编译优化：SLP矢量化

皮昊旋

[1].参考推文
[2].参考论文1-Exploiting Superword Level Parallelism with Multimedia Instruction Sets
[3].参考论文2-Loop-Aware SLP in GCC

Superword Level Parallelism

SIMD指令

将**相似**的**独立**指令组合成向量指令

单指令**多**数据

```
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

```
# %bb.0:
        addss   %xmm0, %xmm2
        mulss   %xmm0, %xmm2
        movss   %xmm2, (%rdi)
        addss   %xmm1, %xmm3
        mulss   %xmm1, %xmm3
        movss   %xmm3, 4(%rdi)
        movss   %xmm2, 8(%rdi)
        movss   %xmm3, 12(%rdi)
        retq
```

clang++ case.cpp -O3 -fno-slp-vectorize -S

%xmm0	a_1
%xmm1	a_2
%xmm2	b_1
%xmm3	b_2

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

```asm
# %bb.0:
        addss   %xmm0, %xmm2
        mulss   %xmm0, %xmm2
        movss   %xmm2, (%rdi)
        addss   %xmm1, %xmm3
        mulss   %xmm1, %xmm3
        movss   %xmm3, 4(%rdi)
        movss   %xmm2, 8(%rdi)
        movss   %xmm3, 12(%rdi)
        retq
```

%xmm0	a_1
%xmm1	a_2
%xmm2	$a_1 + b_1$
%xmm3	b_2

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

```asm
# %bb.0:
        addss   %xmm0, %xmm2
        mulss   %xmm0, %xmm2
        movss   %xmm2, (%rdi)
        addss   %xmm1, %xmm3
        mulss   %xmm1, %xmm3
        movss   %xmm3, 4(%rdi)
        movss   %xmm2, 8(%rdi)
        movss   %xmm3, 12(%rdi)
        retq
```

%xmm0	a_1
%xmm1	a_2
%xmm2	$a_1 * (a_1 + b_1)$
%xmm3	b_2

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

```asm
# %bb.0:
        addss   %xmm0, %xmm2
        mulss   %xmm0, %xmm2
        movss   %xmm2, (%rdi)
        addss   %xmm1, %xmm3
        mulss   %xmm1, %xmm3
        movss   %xmm3, 4(%rdi)
        movss   %xmm2, 8(%rdi)
        movss   %xmm3, 12(%rdi)
        retq
```

	%xmm0	a_1
	%xmm1	a_2
A[0] =	%xmm2	$a_1 * (a_1 + b_1)$
	%xmm3	b_2

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

```asm
# %bb.0:
        addss   %xmm0, %xmm2
        mulss   %xmm0, %xmm2
        movss   %xmm2, (%rdi)
        addss   %xmm1, %xmm3
        mulss   %xmm1, %xmm3
        movss   %xmm3, 4(%rdi)
        movss   %xmm2, 8(%rdi)
        movss   %xmm3, 12(%rdi)
        retq
```

	%xmm0	a_1
	%xmm1	a_2
A[0] =	%xmm2	$a_1 * (a_1 + b_1)$
A[1] =	%xmm3	$a_2 * (a_2 + b_2)$

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

```asm
# %bb.0:
        addss   %xmm0, %xmm2
        mulss   %xmm0, %xmm2
        movss   %xmm2, (%rdi)
        addss   %xmm1, %xmm3
        mulss   %xmm1, %xmm3
        movss   %xmm3, 4(%rdi)
        movss   %xmm2, 8(%rdi)
        movss   %xmm3, 12(%rdi)
        retq
```

	%xmm0	a_1
	%xmm1	a_2
A[2] = A[0] =	%xmm2	$a_1 * (a_1 + b_1)$
A[3] = A[1] =	%xmm3	$a_2 * (a_2 + b_2)$

```cpp
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

clang++ case.cpp -O3 -S

%xmm0	a_1
%xmm1	a_2
%xmm2	b_1
%xmm3	b_2

```asm
# %bb.0:
        unpcklps    %xmm1, %xmm0                    # xmm0 = xmm0[0],xmm1[0],xmm0[1],xmm1[1]
        unpcklps    %xmm3, %xmm2                    # xmm2 = xmm2[0],xmm3[0],xmm2[1],xmm3[1]
        addps   %xmm0, %xmm2
        mulps   %xmm0, %xmm2
        movlhps %xmm2, %xmm2                   # xmm2 = xmm2[0,0]
        movups  %xmm2, (%rdi)
        retq
```

