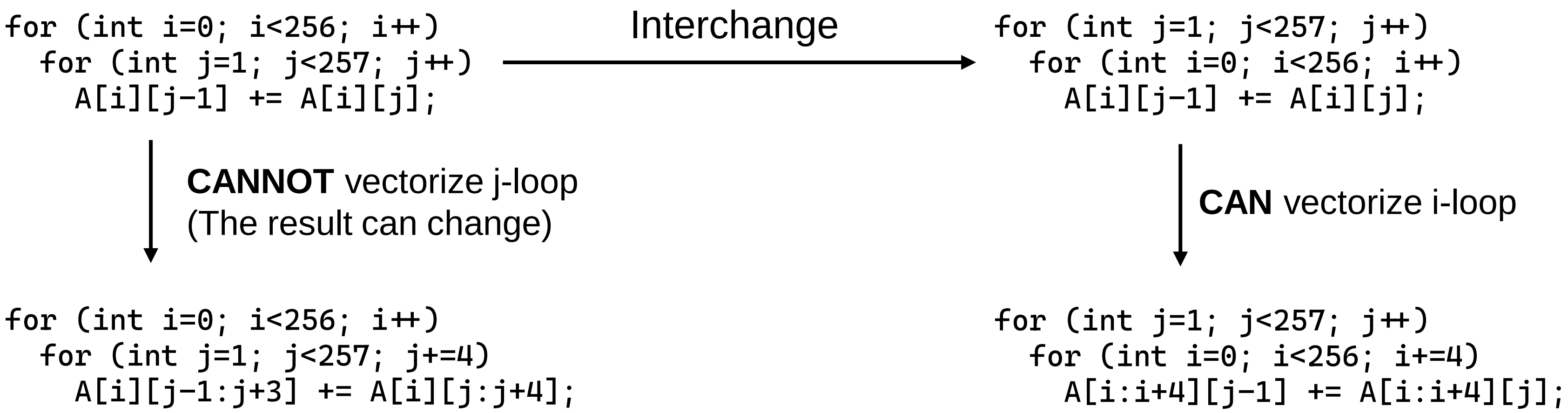
- Some loop transformations can increase opportunities for vectorization
- LoopInterchange is one such transformation
- Recent community activities related to LoopInterchange:
  - 2024 LLVM Dev Mtg - Loop Vectorisation: a quantitative approach to identify/evaluate opportunities
  - Enabling LoopInterchange by default (#124911)
- We've been focusing on the following points:
  - Enhance to interchange more loops
  - Fix correctness issues

Example of Interchange (improve spatial locality)

```
for (int i=0; i<N; i++)            for (int j=0; j<M; j++)
  for (int j=0; j<M; j++)    →      for (int i=0; i<N; i++)
    A[j][i] += B[j][i];                A[j][i] += B[j][i];
```

# Example: LoopInterchange for Vectorization

- **Goal**: Make the innermost loop vectorizable by reordering the loops
- An example inspired by TSVC s231
    - Exchanging the loops makes the innermost loop vectorizable



```
for (int i=0; i<256; i++)
  for (int j=1; j<257; j++)
    A[i][j-1] += A[i][j];
```

Interchange

```
for (int j=1; j<257; j++)
  for (int i=0; i<256; i++)
    A[i][j-1] += A[i][j];
```

**CANNOT** vectorize j-loop
(The result can change)

```
for (int i=0; i<256; i++)
  for (int j=1; j<257; j+=4)
    A[i][j-1:j+3] += A[i][j:j+4];
```

**CAN** vectorize i-loop

```
for (int j=1; j<257; j++)
  for (int i=0; i<256; i+=4)
    A[i:i+4][j-1] += A[i:i+4][j];
```

