Global SLP Vectorization for (almost) arbitrary control flow

Yishen Chen, Charith Mendis, Saman Amarasinghe

The COMMIT Compiler Group

Background: Loop Vectorization

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

** Randy Allen and Ken Kennedy. 1987. Automatic Translation of FORTRAN Programs to Vector Form. TOPLAS*

Background: Loop Vectorization

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

** Randy Allen and Ken Kennedy. 1987. Automatic Translation of FORTRAN Programs to Vector Form. TOPLAS*

Background: SLP Vectorization

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

Background: SLP Vectorization

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

** Samuel Larsen and Saman Amarasinghe. 2000. Exploiting Superword Level Parallelism with Multimedia Instruction Sets. In PLDI*

Background: SLP Vectorization

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tb2  = b[1]
tc   = c[0]
tc2  = c[1]
t    = tb + tc
t2   = tb2 + tc2
a[0] = t
a[1] = t2
```

** Samuel Larsen and Saman Amarasinghe. 2000. Exploiting Superword Level Parallelism with Multimedia Instruction Sets. In PLDI*

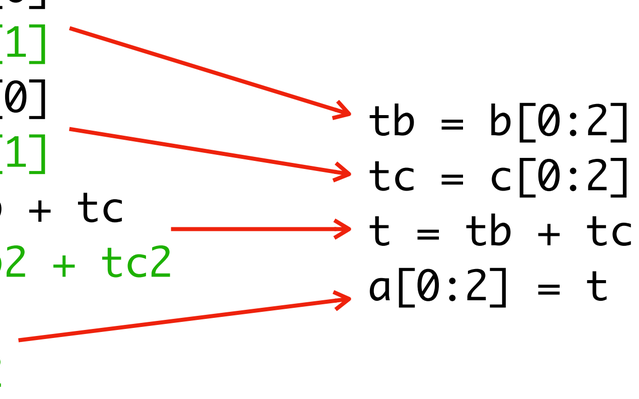
Background: SLP Vectorization

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tb2  = b[1]
tc   = c[0]
tc2  = c[1]
t    = tb + tc
t2   = tb2 + tc2
a[0] = t
a[1] = t2
```



```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

** Samuel Larsen and Saman Amarasinghe. 2000. Exploiting Superword Level Parallelism with Multimedia Instruction Sets. In PLDI*

Background: SLP Vectorization

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

SLP Vectorization

- Simple: no loop dependence analysis
- Flexible: more than just loop-level parallelism

Motivation: Exploiting ILP among different loops

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

SLP Vectorization

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Motivation: Exploiting ILP among different loops

No loop-level parallelism!

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

→

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

→

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

SLP Vectorization

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Motivation: Exploiting ILP among different loops

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

→

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

→

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

SLP Vectorization

Theoretically, yes! Practically, no.

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Requires global code motion + CFG restructuring

Motivation: Exploiting ILP among different loops

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

→

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

→

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

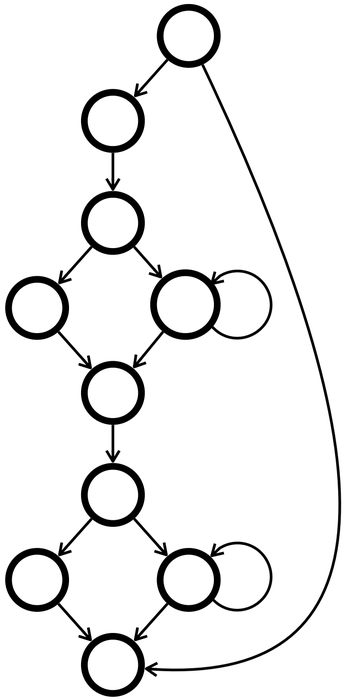
SLP Vectorization

Theoretically, yes! Practically, no.

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Requires global code motion + CFG restructuring



Input CFG

Motivation: Exploiting ILP among different loops

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

→

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

→

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

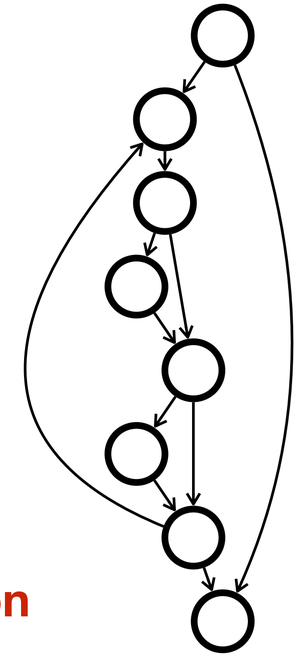
SLP Vectorization

Theoretically, yes! Practically, no.

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Requires global code motion + CFG restructuring



Store and Compare Vectorized

Motivation: Exploiting ILP among different loops

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

→

```
for (i : 0…n step by 2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

Loop Vectorization

```
tb   = b[0]
tc   = c[0]
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

→

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

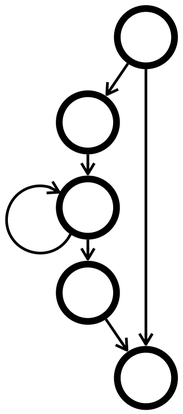
SLP Vectorization

Theoretically, yes! Practically, no.

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Requires global code motion + CFG restructuring



Fully Vectorized

Motivation: Exploiting ILP among different loops

```
for (i : 0…n) {
  tb   = b[i];
  tc   = c[i];
  t    = tb + tc
  a[i] = t;
}
```

```
for (i : 0…n/2) {
  tb   = b[i:i+2];
  tc   = c[i:i+2];
  t    = tb + tc
  a[i:i+2] = t;
}
```

```
t    = tb + tc
a[0] = t
tb2  = b[1]
tc2  = c[1]
t2   = tb2 + tc2
a[1] = t2
```

```
tb = b[0:2]
tc = c[0:2]
t = tb + tc
a[0:2] = t
```

SLP Vectorization

Theoretically, yes! Practically, no.

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```

Requires global code motion + CFG restructuring

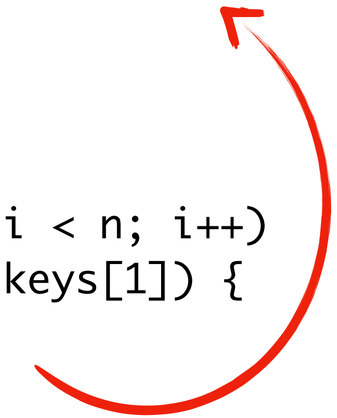
Fully Vectorized

Contribution: Extend SLP vectorization to arbitrary (reducible) control flow

Idea: an IR that makes code motion trivial

```
for (int i = 0; i < n; i++)
  if (arr[i] == keys[0]) {
    idxs[0] = i;
    break;
  }

for (int i = 0; i < n; i++)
  if (arr[i] == keys[1]) {
    idxs[1] = i;
    break;
  }
```



Requires global code motion + CFG restructuring

