Unlocking Vectorization Scope: Extensible Vectorization via Unified Dependence Semantics

Shihan Fang
Shanghai Jiao Tong University
fang-account@sjtu.edu.cn

Background & Motivation

- SIMD architectures execute one instruction over multiple data
- **Auto-vectorization**

🌟 Vectorization should be **dependence-centric**

- Loop vectorization

```
for (int i = 0; i < 256; i++)
    a[i] = b[i] + 1;
```

```
for (int i = 0; i < 256; i += 4)
    a [i:i + 3] = b [i:i + 3] + 1;
```

- SLP vectorization

```
b[0] = a[0] + 0;
b[1] = a[1] + 1;
b[2] = a[2] + 2;
b[3] = a[3] + 3;
```

```
b [0:3] = a [0:3] + <0, 1, 2, 3>;
```

```
static std::optional<TargetTransformInfo::Sh
isFixedVectorShuffle(ArrayRef<Value *> VL, S
                     AssumptionCache *AC) {
```

```
/// \returns true if all of the instructions in \p VL are in the same block or
/// false otherwise.
static bool allSameBlock(ArrayRef<Value
```

```
/// InnerLoopVectorizer vectorizes loops which contain only one basic
/// block to a specified vectorization factor (VF).
```

Specific checking algorithms

Control boundaries

...

🌟 **Extensibility** should be a first-class design goal for vectorization.

VIR: Unified Dependence Representation

Dependence-centric.

Encode data & control dependences uniformly

VIR

- A predicated IR that replaces basic blocks with a list of (`Control Predicate`, `Iterator`, `Item`)
  - `Control Predicate`: boolean formula (from PSSA [1]) that guards execution, encoding control.
  - `Iterator`: explicit loop iteration pattern.
  - `Item`: `Instruction` or `Loop` (a nested layer)

```
for (int i = 0; i < LENGTH; i = i + 2)
  dst[i] = src[i] + 1;
```

```
[CP]: (bool true)
-> int length = load ptr @LENGTH
[CP]: (bool true)
-> cmp bool loop_cond = 0 < int length
[CP]: (bool loop_cond)
-> with
     int {iter} = iota( 0, {iter_rec}, + 2)
   do
     [CP]: (bool true) | [0, int length) + 2
     -> int load_value = load ptr <@src[{iter}]>
     [CP]: (bool true) | [0, int length) + 2
     -> int store_value = int load_value + 1
     [CP]: (bool true) | [0, int length) + 2
     -> store int store_value ptr <@dst[{iter}]>
     [CP]: (bool true) | [0, int length) + 2
     -> {iter_rec} = {iter} + 2
     [CP]: (bool true) | [0, int length) + 2
     -> cmp bool latch = {iter_rec} < int length
   while [CP]: (bool latch)
```