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

%xmm0	$[a_1, a_2, -, -]$
%xmm1	a_2
%xmm2	$[b_1, b_2, -, -]$
%xmm3	b_2

```asm
# %bb.0:
        unpcklps    %xmm1, %xmm0                    # xmm0 = xmm0[0],xmm1[0],xmm0[1],xmm1[1]
        unpcklps    %xmm3, %xmm2                    # xmm2 = xmm2[0],xmm3[0],xmm2[1],xmm3[1]
        addps   %xmm0, %xmm2
        mulps   %xmm0, %xmm2
        movlhps %xmm2, %xmm2                    # xmm2 = xmm2[0,0]
        movups  %xmm2, (%rdi)
        retq
```

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

%xmm0	$[a_1, a_2, -, -]$
%xmm1	a_2
%xmm2	$[a_1 + b_1, a_2 + b_2, -, -]$
%xmm3	b_2

```
# %bb.0:
    unpcklps    %xmm1, %xmm0                # xmm0 = xmm0[0],xmm1[0],xmm0[1],xmm1[1]
    unpcklps    %xmm3, %xmm2                # xmm2 = xmm2[0],xmm3[0],xmm2[1],xmm3[1]
    addps   %xmm0, %xmm2
    mulps   %xmm0, %xmm2
    movlhps %xmm2, %xmm2                # xmm2 = xmm2[0,0]
    movups  %xmm2, (%rdi)
    retq
```

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

%xmm0	$[a_1, a_2, -, -]$
%xmm1	a_2
%xmm2	$[a_1*(a_1 + b_1), a_2*(a_2 + b_2), -, -]$
%xmm3	b_2

```
# %bb.0:
    unpcklps    %xmm1, %xmm0                # xmm0 = xmm0[0],xmm1[0],xmm0[1],xmm1[1]
    unpcklps    %xmm3, %xmm2                # xmm2 = xmm2[0],xmm3[0],xmm2[1],xmm3[1]
    addps   %xmm0, %xmm2
    mulps   %xmm0, %xmm2
    movlhps %xmm2, %xmm2                # xmm2 = xmm2[0,0]
    movups  %xmm2, (%rdi)
    retq
```

```c
void foo(float a1, float a2, float b1, float b2, float *A) {
  A[0] = a1*(a1 + b1);
  A[1] = a2*(a2 + b2);
  A[2] = a1*(a1 + b1);
  A[3] = a2*(a2 + b2);
}
```

%xmm0	$[a_1, a_2, -, -]$
%xmm1	a_2
%xmm2	$[a_1*(a_1 + b_1), a_2*(a_2 + b_2), a_1*(a_1 + b_1), a_2*(a_2 + b_2),]$
%xmm3	b_2

```asm
# %bb.0:
    unpcklps    %xmm1, %xmm0                # xmm0 = xmm0[0],xmm1[0],xmm0[1],xmm1[1]
    unpcklps    %xmm3, %xmm2                # xmm2 = xmm2[0],xmm3[0],xmm2[1],xmm3[1]
    addps   %xmm0, %xmm2
    mulps   %xmm0, %xmm2
    movlhps %xmm2, %xmm2                # xmm2 = xmm2[0,0]
    movups  %xmm2, (%rdi)
    retq
```

```
a = b + c * z[i+0]
d = e + f * z[i+1]
r = s + t * z[i+2]
w = x + y * z[i+3]
```

$$\begin{bmatrix} a \\ d \\ r \\ w \end{bmatrix} = \begin{bmatrix} b \\ e \\ s \\ x \end{bmatrix} +_{\mathrm{SIMD}} \begin{bmatrix} c \\ f \\ t \\ y \end{bmatrix} *_{\mathrm{SIMD}} \begin{bmatrix} z[i+0] \\ z[i+1] \\ z[i+2] \\ z[i+3] \end{bmatrix}$$

Figure 1: Isomorphic statements that can be packed and executed in parallel.

```
for (i=0; i<16; i++) {
  localdiff = ref[i] - curr[i];
  diff += abs(localdiff);
}
```