```
with i = mu(0, i') do
  x = arr[i]                  : true
  k = keys[0]                 : true
  found = cmp eq t, k         : true
  i' = add i, 1               : true
  lt_n = cmp lt i', n         : true
while not found and lt_n      : true
idxs[0] = i                   : found
with i2 = mu(0, i2') do
  x2 = arr[i2]                : true
  k2 = keys[1]                : true
  found2 = cmp eq t2, k2      : true
  i2' = add i2, 1             : true
  lt_n2 = cmp lt i2', n       : true
while not found2 and lt_n2    : true
idxs[1] = i2                  : found2
```

```
with i = mu(0, i') do
  x = arr[i]               : true
  k = keys[0]              : true
  found = cmp eq t, k      : true
  i' = add i, 1            : true
  lt_n = cmp lt i', n      : true
while not found and lt_n   : true
idxs[0] = i                : found
with i2 = mu(0, i2') do
  x2 = arr[i2]             : true
  k2 = keys[1]             : true
  found2 = cmp eq t2, k2   : true
  i2' = add i2, 1          : true
  lt_n2 = cmp lt i2', n    : true
while not found2 and lt_n2 : true
idxs[1] = i2               : found2
```

Convert to Predicated SSA

```
with i = mu(0, i') do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i' = add i, 1             : true
  lt_n = cmp lt i', n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2') do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2' = add i2, 1           : true
  lt_n2 = cmp lt i2', n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i') do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i' = add i, 1             : true
  lt_n = cmp lt i', n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2') do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2' = add i2, 1           : true
  lt_n2 = cmp lt i2', n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Scheduling

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                    …
  x = arr[i]                : true
  x2 = arr[i2]              : true
  k = keys[0]               : true
  k2 = keys[1]              : true
  found = cmp eq t, k       : true
  found2 = cmp eq t2, k2    : true
  i’ = add i, 1             : true
  i2’ = add i2, 1           : true

                    …
while not found and lt_n    : true
idxs[0] = i                 : found
idxs[1] = i2                : found2
```

Scheduling

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                    …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                    …
while active1 or active2    : true
idxs[0] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Scheduling

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
idxs[0] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Scheduling

```
with i = mu(0, i’)
                …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq x, k       : active1
  found2 = cmp eq x2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
idxs[0] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Vector Code Generation

```
with i = mu(0, i') do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i' = add i, 1             : true
  lt_n = cmp lt i', n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2') do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2' = add i2, 1           : true
  lt_n2 = cmp lt i2', n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i')
                  …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                  …

while active1 or active2    : true
idxs[0] = i                 : found_out'
idxs[1] = i2                : found_out2'
```

Scheduling

```
with i = mu(0, i')
                  …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                  …

while active1 or active2    : true
masked-vstore idxs, i_out', found_out': true
```

Vector Code Generation

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
idxs[1] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Scheduling

```
with i = mu(0, i’)
                …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
masked-vstore idxs, i_out’, found_out’: true
```

Vector Code Generation

Predicated SSA

```
if (c)
  x = f();
else
  x = g();
```

```
x1 = f() : c
x2 = g() : not c
x  = phi(c: x1, not c: x2) : true
```

ϕ operands labelled by predicates (not basic blocks)

Predicated SSA

```
if (c)
  x = f();
else
  x = g();
```

```
x1 = f() : c
x2 = g() : not c
x  = phi(c: x1, not c: x2) : true
```

```
while (cont())
  if (exit())
    break;
```

```
c = cont()
do
  ext = exit() : true
  c2 = cont()  : not ext
while not ext and c2 : c
```

Loop predicate

Continue predicate

Predicated SSA

```
if (c)
  x = f();
else
  x = g();
```

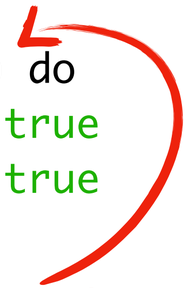
```
x1 = f() : c
x2 = g() : not c
x  = phi(c: x1, not c: x2) : true
```

```
while (cont())
  if (exit())
    break;
```

```
c = cont()
do
  ext = exit() : true
  c2 = cont()  : not ext
while not ext and c2 : c
```

```
for (i : 0…n)
  f(i);
```

```
with i = mu(0, i') do
  f(i)           : true
  i' = add i, 1  : true
while lt_n : true
```



Recursive ϕ nodes become μ nodes

** Karl J. Ottenstein, Robert A. Ballance, and Arthur B. MacCabe. 1990. The Program Dependence Web: A Representation Supporting Control-, Data-, and Demand-Driven Interpretation of Imperative Languages. In PLDI.*

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
idxs[1] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Scheduling

```
with i = mu(0, i’)
                …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
masked-vstore idxs, i_out’, found_out’: true
```

Vector Code Generation

```
with i = mu(0, i') do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i' = add i, 1             : true
  lt_n = cmp lt i', n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2') do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2' = add i2, 1           : true
  lt_n2 = cmp lt i2', n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i')
                    …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                    …
while active1 or active2    : true
idxs[1] = i                 : found_out'
idxs[1] = i2                : found_out2'
```

Scheduling

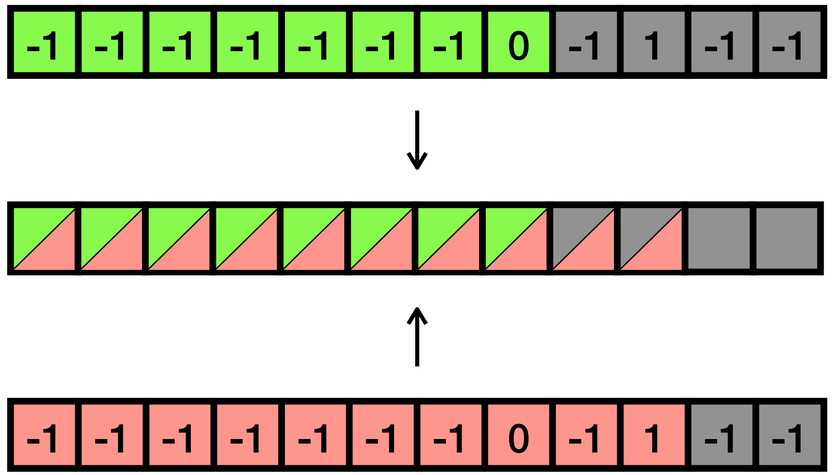
```
with i = mu(0, i')
                    …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                    …
while active1 or active2    : true
masked-vstore idxs, i_out', found_out': true
```