An Entry

Control Predicate

Iterator

Instruction Item

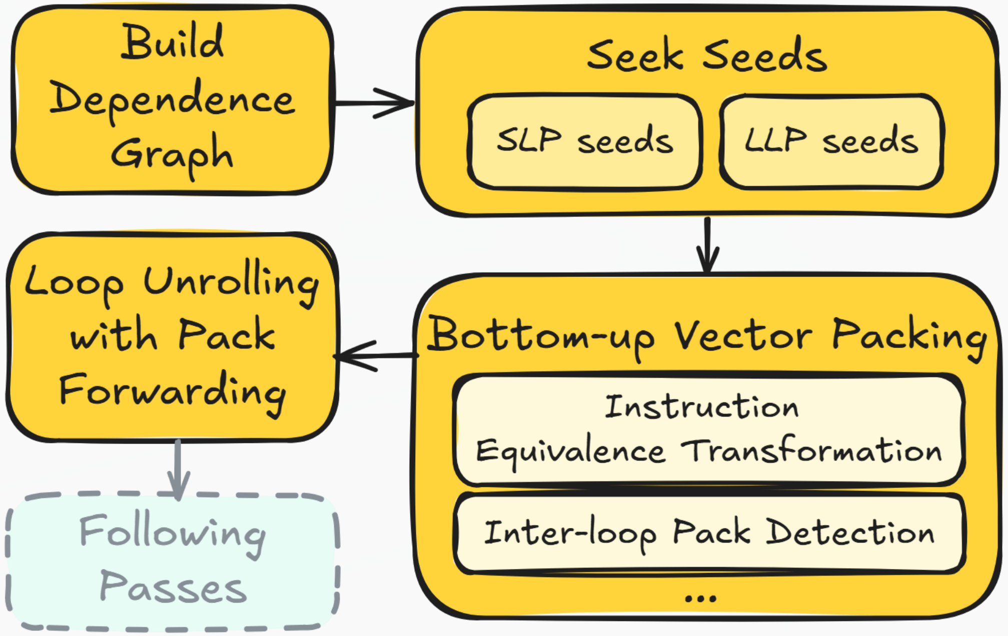
Loop Item

[1] Chen, Yishen, Charith Mendis, and Saman Amarasinghe. "All you need is superword-level parallelism: systematic control-flow vectorization with SLP." In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 301-315. 2022.

Dependence-centric Framework

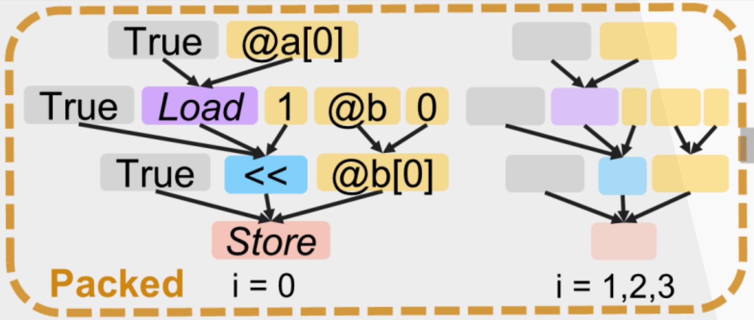
Dependence-centric.
Vectorization as generic traversals over *Dependence Graph*

- Hierarchical, tree-like representation on VIR
- Structured dependence relationship
- Vectorization is performed by traversing and matching structure on it.




```
b[0] = a[0]<<1; b[1] = a[1]<<1; b[2] = a[2]<<2; b[3] = a[3]<<3;
```

```
[CP]: (bool true)
->int x = load ptr <@a[i]>
[CP]: (bool true)
->int x_ = int x << i
[CP]: (bool true)
->store int x_ ptr <@b[i]>
```



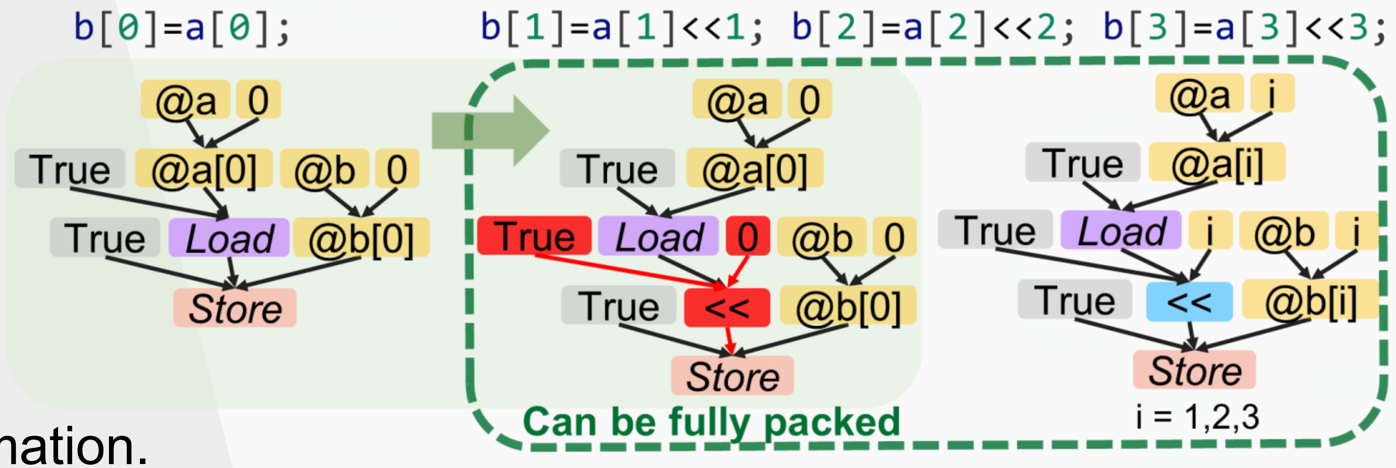
Vectorized VIR

```
[CP]: (bool true)
-> vec(4) v = load ptr <@a[0]>
[CP]: (bool true)
-> vec(4) c_v = vec(4) v << vec<1,1,2,3>
[CP]: (bool true)
-> store vec(4) c_v ptr <@b[0]>
```