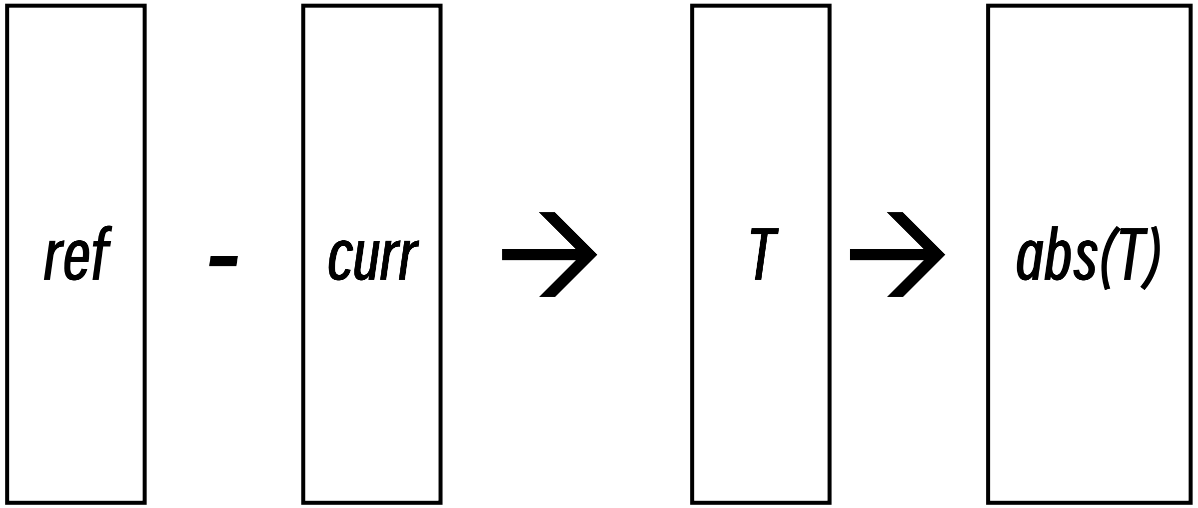
(a) Original loop.

```
for (i=0; i<16; i++) {
  T[i] = ref[i] - curr[i];
}

for (i=0; i<16; i++) {
  diff += abs(T[i]);
}
```

(b) After scalar expansion and loop fission.

体现出较强的数据依赖性 ⟹ 对向量化造成阻碍



减少数据依赖之后，即可使用SIMD指令，在一个时钟周期内计算多个元素的差值，从而提高循环效率。

```
for (i=0; i<16; i+=4) {
   localdiff = ref[i+0] - curr[i+0];
   diff += abs(localdiff);

   localdiff = ref[i+1] - curr[i+1];
   diff += abs(localdiff);

   localdiff = ref[i+2] - curr[i+2];
   diff += abs(localdiff);

   localdiff = ref[i+3] - curr[i+3];
   diff += abs(localdiff);
 }
```

(c) Superword level parallelism exposed after unrolling.

```
for (i=0; i<16; i+=4) {
  localdiff0 = ref[i+0] - curr[i+0];
  localdiff1 = ref[i+1] - curr[i+1];
  localdiff2 = ref[i+2] - curr[i+2];
  localdiff3 = ref[i+3] - curr[i+3];

  diff += abs(localdiff0);
  diff += abs(localdiff1);
  diff += abs(localdiff2);
  diff += abs(localdiff3);
}
```

(d) Packable statements grouped together after renaming.

手动构造类似于向量的代码，
从而减少循环次数，提高效率
（循环展开后暴露了并行性）

重命名消除数据依赖
将同类型操作放在一起

```
do {
  dst[0] = (src1[0] + src2[0]) >> 1;
  dst[1] = (src1[1] + src2[1]) >> 1;
  dst[2] = (src1[2] + src2[2]) >> 1;
  dst[3] = (src1[3] + src2[3]) >> 1;

  dst  += 4;
  src1 += 4;
  src2 += 4;
}
while (dst != end);
```

Figure 3: An example of a hand-optimized matrix operation that proves unvectorizable.