Vector Code Generation

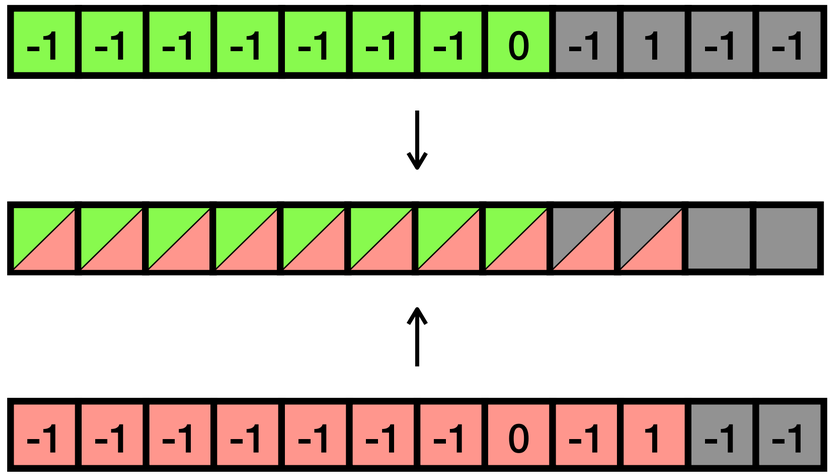
Scheduling

```
with i = mu(0, i')
do
   x = a[i]                 : true
   found = cmp eq x, 0      : true
   i' = i + 1               : true
while not found             : true
idxs[0] = i                 : true
with i2 = mu(0, i2')
do
   y = a[i2]                : true
   found2 = cmp eq y, 1     : true
   i2' = i2 + 1             : true
while not found2            : true
idxs[1] = i2                : true
```



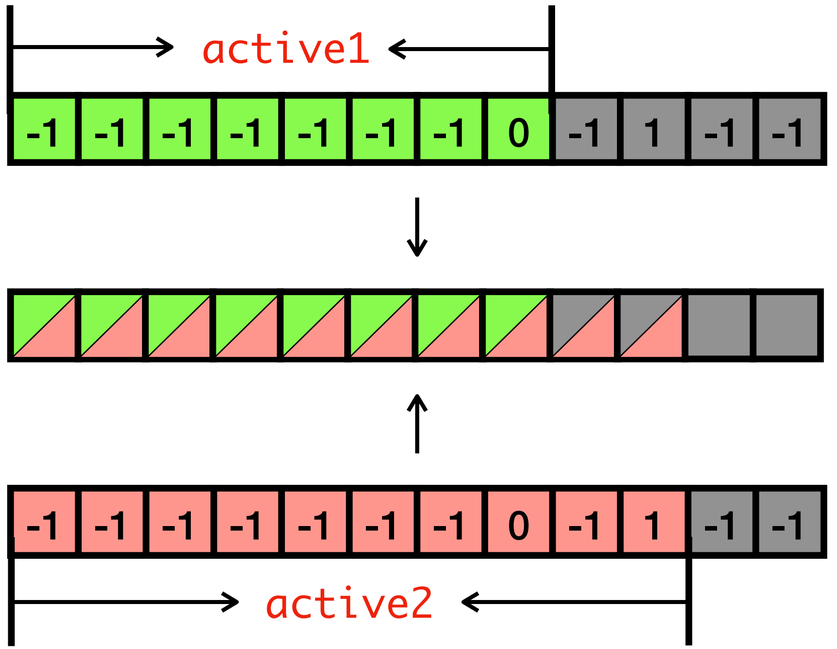
Scheduling

```
with i = mu(0, i')
do
   x = a[i]                 : true
   found = cmp eq x, 0      : true
   i' = i + 1               : true
while not found             : true
with i2 = mu(0, i2')
do
   y = a[i2]                : true
   found2 = cmp eq y, 1     : true
   i2' = i2 + 1             : true
while not found2            : true
idxs[0] = i                 : true
idxs[1] = i2                : true
```



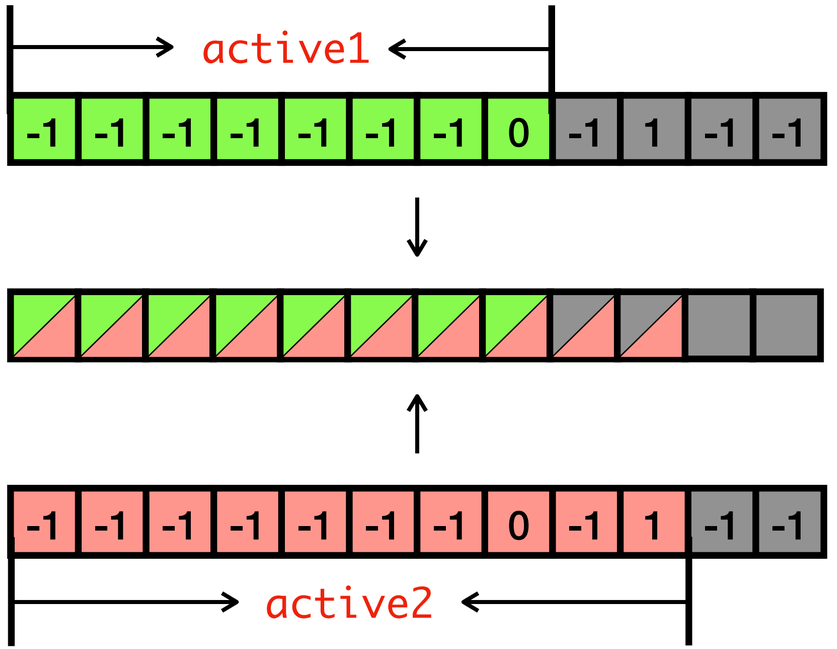
Scheduling: Rewriting Predicates

```
with i = mu(0, i')
     i2 = mu(0, i2')
do
   x = a[i]                 : true
   found = cmp eq x, 0      : true
   i' = i + 1               : true
   y = a[i2]                : true
   found2 = cmp eq y, 1     : true
   i2' = i2 + 1             : true
while not found             : true
idxs[0] = i                 : true
idxs[1] = i2                : true
```



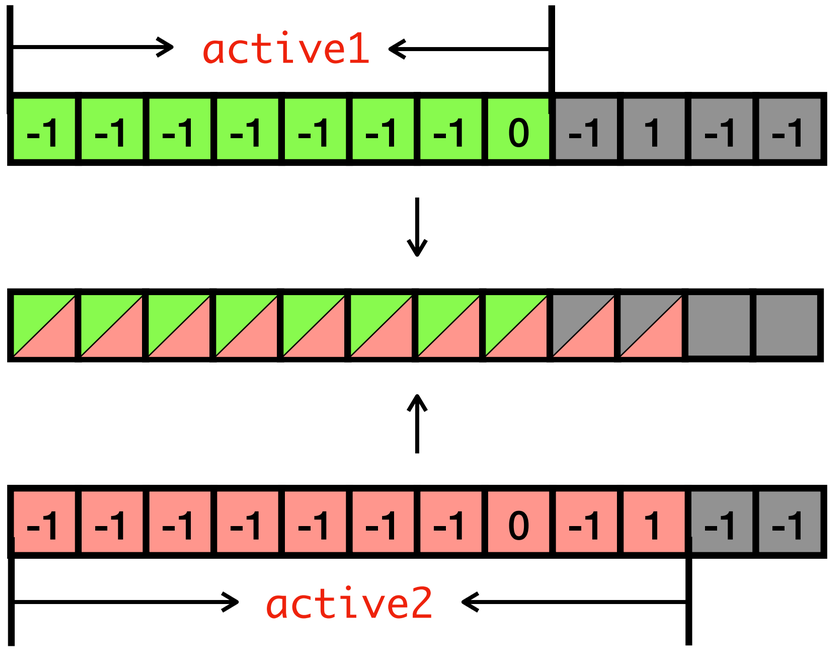
Scheduling: Rewriting Predicates

```
with i = mu(0, i')
     i2 = mu(0, i2')
do
   x = a[i]                 : true
   found = cmp eq x, 0      : true
   i' = i + 1               : true
   y = a[i2]                : true
   found2 = cmp eq y, 1     : true
   i2' = i2 + 1             : true
while active1 or active2    : true
idxs[0] = i                 : true
idxs[1] = i2                : true
```