# Overview of LoopInterchange

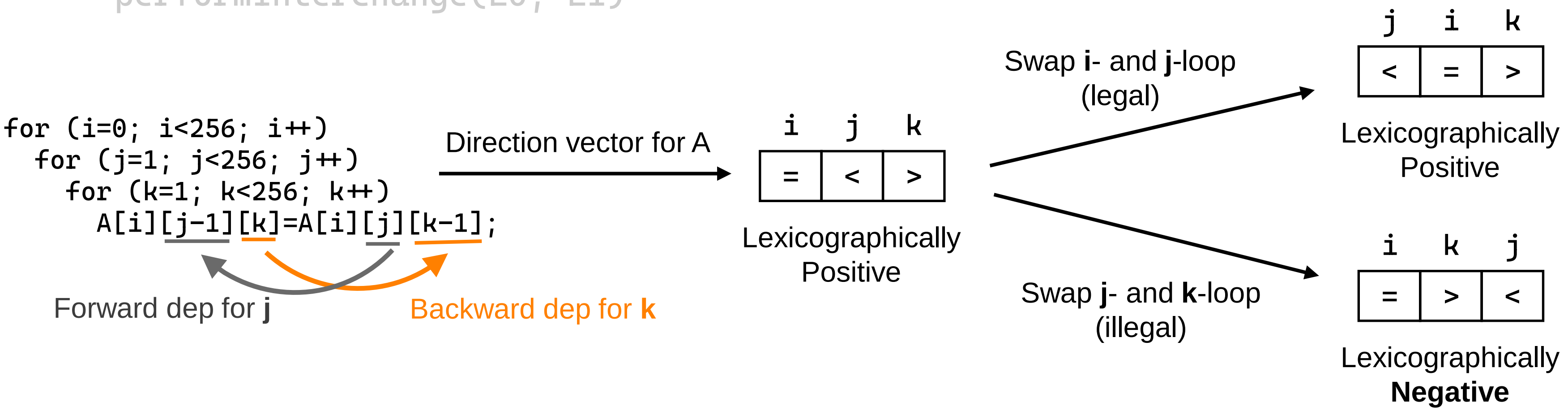
```
interchangeLoops(LoopNest LN)
  for (L0, L1) in candidates
    if NOT isLegal(L0, L1)
      continue
    if NOT isProfitable(L0, L1)
      continue
    performInterchange(L0, L1)
```

- The left shows simplified pseudo code of LoopInterchange
- Roughly consists of three phases
  - Legality check
  - Profitability check
  - Transformation
- We have enhanced Legality check and Profitability check
  - isLegal and isProfitable, respectively

# Legality Check: Example

```
interchangeLoops(LoopNest LN)
  for (L0, L1) in candidates
    if NOT isLegal(L0, L1)
      continue
    if NOT isProfitable(L0, L1)
      continue
    performInterchange(L0, L1)
```

- Check whether exchanging the two loops is legal
    - Using direction vectors
    - Legal if the lexicographic order doesn't change by swapping



```
for (i=0; i<256; i++)
  for (j=1; j<256; j++)
    for (k=1; k<256; k++)
      A[i][j-1][k]=A[i][j][k-1];
```

Forward dep for **j**

Backward dep for **k**

Direction vector for A

| i | j | k |
|---|---|---|
| = | < | > |

Lexicographically Positive

Swap **i**- and **j**-loop (legal)

| j | i | k |
|---|---|---|
| < | = | > |

Lexicographically Positive

Swap **j**- and **k**-loop (illegal)

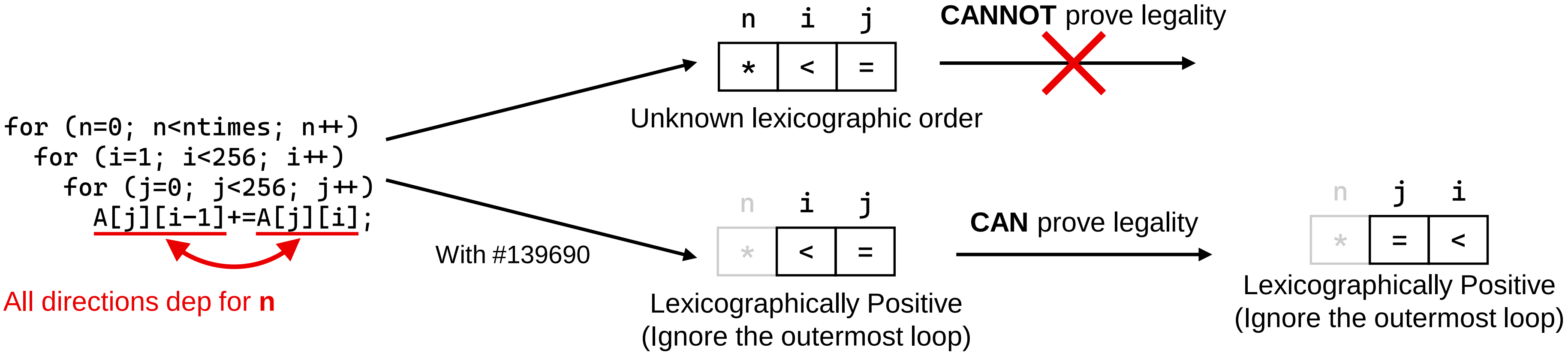
| i | k | j |
|---|---|---|
| = | > | < |