循环向量化:

①先将do while 循环转换成for循环（恢复归纳变量）

②将展开循环恢复成未展开的状态（rerolling）

③再进行上述优化

SLP：直接做**基本块（basic block）**间优化

循环向量化：更关注**迭代间**的优化

SLP：更关注**迭代内**的优化

机会来了

预处理

- 循环展开 Loop Unrolling
- 对齐分析 Alignment analysis
- 预 优 化 Pre-Optimization

```
SLP_extract: BasicBlock B → BasicBlock
    PackSet P ← ∅
    P ← find_adj_refs(B, P)
    P ← extend_packlist(B, P)
    P ← combine_packs(P)
    return schedule(B, [ ], P)
```

识别相邻内存引用 *find_adj_refs*

“扩张”PackSet *extend_packlist*

组合Packs *combine_packs*

准备调度 *schedule*

识别相邻内存引用 find_adj_refs

find_adj_refs: BasicBlock B $\times$ PackSet P $\rightarrow$ PackSet
  foreach Stmt $s \in B$ **do**
    foreach Stmt $s' \in B$ **where** $s \neq s'$ **do**
      if has_mem_ref(s) $\wedge$ has_mem_ref(s') **then**
        if adjacent(s, s') **then**
          Int $align \leftarrow$ get_alignment(s)
          if stmts_can_pack($B, P, s, s', align$) **then**
            $P \leftarrow P \cup \{\langle s, s' \rangle\}$
  return P

get_alignment(s)

int align

对齐分析 Alignment analysis

U

```
(1)   b = a[i+0]
(2)   c = 5
(3)   d = b + c

(4)   e = a[i+1]
(5)   f = 6
(6)   g = e + f

(7)   h = a[i+2]
(8)   j = 7
(9)   k = h + j
```

(a)

U

```
(2)   c = 5
(3)   d = b + c

(5)   f = 6
(6)   g = e + f

(8)   j = 7
(9)   k = h + j
```

P

```
(1)   b = a[i+0]
(4)   e = a[i+1]

(4)   e = a[i+1]
(7)   h = a[i+2]
```

(b)

<1,4>
<4,7>

find_adj_refs: $\text{BasicBlock } B \times \text{PackSet } P \rightarrow \text{PackSet}$

foreach Stmt $s \in B$ **do**
  foreach Stmt $s' \in B$ **where** $s \neq s'$ **do**
    if has_mem_ref(s) $\wedge$ has_mem_ref(s') **then**
      if adjacent(s, s') **then**
        Int $align \leftarrow$ get_alignment(s)
        if stmts_can_pack($B, P, s, s', align$) **then**
          $P \leftarrow P \cup \{\langle s, s' \rangle\}$
return P

stmts_can_pack: $\text{BasicBlock } B \times \text{PackSet } P \times \text{Stmt } s \times \text{Stmt } s' \times \text{Int } align \rightarrow \text{Boolean}$

if isomorphic(s, s') **then**
  if independent(s, s') **then**
    if $\forall \langle t, t' \rangle \in P.t \neq s$ **then**
      if $\forall \langle t, t' \rangle \in P.t' \neq s'$ **then**
        Int $align_s \leftarrow$ get_alignment(s)
        Int $align_{s'} \leftarrow$ get_alignment(s')
        if $align_s \equiv \top \vee align_s \equiv align$ **then**
          if $align_{s'} \equiv \top \vee align_{s'} \equiv align +$ data_size(s') **then**
            return true
return false

“扩张”PackSet extend_packlist

extend_packlist: $\text{BasicBlock } B \times \text{PackSet } P \rightarrow \text{PackSet}$

repeat

  $\text{PackSet } P_{prev} \leftarrow P$

  foreach $\text{Pack } p \in P$ **do**

    $P \leftarrow \text{follow\_use\_defs}(B, P, p)$

    $P \leftarrow \text{follow\_def\_uses}(B, P, p)$

until $P \equiv P_{prev}$

return P

U

```
(2)   c = 5
(3)   d = b + c

(5)   f = 6
(6)   g = e + f

(8)   j = 7
(9)   k = h + j
```

P

```
(1)   b = a[i+0]
(4)   e = a[i+1]

(4)   e = a[i+1]
(7)   h = a[i+2]
```

(b)

U

```
(2)   c = 5
(5)   f = 6
(8)   j = 7
```

P

```
(1)   b = a[i+0]
(4)   e = a[i+1]

(4)   e = a[i+1]
(7)   h = a[i+2]

(3)   d = b + c
(6)   g = e + f

(6)   g = e + f
(9)   k = h + j
```

(c)