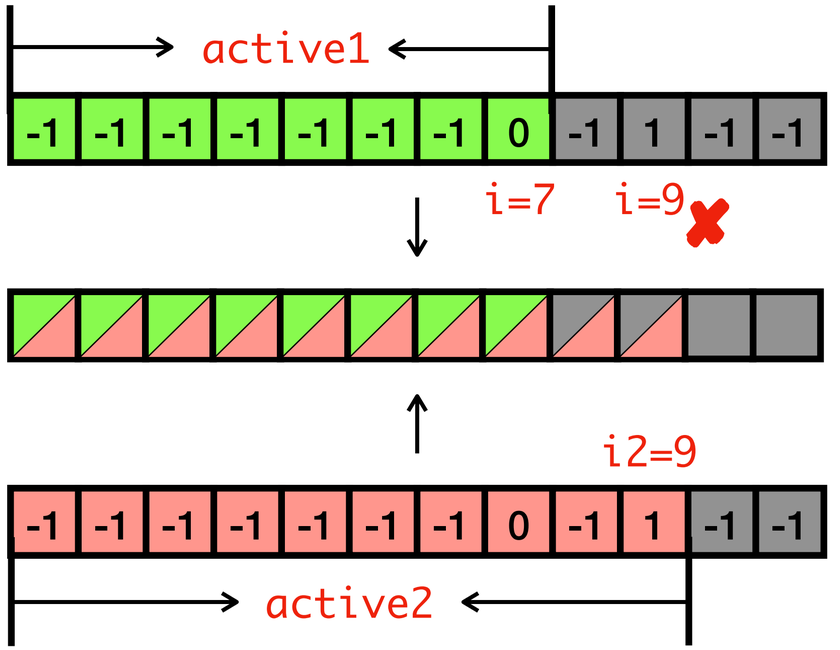
Scheduling: Rewriting Predicates

```
with i = mu(0, i')
     i2 = mu(0, i2')
do
   x = a[i]                 : active1
   found = cmp eq x, 0      : active1
   i' = i + 1               : active1
   y = a[i2]                : active2
   found2 = cmp eq y, 1     : active2
   i2' = i2 + 1             : active2
while active1 or active2    : true
idxs[0] = i                 : true
idxs[1] = i2                : true
```



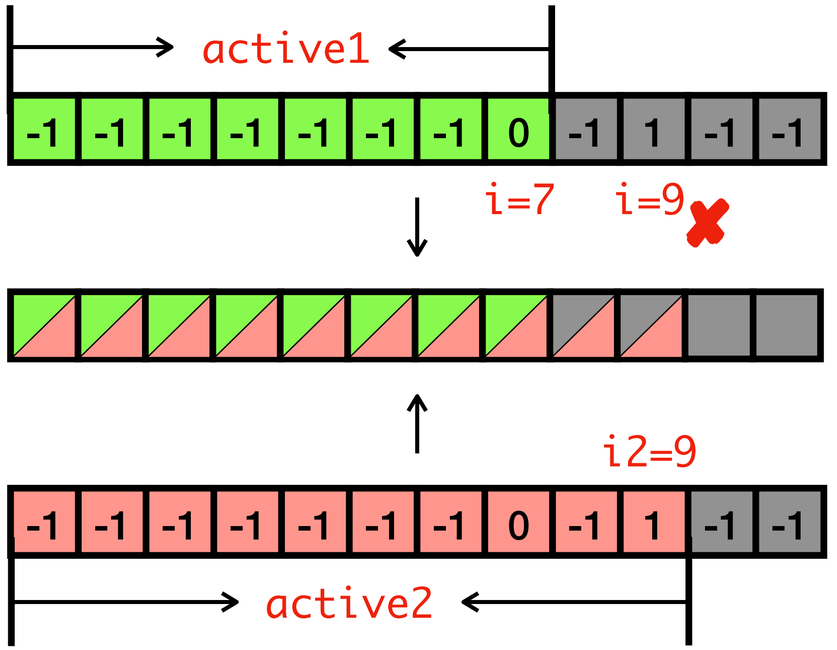
Scheduling: Guard Live-Out Values

```
with i = mu(0, i')
     i2 = mu(0, i2')
do
   x = a[i]                  : active1
   found = cmp eq x, 0       : active1
   i' = i + 1                : active1
   y = a[i2]                 : active2
   found2 = cmp eq y, 1      : active2
   i2' = i2 + 1              : active2
while active1 or active2     : true
idxs[0] = i                  : true
idxs[1] = i2                 : true
```



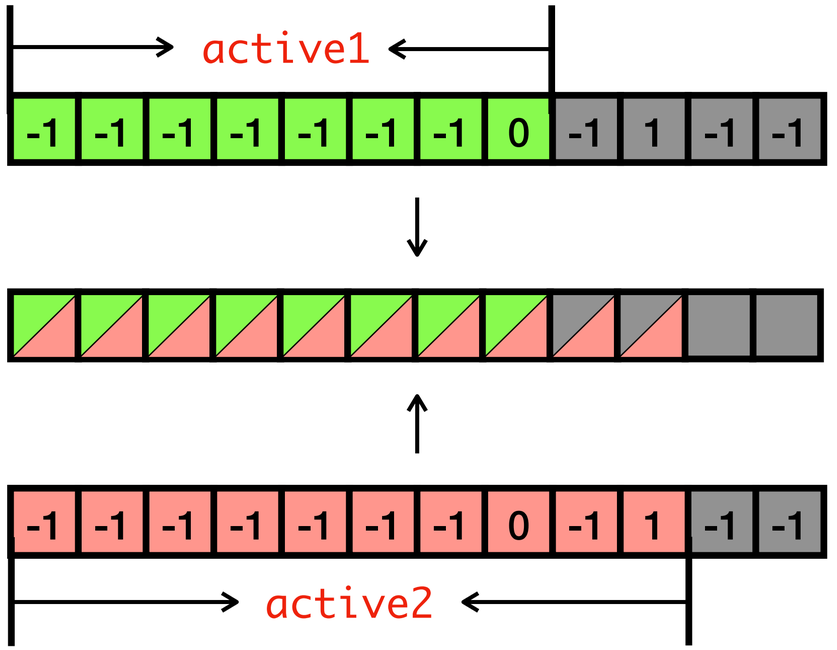
Scheduling: Guard Live-Out Values

```
with i = mu(0, i')
     i2 = mu(0, i2')
     i_out = mu(undef, i_out')
do
   x = a[i]                 : active1
   found = cmp eq x, 0      : active1
   i' = i + 1               : active1
   y = a[i2]                : active2
   found2 = cmp eq y, 1     : active2
   i2' = i2 + 1             : active2
    i_out' = phi(
      active1: i, _: i_out) : true
while active1 or active2: true
idxs[0] = i_out'           : true
idxs[1] = i2               : true
```



