Vectorization in the Presence of Control Flow

Josh Clune and Thomas Talbot

Vectorization Techniques

- **Loop Vectorization** - leverages parallelism of inner loop operations.

```
for (i =0; i<n; i++){
   a[i] = b[i] + c[i]
}
```

```
ld r1, addr1
ld r2, addr2
add r3, r1, r2
st r3, addr3
```

```
ldv vr1, addr1
ldv vr2, addr2
vadd vr3, vr1, vr2
stv vr3, addr3
```

- **SLP Vectorization** - groups together sets of independent, isomorphic within the same basic block.

```
(1)   b = a[i+0]
(2)   c = 5
(3)   d= b+c
(4)   e = a[i+1]
(5)   f=6
(6)   g = e+f
```

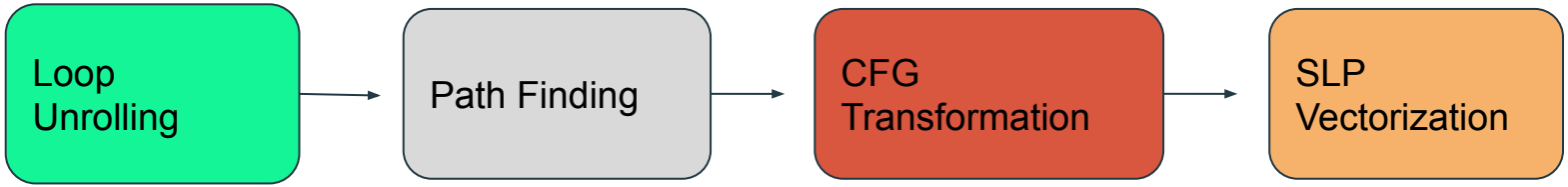
```
b = a[i+0]
e = a[i+1]
```

```
c = 5
f = 6
```

```
d = b + c
g = e + f
```

Overview

- Neither Loop vectorization nor SLP can directly reason about control flow. This inhibits vectorizers from taking advantage of instruction-level parallelism that spans multiple basic blocks.
- We designed and implemented a sequence of four compiler optimization passes in the LLVM framework to perform vectorization across basic blocks.



Example

```
// Original loop
bool b = ...;
int i = 0;
while(i < 100) {
  if(b) {
    inst1;
  }
  else {
    inst2;
  }
  i++;
}
```

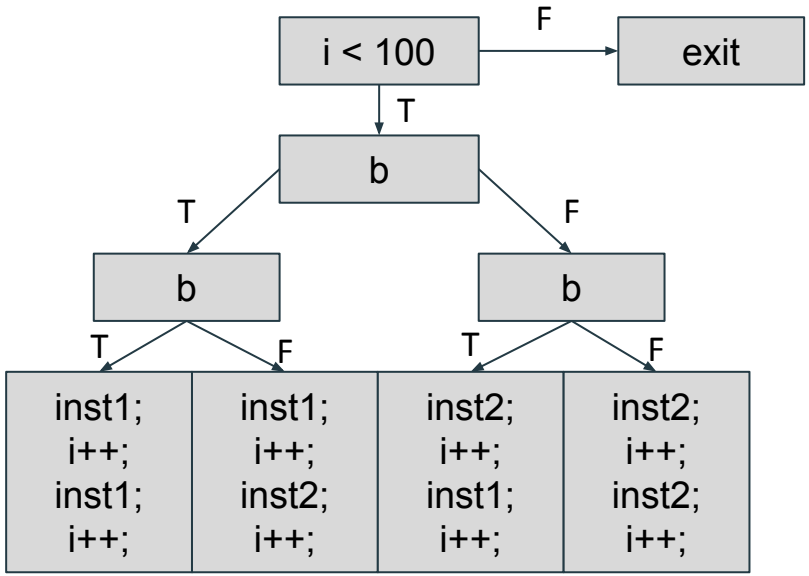
Stage 1: Unrolling

```
// After unrolling (count = 2)
bool b = ...;
int i = 0;
while(i < 100) {
  if(b) {
    inst1;
  }
  else {
    inst2;
  }
  i++;
  if(b) {
    inst1;
  }
  else {
    inst2;
  }
  i++;
}
```