P

```
(1)   b = a[i+0]
(4)   e = a[i+1]

(4)   e = a[i+1]
(7)   h = a[i+2]

(3)   d = b + c
(6)   g = e + f

(6)   g = e + f
(9)   k = h + j

(2)   c = 5
(5)   f = 6

(5)   f = 6
(8)   j = 7
```

(d)

$\textsf{follow\_use\_defs}$: BasicBlock B × PackSet P × Pack p → PackSet

where $p = \langle s, s' \rangle$, $s = [\, \mathtt{x_0} := \mathtt{f}(\mathtt{x_1}, \ldots, \mathtt{x_m}) \,]$, $s' = [\, \mathtt{x'_0} := \mathtt{f}(\mathtt{x'_1}, \ldots, \mathtt{x'_m}) \,]$

```
Int align ← get_alignment(s)
for j ← 1 to m do
   if ∃t ∈ B.t = [ x_j := … ] ∧ ∃t' ∈ B.t' = [ x'_j := … ] then
      if stmts_can_pack(B, P, t, t', align)
         if est_savings (⟨t, t'⟩, P) ≥ 0 then
            P ← P ∪ {⟨t, t'⟩}
            set_alignment(s, s', align)
return P
```

$\textsf{follow\_def\_uses}$: BasicBlock B × PackSet P × Pack p → PackSet

where $p = \langle s, s' \rangle$, $s = [\, \mathtt{x_0} := \mathtt{f}(\mathtt{x_1}, \ldots, \mathtt{x_m}) \,]$, $s' = [\, \mathtt{x'_0} := \mathtt{f}(\mathtt{x'_1}, \ldots, \mathtt{x'_m}) \,]$

```
Int align ← get_alignment(s)
Int savings ← −1
foreach Stmt t ∈ B where t = [ … := g(…, x_0, …) ] do
   foreach Stmt t' ∈ B where t ≠ t' = [ … := h(…, x'_0, …) ] do
      if stmts_can_pack(B, P, t, t', align) then
         if est_savings(⟨t, t'⟩, P) > savings then
            savings ← est_savings(⟨t, t'⟩, P)
            Stmt u ← t
            Stmt u' ← t'
if savings ≥ 0 then
   P ← P ∪ {⟨u, u'⟩}
   set_alignment(u, u')
return P
```

组合Packs combine_packs

combine_packs: $\text{PackSet } P \rightarrow \text{PackSet}$

repeat

$\quad\quad \text{PackSet } P_{prev} \leftarrow P$

$\quad\quad \textbf{foreach} \text{ Pack } p = \langle s_1, \ldots, s_n \rangle \in P \textbf{ do}$

$\quad\quad\quad\quad \textbf{foreach} \text{ Pack } p' = \langle s'_1, \ldots, s'_m \rangle \in P \textbf{ do}$

$\quad\quad\quad\quad\quad\quad \textbf{if } s_n \equiv s'_1 \textbf{ then}$

$\quad\quad\quad\quad\quad\quad\quad\quad P \leftarrow P - \{p, p'\} \cup \{\langle s_1, \ldots, s_n, s'_2, \ldots, s'_m \rangle\}$

$\textbf{until } P \equiv P_{prev}$

$\textbf{return } P$

去重、打包

P

```
(1)  b = a[i+0]
(4)  e = a[i+1]

(4)  e = a[i+1]
(7)  h = a[i+2]

(3)  d = b + c
(6)  g = e + f

(6)  g = e + f
(9)  k = h + j

(2)  c = 5
(5)  f = 6

(5)  f = 6
(8)  j = 7
```

(d)