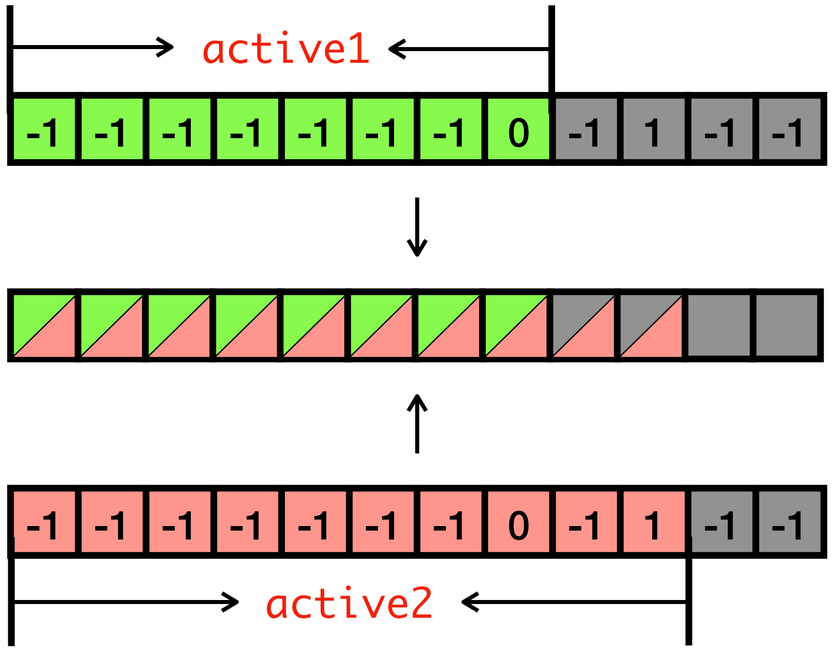
Scheduling

```
with i = mu(0, i')
     i2 = mu(0, i2')
                  …
do
   x = a[i]                : active1
   found = cmp eq x, 0     : active1
   i' = i + 1              : active1
   y = a[i2]               : active2
   found2 = cmp eq y, 1    : active2
   i2' = i2 + 1            : active2
                  …
while active1 or active2: true
idxs[0] = i_out'          : true
idxs[1] = i2_out'         : true
```



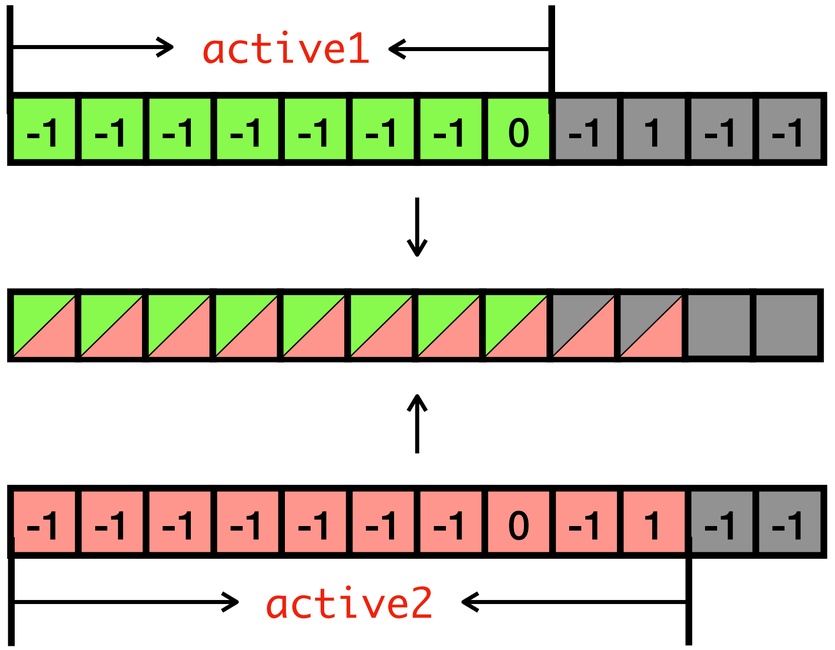
Scheduling

```
with i = mu(0, i')
     i2 = mu(0, i2')
     active1 = mu(true, active1')
     active2 = mu(true, active2')
                    …
do
   x = a[i]                 : active1
   found = cmp eq x, 0      : active1
   i' = i + 1               : active1
   y = a[i2]                : active2
   found2 = cmp eq y, 1     : active2
   i2' = i2 + 1             : active2
                    …
while active1 or active2: true
idxs[0] = i_out'         : true
idxs[1] = i2_out'        : true
```



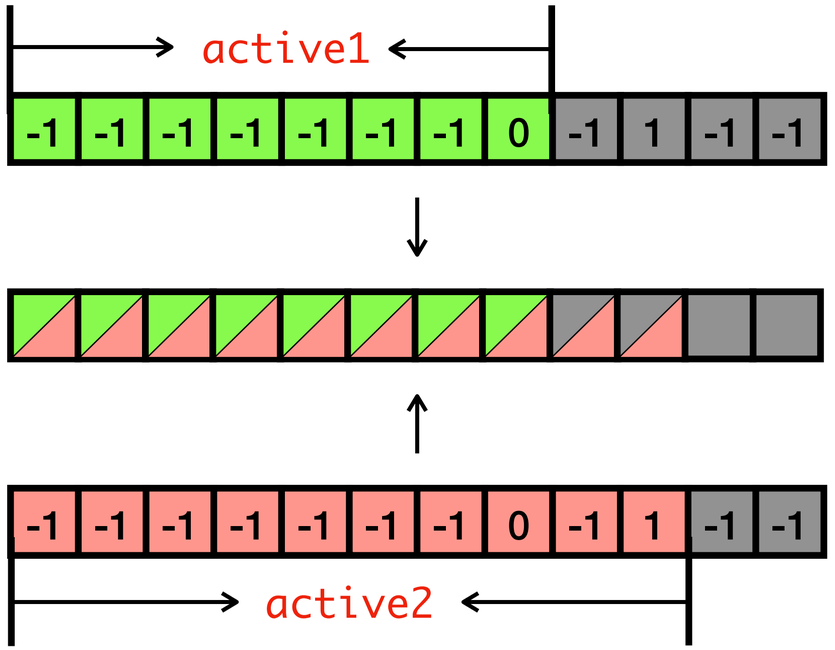
Scheduling

```
with i = mu(0, i')
     i2 = mu(0, i2')
     active1 = mu(true, active1')
     active2 = mu(true, active2')
                    …
do
   x = a[i]                 : active1
   found = cmp eq x, 0      : active1
   i' = i + 1               : active1
   y = a[i2]                : active2
   found2 = cmp eq y, 1     : active2
   i2' = i2 + 1             : active2
    active1' = active1 and not found: true
    active2' = active2 and not found2: true
                    …
while active1 or active2: true
idxs[0] = i_out'          : true
idxs[1] = i2_out'         : true
```