Lexicographically **Negative**

# Legality Check: Improvements

```
interchangeLoops(LoopNest LN)
  for (L0, L1) in candidates
    if NOT isLegal(L0, L1)
      continue
    if NOT isProfitable(L0, L1)
      continue
    performInterchange(L0, L1)
```

- Contributions
  - Relax the legality check: #139690
    - Idea: Ignore the dependencies of surrounding loops
  - Fix corner cases: #124901, #140709



```
for (n=0; n<ntimes; n++)
  for (i=1; i<256; i++)
    for (j=0; j<256; j++)
      A[j][i-1]+=A[j][i];
```

All directions dep for **n**

With #139690

Unknown lexicographic order

**CANNOT** prove legality

Lexicographically Positive
(Ignore the outermost loop)

**CAN** prove legality

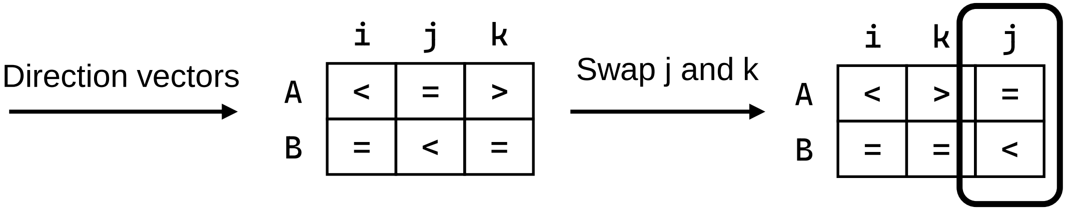
Lexicographically Positive
(Ignore the outermost loop)

# Profitability Check

```
interchangeLoops(LoopNest LN)
  for (L0, L1) in candidates
    if NOT isLegal(L0, L1)
      continue
    if NOT isProfitable(L0, L1)
      continue
    performInterchange(L0, L1)
```

- There are several heuristics for profitability
  - Cache access, Instruction order and Vectorization
- Contributions
  - Option to control heuristics order: #133664
  - Improve the vectorization heuristic: #133667, #133672

```
for (i=1; i<N; i++)
  for (j=1; j<N; j++)
    for (k=1; k<N; k++) {
      A[i-1][j][k]+=A[i][j][k-1];
      B[i][j-1][k]+=B[i][j][k];
    }
```



Direction vectors

|   | i | j | k |
|---|---|---|---|
| A | < | = | > |
| B | = | < | = |

Swap j and k

|   | i | k | j |
|---|---|---|---|
| A | < | > | = |
| B | = | = | < |

Can vectorize j-loop

Exchanging the j-loop and k-loop were always rejected by isProfitable, because:
- The cache profitability was always prioritized
- A loop with only forward dependencies was NOT considered vectorizable

# Experiment

- An example where an interchange prioritizing vectorization improves performance
  - With some patches that are not posted yet
  - Common options: `-O3 -mcpu=neoverse-v2`
  - Interchange options: `-floop-interchange -mllvm -loop-interchange-profitabilities=vectorize`
- About 2x faster by interchanging the loops