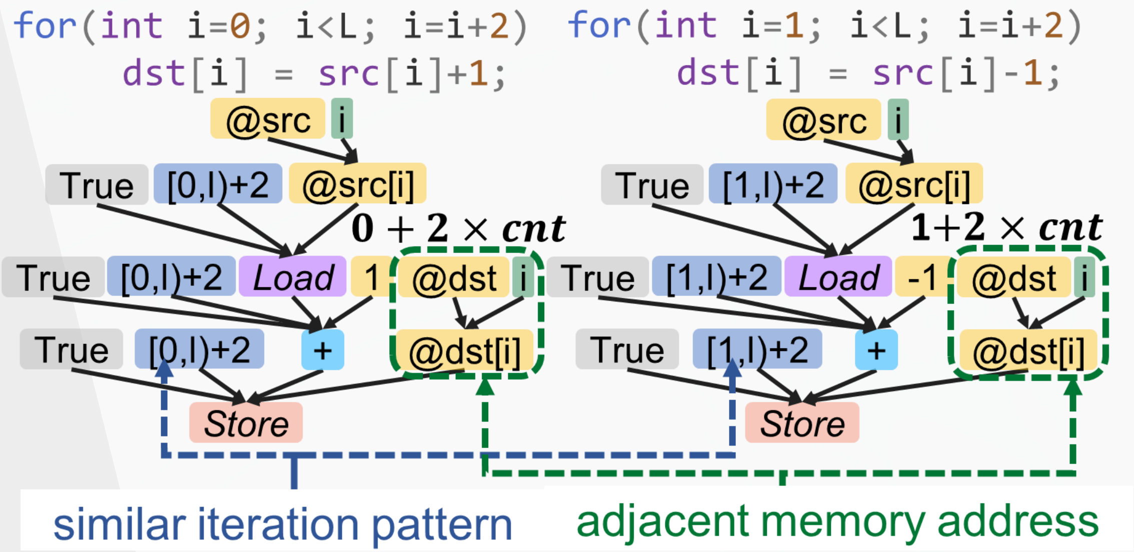
Dependence-centric Framework

Extensibility.

Enable new strategies without redesigning the framework



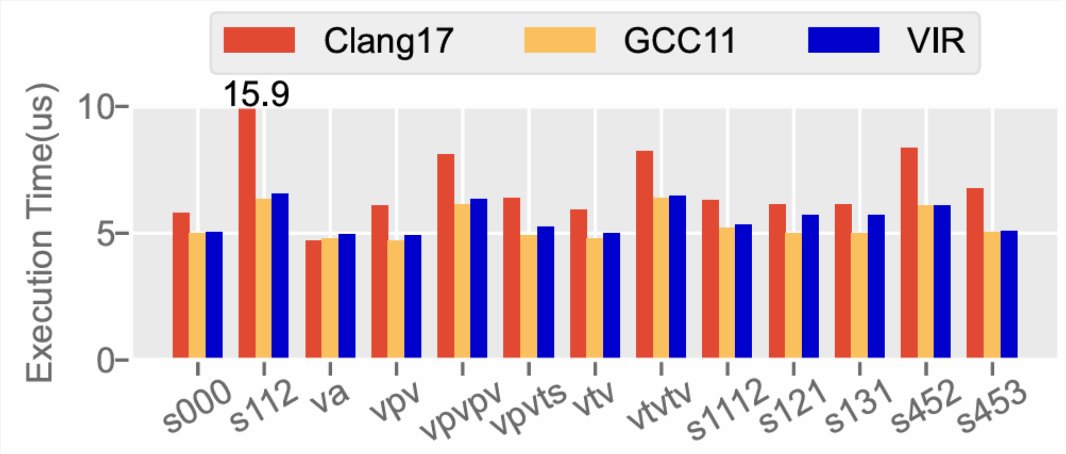
- Instruction Equivalence Transformation.
- Inter-loop Pack Detection.
- …