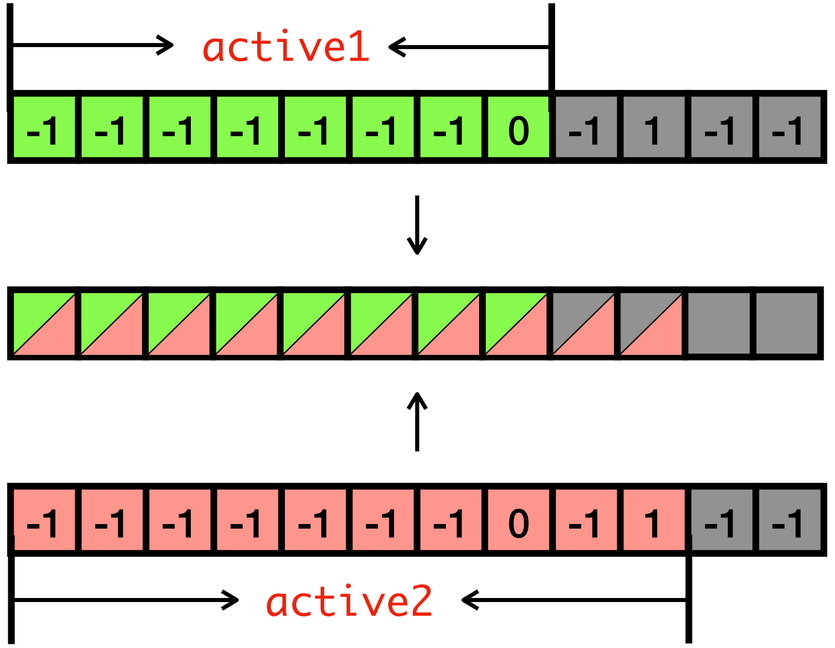
Scheduling

```
with i = mu(0, i')
     i2 = mu(0, i2')
     active1 = mu(true, active1')
     active2 = mu(true, active2')
                  …
do
   x = a[i]                : active1
   found = cmp eq x, 0     : active1
   i' = i + 1              : active1
   y = a[i2]               : active2
   found2 = cmp eq y, 1    : active2
   i2' = i2 + 1            : active2
    active1' = active1 and not found: true
    active2' = active2 and not found2: true
                  …
while active1 or active2: true
idxs[0] = i_out'         : true
idxs[1] = i2_out'        : true
```



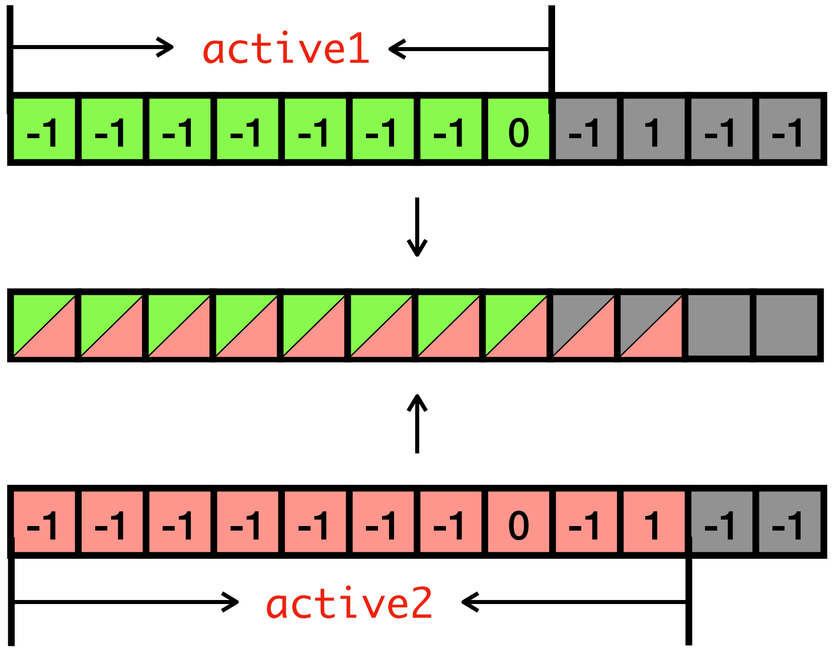
Scheduling

```
with i = mu(0, i')
     i2 = mu(0, i2')
     active1 = mu(true, active1')
     active2 = mu(true, active2')
                  …
do
   x = a[i]                 : active1
   y = a[i2]                : active2
   found = cmp eq x, 0      : active1
   found2 = cmp eq y, 1     : active2
   i' = i + 1               : active1
   i2' = i2 + 1             : active2
    active1' = active1 and not found: true
    active2' = active2 and not found2: true
                  …
while active1 or active2: true
idxs[0] = i_out'         : true
idxs[1] = i2_out'        : true
```



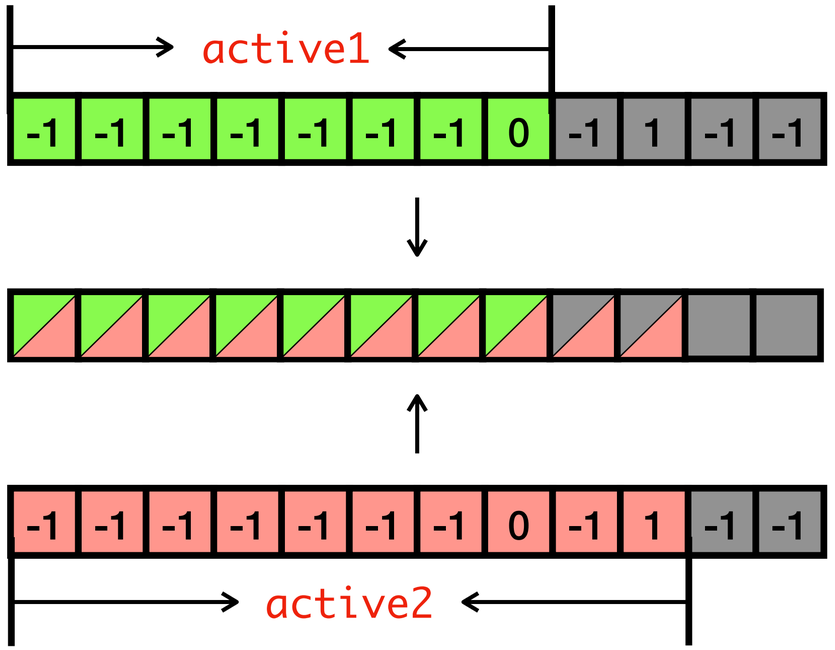
Scheduling

```
with i = mu(0, i')
     i2 = mu(0, i2')
     active1 = mu(true, active1')
     active2 = mu(true, active2')
                  …
do
   x = a[i]                  : active1
   y = a[i2]                 : active2
   found = cmp eq x, 0       : active1
   found2 = cmp eq y, 1      : active2
   i' = i + 1                : active1
   i2' = i2 + 1              : active2
   active1' = active1 and not found: true
   active2' = active2 and not found2: true
                  …
while active1 or active2: true
idxs[0] = i_out'          : true
idxs[1] = i2_out'         : true
```