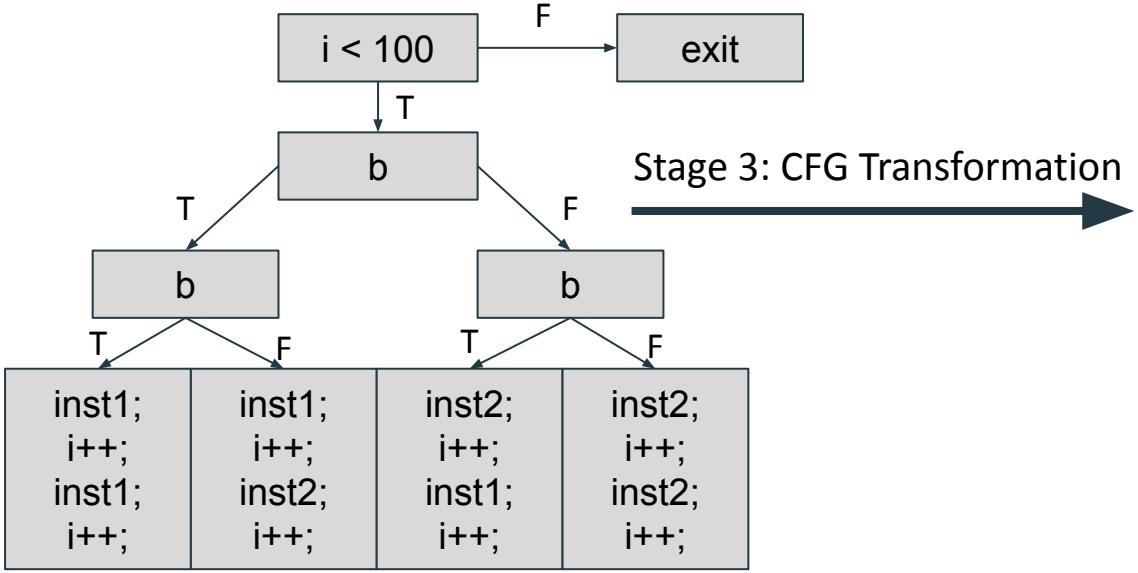
Example

```
// After unrolling (count = 2)
bool b = ...;
int i = 0;
while(i < 100) {
  if(b) {
    inst1;
  }
  else {
    inst2;
  }
  i++;
  if(b) {
    inst1;
  }
  else {
    inst2;
  }
  i++;
}
```

Stage 2: Pathfinding

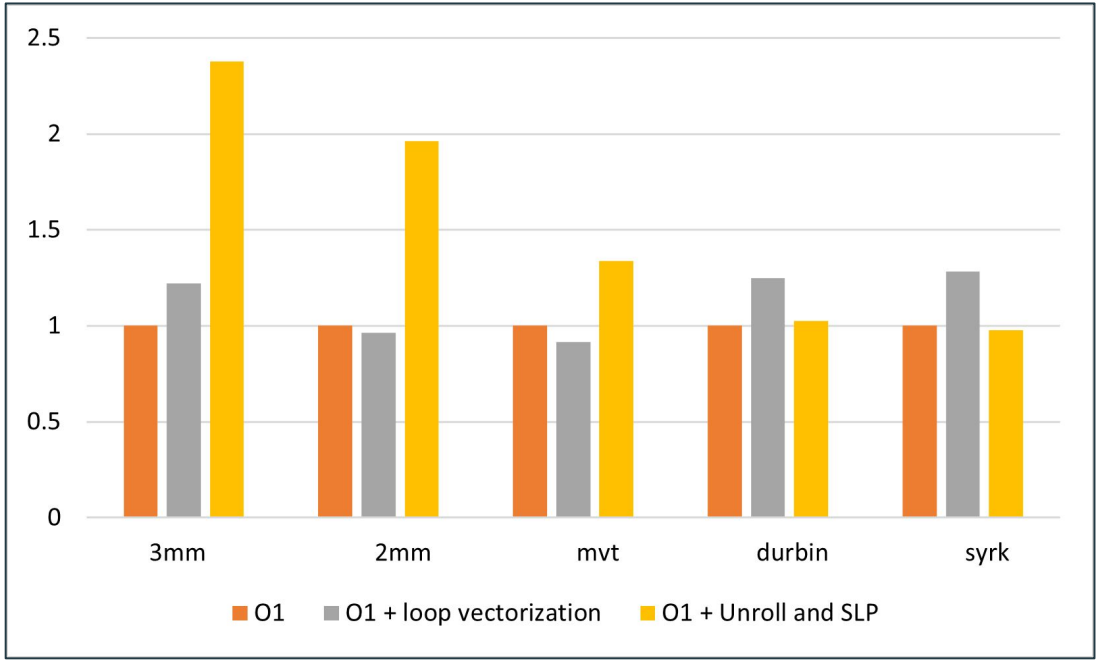


Example




```
// After our CFG transformation
bool b = ...;
int i = 0;
if(i < 100) {
  if(b) {
    if(b) {
      do {
        inst1;
        i++;
        inst1;
        i++;
      } while(i < 100);
    }
    else {
      do {
        inst1;
        i++;
        inst2;
        i++;
      } while(i < 100);
    }
  }
  else {
    if(b) {
      do {
        inst2;
        i++;
        inst1;
        i++;
      } while(i < 100);
    }
    else {
      do {
        inst2;
        i++;
        inst2;
        i++;
      } while(i < 100);
    }
  }
}
```

Experiments



Benchmark	Description
3mm	Three matrix multiplications
2mm	Two matrix multiplications
mvt	Matrix vector product and transpose
durbin	Toeplitz system solver
syrk	Symmetric rank-k operations

Results

- The entire vectorization pipeline (including path finding + CFG transformation) is only effective in a handful of test cases.
- We benchmarked a subset of our vectorization pipeline (unrolling + SLP) with LLVM's loop vectorizer.
- Unrolling + SLP had a significant performance improvement (2x speedup) on benchmarks with outer loop vectorization.
- Loop vectorization outperformed unroll + SLP on other benchmarks; confirming our expectation that neither vectorization strategy is superior.