组合Packs combine_packs

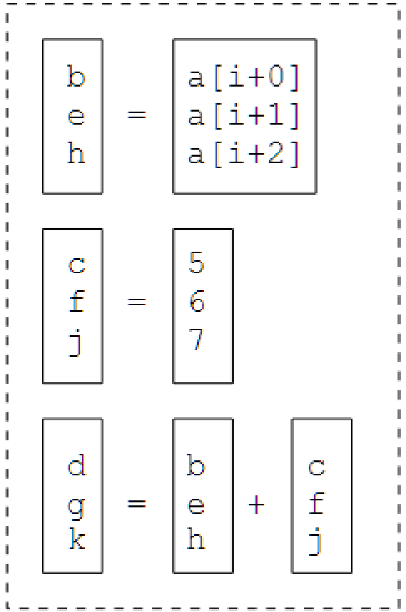
combine_packs: PackSet $P \rightarrow$ PackSet
repeat
  PackSet $P_{prev} \leftarrow P$
  foreach Pack $p = \langle s_1, \ldots, s_n \rangle \in P$ **do**
    foreach Pack $p' = \langle s'_1, \ldots, s'_m \rangle \in P$ **do**
      if $s_n \equiv s'_1$ **then**
        $P \leftarrow P - \{p, p'\} \cup \{\langle s_1, \ldots, s_n, s'_2, \ldots, s'_m \rangle\}$
until $P \equiv P_{prev}$
return P

```
(1)  b = a[i+0]
(4)  e = a[i+1]
(7)  h = a[i+2]

(3)  d = b + c
(6)  g = e + f
(9)  k = h + j

(2)  c = 5
(5)  f = 6
(8)  j = 7
```

(e)



(f)

去重、打包

准备调度 schedule

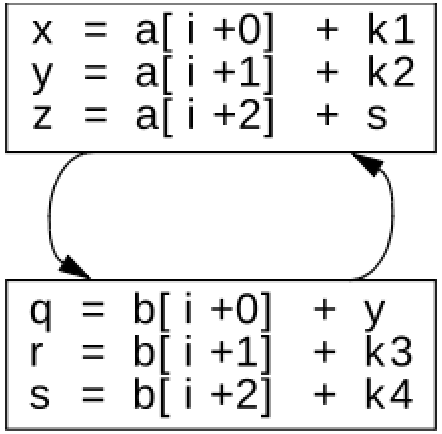
$$
\begin{array}{l}
\textsf{schedule:}\ \text{BasicBlock}\ B \times \text{BasicBlock}\ B' \times \text{PackSet}\ P \\
\qquad\qquad \rightarrow \text{BasicBlock} \\
\quad \textbf{for}\ i \leftarrow 0\ \textbf{to}\ |B|\ \textbf{do} \\
\quad\quad \textbf{if}\ \exists p = \langle \ldots, s_i, \ldots \rangle \in P\ \textbf{then} \\
\quad\quad\quad \textbf{if}\ \forall s \in p.\ \textsf{deps\_scheduled}(s, B')\ \textbf{then} \\
\quad\quad\quad\quad \textbf{foreach}\ \text{Stmt}\ s \in p\ \textbf{do} \\
\quad\quad\quad\quad\quad B \leftarrow B - s \\
\quad\quad\quad\quad\quad B' \leftarrow B' \cdot s \\
\quad\quad\quad\quad \textbf{return}\ \textsf{schedule}(B, B', P) \\
\quad\quad \textbf{else if}\ \textsf{deps\_scheduled}(s_i, B')\ \textbf{then} \\
\quad\quad\quad \textbf{return}\ \textsf{schedule}(B - s_i, B' \cdot s_i, P) \\
\quad \textbf{if}\ |B| \neq 0\ \textbf{then} \\
\quad\quad P \leftarrow P - \{p\}\ \textbf{where}\ p = \textsf{first}(B, P) \\
\quad\quad \textbf{return}\ \textsf{schedule}(B, B', P) \\
\quad \textbf{return}\ B'
\end{array}
$$

```
x = a[i+0] + k1
y = a[i+1] + k2

q = b[i+0] + y
r = b[i+1] + k3
s = b[i+2] + k4

z = a[i+2] + s
```

→



Z依赖s
q依赖y

分开**最早**未被调度的组

最早：根据原始basic block进行判定
即把xyz组中的y单独拿出先调度

缺点

不适合长向量架构

The drawback to this method is that it may not be applicable to long vector architectures. Since the unroll factor must be consistent with the vector size, unrolling may produce basic blocks that overwhelm the analysis and the code generator. As such, this method is mainly applicable to architectures with short vectors.

3.1 Loop Unrolling

Loop unrolling is performed early since it is most easily done at a high level. As discussed, it is used to transform vector parallelism into basic blocks with superword level parallelism. In order to ensure full utilization of the superword datapath in the presence of a vectorizable loop, the unroll factor must be customized to the data sizes used within the loop. For example, a vectorizable loop containing 16-bit values should be unrolled 8 times for a 128-bit datapath. Our system currently unrolls loops based on the smallest data type present.

Thank you