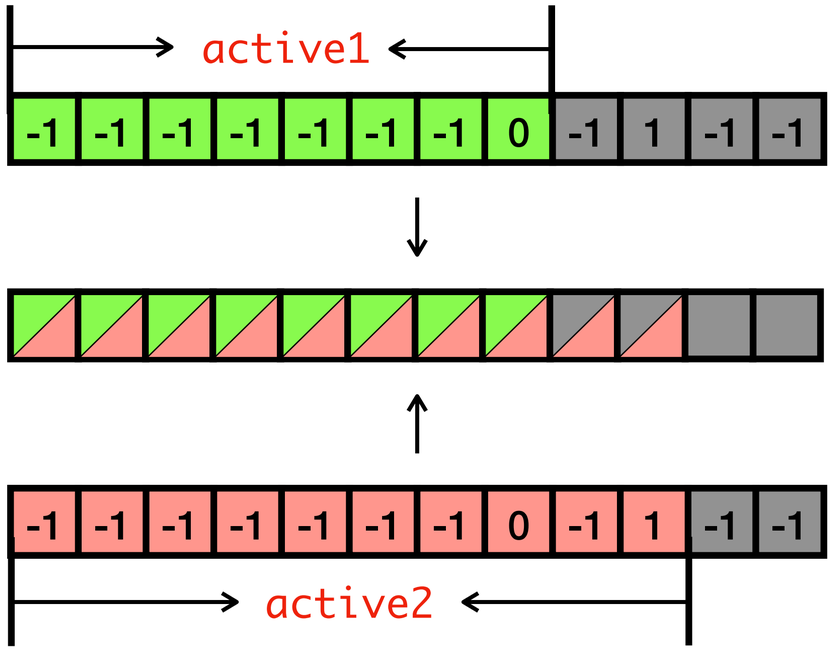
Scheduling

```
with i = mu(0, i’)
     i2 = mu(0, i2’)
     active1 = mu(true, active1’)
     active2 = mu(true, active2’)
                    …
do
  xy = a[i:i+2]             : …
   found = cmp eq x, 0      : active1
   found2 = cmp eq y, 1     : active2
   i’ = i + 1               : active1
   i2’ = i2 + 1             : active2
   active1’ = active1 and not found: true
   active2’ = active2 and not found2: true
                    …
while active1 or active2: true
idxs[0] = i_out’          : true
idxs[1] = i2_out’         : true
```



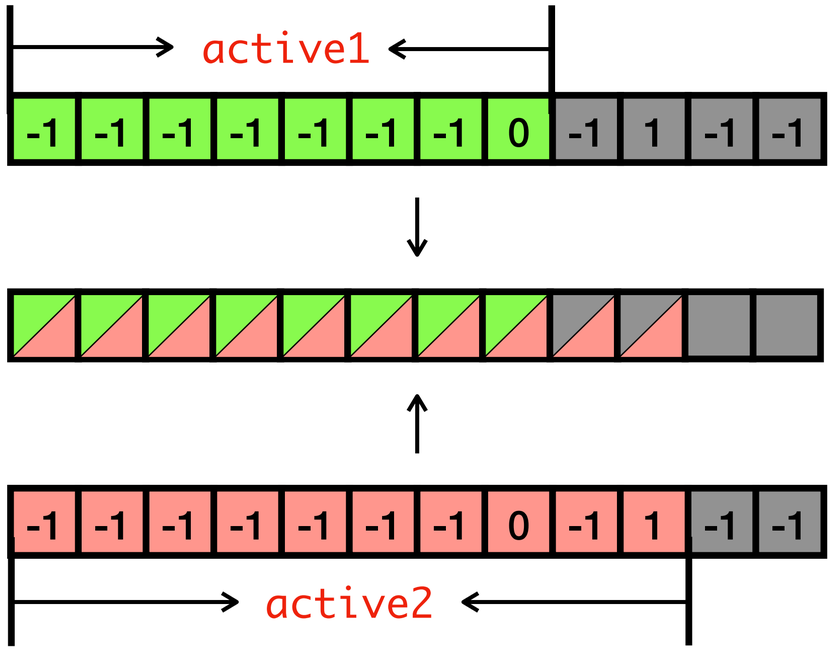
Scheduling

```
with i = mu(0, i’)
     i2 = mu(0, i2’)
     active1 = mu(true, active1’)
     active2 = mu(true, active2’)
                    …
do
   xy = a[i:i+2]           : …
   found = cmp eq x, 0     : active1
   found2 = cmp eq y, 1    : active2
   i’ = i + 1              : active1
   i2’ = i2 + 1            : active2
   active1’ = active1 and not found: true
   active2’ = active2 and not found2: true
                    …
while active1 or active2: true
idxs[0] = i_out’          : true
idxs[1] = i2_out’         : true
```



Scheduling

```
with i = mu(0, i’)
     i2 = mu(0, i2’)
     active1 = mu(true, active1’)
     active2 = mu(true, active2’)
                    …
do
   xy = a[i:i+2]           : …
   found = vcmp eq xy, {0,1} : …
   i’ = i + 1              : active1
   i2’ = i2 + 1            : active2
   active1’ = active1 and not found: true
   active2’ = active2 and not found2: true
                    …
while active1 or active2: true
idxs[0] = i_out’           : true
idxs[1] = i2_out’          : true
```



```
with i = mu(0, i') do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i' = add i, 1             : true
  lt_n = cmp lt i', n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2') do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2' = add i2, 1           : true
  lt_n2 = cmp lt i2', n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i')
                  …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                  …

while active1 or active2    : true
idxs[1] = i                 : found_out'
idxs[1] = i2                : found_out2'
```

Scheduling

```
with i = mu(0, i')
                  …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                  …

while active1 or active2    : true
masked-vstore idxs, i_out', found_out': true
```

Vector Code Generation

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
idxs[1] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Scheduling

```
with i = mu(0, i’)
                …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
masked-vstore idxs, i_out’, found_out’: true
```

Vector Code Generation

Vector Code Generation

```
a[0] = 0 : c
a[1] = 0 : c
```

```
vstore {0,0}, a : c
```

```
a[0] = 0 : c1
a[1] = 0 : c2
```

```
masked-vstore {0,0}, a, {c1,c2} : true
```

```
t1 = phi(c : 0, not c : 1)
t2 = phi(c : 0, not c : 1)
```

```
t = phi(c : {0,0}, not c : {1,1})
```

```
t1 = phi(c1 : 0, not c1 : 1)
t2 = phi(c2 : 0, not c2 : 1)
```

```
t = vselect {c1,c2}, {0,0}, {1,1}
```

```
with i = mu(0, i’) do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i’ = add i, 1             : true
  lt_n = cmp lt i’, n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2’) do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2’ = add i2, 1           : true
  lt_n2 = cmp lt i2’, n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i’)
                …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
idxs[1] = i                 : found_out’
idxs[1] = i2                : found_out2’
```

Scheduling

```
with i = mu(0, i’)
                …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i’ = add i, 1             : active1
  i2’ = add i2, 1           : active2

                …
while active1 or active2    : true
masked-vstore idxs, i_out’, found_out’: true
```