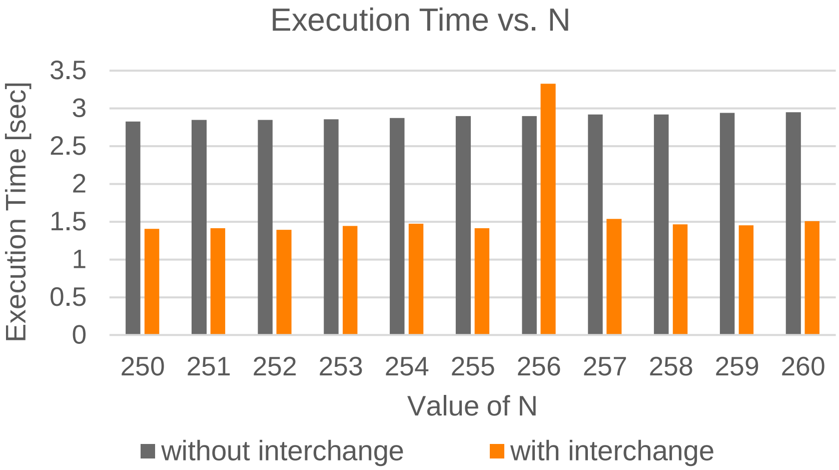
```
float A[N][N],B[N][N],C[N][N];

for (int t=0; t<times; t++)
  for (int i=0; i<N; i++)
    for (int j=1; j<N; j++)
      A[j][i]=A[j-1][i]+B[i][j]+C[i][j];
```

Interchange

```
for (int t=0; t<times; t++)
  for (int j=1; j<N; j++)
    for (int i=0; i<N; i++)
      A[j][i]=A[j-1][i]+B[i][j]+C[i][j];
```

- Can vectorize the innermost loop (i-loop)
- Worse spatial locality of memory accesses

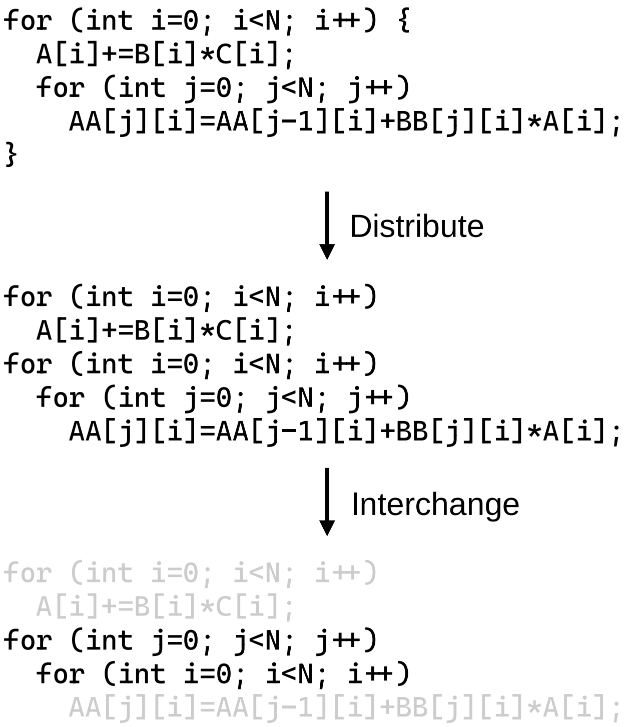


(Cache slashing may occur when N=256)

# Future Work

- Sophisticated vectorization heuristic
  - Target-aware cost model
    - vector-length, gather/scatter cost, etc.
    - cf. #131130
- Coordination with other loop passes
  - For instance, LoopDistribute can increase opportunities for interchange
  - However, in the current pipeline, LoopInterchange runs before LoopDistribute
- Support for more loop patterns
  - Triangular loops, Reduction variables, etc.

Example: TSVC s235



```
for (int i=0; i<N; i++) {
  A[i]+=B[i]*C[i];
  for (int j=0; j<N; j++)
    AA[j][i]=AA[j-1][i]+BB[j][i]*A[i];
}
```

Distribute

```
for (int i=0; i<N; i++)
  A[i]+=B[i]*C[i];
for (int i=0; i<N; i++)
  for (int j=0; j<N; j++)
    AA[j][i]=AA[j-1][i]+BB[j][i]*A[i];
```

Interchange

```
for (int i=0; i<N; i++)
  A[i]+=B[i]*C[i];
for (int j=0; j<N; j++)
  for (int i=0; i<N; i++)
    AA[j][i]=AA[j-1][i]+BB[j][i]*A[i];
```

# Acknowledge

- This presentation is based on results obtained from a project, JPNP21029, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

# Thank you