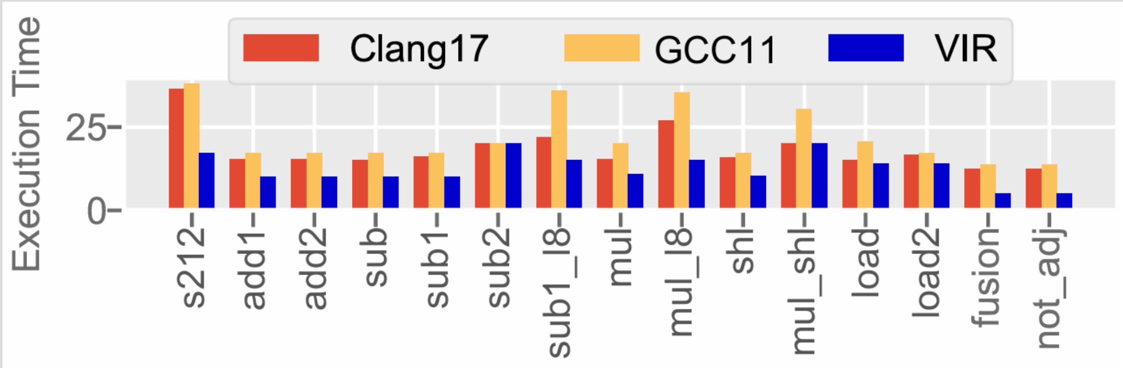
Experiments

Compile to x86 AVX2 (`-mavx2 -O3`).
Ran on an Intel Ultra 7 PC (96GB RAM) with Turbo Boost off.

- Test cases from `tsvc`



- Test cases for image pixel processing




```
color[0] = (hexValue >> 24) & 255;
color[1] = (hexValue >> 16) & 255;
color[2] = (hexValue >> 8) & 255;
color[3] = hexValue & 255;
```

(a) Source Code

```
%1 = load i32, ptr @hexValue, align 4
%2 = lshr i32 %1, 24
store i32 %2, ptr @color, align 16
%3 = lshr i32 %1, 16
%4 = and i32 %3, 255
%5 = getelementptr [4 x i32], ptr @color,i64 0,i64 1
store i32 %4, ptr %5, align 4
%6 = lshr i32 %1, 8
%7 = and i32 %6, 255
%8 = getelementptr [4 x i32], ptr @color,i64 0,i64 2
store i32 %7, ptr %8, align 8
%9 = and i32 %1, 255
%10 = getelementptr [4 x i32], ptr @color,i64 0,i64 3
store i32 %9, ptr %10, align 4
```

Clang17

```
%1 = load i32, ptr @hexValue, align 4
%2 = insertelement <4 x i32>
     <i32 0, i32 0, i32 0, i32 0>, i32 %1, i32 0
%3 = insertelement <4 x i32> %2, i32 %1, i32 1
%4 = insertelement <4 x i32> %3, i32 %1, i32 2
%5 = insertelement <4 x i32> %4, i32 %1, i32 3
%6 = ashr <4 x i32> %5, <i32 24,i32 16,i32 8,i32 0>
%7 = and <4xi32> %6,<i32 255,i32 255,i32 255,i32 255>
%8 = getelementptr [4 x i32], ptr @color,i32 0,i32 0
store <4 x i32> %7, ptr %8, align 4
```

VIR

(b) LLVM IR Generated by Clang 17 and VIR

Future Work

We plan to

- Integrate the framework into the LLVM toolchain.
- Extend vectorization strategies.
- Improve scalability.