Vector Code Generation

```
with i = mu(0, i') do
  x = arr[i]                : true
  k = keys[0]               : true
  found = cmp eq t, k       : true
  i' = add i, 1             : true
  lt_n = cmp lt i', n       : true
while not found and lt_n    : true
idxs[0] = i                 : found
with i2 = mu(0, i2') do
  x2 = arr[i2]              : true
  k2 = keys[1]              : true
  found2 = cmp eq t2, k2    : true
  i2' = add i2, 1           : true
  lt_n2 = cmp lt i2', n     : true
while not found2 and lt_n2  : true
idxs[1] = i2                : found2
```

Convert to Predicated SSA

```
with i = mu(0, i')
                  …
  x = arr[i]                : active1
  x2 = arr[i2]              : active2
  k = keys[0]               : active1
  k2 = keys[1]              : active2
  found = cmp eq t, k       : active1
  found2 = cmp eq t2, k2    : active2
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                  …

while active1 or active2    : true
idxs[1] = i                 : found_out'
idxs[1] = i2                : found_out2'
```

Scheduling

```
with i = mu(0, i')
                  …
  x = arr[i:i+2]            : true
  k = keys[0:2]             : true
  found = vcmp eq x, k      : true
  i' = add i, 1             : active1
  i2' = add i2, 1           : active2

                  …

while active1 or active2    : true
masked-vstore idxs, i_out', found_out': true
```

Vector Code Generation

Evaluation

Vectorizing ISPC's serial baseline

releases/v1.16...

This branch is 5 commits ahea

aneshlya and dbaboki

..

.gitignore

CMakeLists.txt

camera.dat

density_highres.vol

density_lowres.vol

volume.cpp

volume.ispc

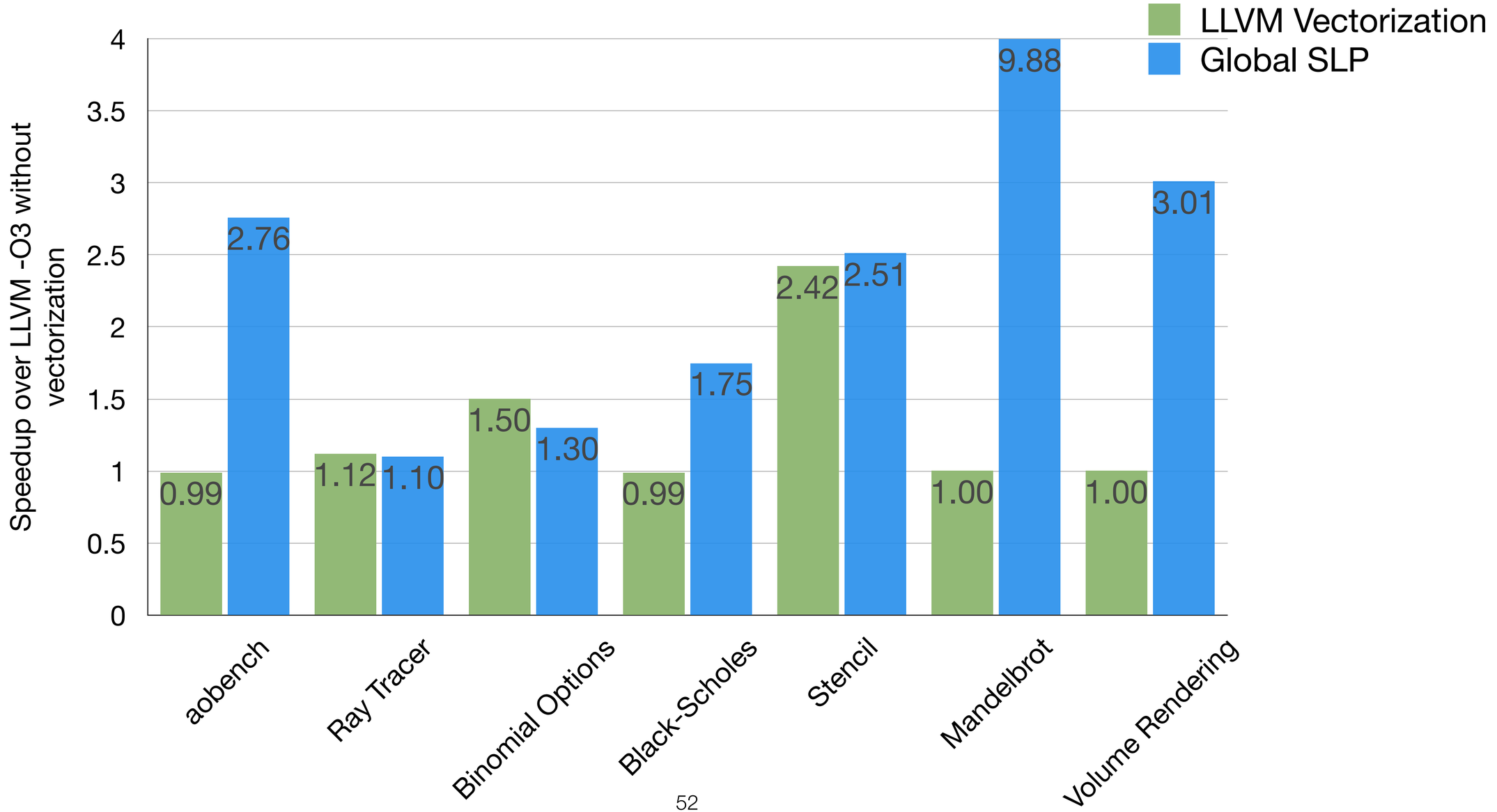
volume_serial.cpp

volume_serial.cpp

github.com/ispc/ispc/blob/releases/v1.16.x/examples/cpu/volume_rendering/volume_serial.cpp

```
221     float stepT = stepDist / rayLength;
222
223     float t = rayT0;
224     float3 pos = ray.origin + ray.dir * rayT0;
225     float3 dirStep = ray.dir * stepT;
226     while (t < rayT1) {
227         float d = Density(pos, pMin, pMax, density, nVoxels);
228
229         // terminate once attenuation is high
230         float atten = expf(-tau);
231         if (atten < .005f)
232             break;
233
234         // direct lighting
235         float Li = lightIntensity / distanceSquared(lightPos, pos) *
236                    transmittance(lightPos, pos, pMin, pMax, sigma_a + sigma_s, density, nVoxels);
237         L += stepDist * atten * d * sigma_s * (Li + Le);
238
239         // update beam transmittance
240         tau += stepDist * (sigma_a + sigma_s) * d;
241
242         pos = pos + dirStep;
243         t += stepT;
244     }
245
246     // Gamma correction
247     return powf(L, 1.f / 2.2f);
248 }
249
250 void volume_serial(float density[], int nVoxels[3], const float raster2camera[4][4], const float camera2
251                    int width, int height, float image[]) {
252     int offset = 0;
253     for (int y = 0; y < height; ++y) {
```

Vectorizing ISPC's serial baseline



Conclusion and Future Work

- Our framework generalizes SLP to arbitrary (reducible) control flow
- Can accelerate benchmarks with complex, irregular control flow