Loop unrolling in details

Иван Афанасьев

Unipro

C++ Russia 2023

Bio

Иван Афанасьев

- ММФ НГУ. к.ф.м.н.
- 12 лет C++
- 2 года в разработке компиляторов 👶

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

Loop from compiler perspective

```
for (i = s; i < b; i++)

  statements(i);
```

Loop from compiler perspective

```
for (i = s; i < b; i++)
  statements(i);
```

```
if (s < b) {
    i = s;
    do {
        statements(i);
        i++;
    } while (i < b);
}
```

Loop from compiler perspective

```
for (i = s; i < b; i++)
  statements(i);
```

```
    bool cond = s < b;
    if (!cond) goto LBB_EXIT;
    i = s;
LBB_HEAD:
    statements(i);
    ++i;
    bool cond = i < b;
    if (cond) goto LBB_HEAD;
LBB_EXIT:
```

Loop from compiler perspective

```
    bool cond = s < b;
    if (!cond) goto LBB_EXIT;
    i = s;
LBB_HEAD:
    statements(i);
    ++i;
    bool cond = i < b;
    if (cond) goto LBB_HEAD;
LBB_EXIT:
```

- loop guard
- prologue
- head
- loop body
- back edge

Loop payload and administration overhead

```
for (i = s; i < b; i++)
  statements(i);
```

```
    bool cond = s < b;
    if (!cond) goto LBB_EXIT;
    i = s;
LBB_HEAD:
    statements(i);
    ++i;
    bool cond = i < b;
    if (cond) goto LBB_HEAD;
LBB_EXIT:
```

Loop payload and administration overhead

```
for (i = s; i < b; i++)
  statements(i);
```

payload

```
    bool cond = s < b;
    if (!cond) goto LBB_EXIT;
    i = s;
LBB_HEAD:
    statements(i);
    ++i;
    bool cond = i < b;
    if (cond) goto LBB_HEAD;
LBB_EXIT:
```

Loop payload and administration overhead

```
for (i = s; i < b; i++)
  statements(i);
```

payload

loop administration overhead

```
    bool cond = s < b;
    if (!cond) goto LBB_EXIT;
    i = s;
LBB_HEAD:
    statements(i);
    ++i;
    bool cond = i < b;
    if (cond) goto LBB_HEAD;
LBB_EXIT:
```

Loop payload and administration overhead

```
for (i = s; i < b; i++)
  statements(i);
```

payload

loop administration overhead

unrolling

```
    bool cond = s < b;
    if (!cond) goto LBB_EXIT;
    i = s;
LBB_HEAD:
    statements(i);
    ++i;
    bool cond = i < b;
    if (cond) goto LBB_HEAD;
LBB_EXIT:
```

Examples. Loop

```c
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

A▾ Output...▾ Filter...▾ Libraries + Add new...▾ Add tool...▾

```asm
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Examples. Loop

TEST - Logical Compare

Computes the bitwise logical AND of the first operand and the second operand and sets the SF, ZF, and PF status flags.

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Examples. Loop

JLE - Conditional Jump

Jump to the destination if one or more of the status flags is set (... ZF ...).

A Output... Filter... Libraries + Add new... Add tool...

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Examples. Loop

LEA - Load Effective Address

Computes the effective address of the second operand and stores it in the first operand.

A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Examples. Loop

XOR

Performs a bitwise XOR operation on the first and second operands and stores the result in the first operan.

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Examples. Loop

```c
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

A▾ Output...▾ Filter...▾ Libraries + Add new...▾ Add tool...▾

```asm
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Examples. Loop

```c
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

loop guard (the `test esi, esi` / `jle .L4` instructions)

Examples. Loop

```c
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```asm
sum(int*, int):
        test    esi, esi                ; loop guard
        jle     .L4
        movsx   rsi, esi                ; prologue
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

loop guard

prologue

Examples. Loop

```c
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

A▾ Output...▾ Filter...▾ Libraries + Add new...▾ Add tool...▾

```asm
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

loop guard (test esi, esi / jle .L4)

prologue (movsx rsi, esi / lea rdx, [rdi+rsi*4] / xor eax, eax)

iteration check back edge (add rdi, 4 / cmp rdi, rdx / jne .L3)

Examples. Loop

```c
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

A▾ Output...▾ Filter...▾ Libraries + Add new...▾ Add tool...▾

```asm
sum(int*, int):
        test    esi, esi                  ; loop guard
        jle     .L4
        movsx   rsi, esi                  ; prologue
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]      ; payload
        add     rdi, 4                    ; iteration check back edge
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Annotations: loop guard; prologue; payload; iteration check back edge

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

Loop unrolling

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Loop unrolling

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

```
N_jumps  /= 4
N_checks /= 4
```

Loop unrolling

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

```
N_jumps  /= 4
N_checks /= 4
```

```
(b - s) % 4 == 0 ?
```

Prolog + epilogue loop

```
b′ = b - (b - s) % 4;
for (i = s; i < b′; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b′; i < b; i++)
    statements(i);
```

Overhead:

- Epilogue (remainder) loop
- Calculate b′

Prolog + epilogue loop

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

Overhead:

- Epilogue (remainder) loop
- Calculate b'

Prolog + epilogue loop

```
b′ = b - (b - s) % 4;
for (i = s; i < b′; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b′; i < b; i++)
    statements(i);
```

Overhead:

- Epilogue (remainder) loop
- Calculate b′

Prolog + epilogue loop

```
b′ = b - (b - s) % 4;

for (i = s; i < b′; i += 4) {

    statements(i);

    statements(i + 1);

    statements(i + 2);

    statements(i + 3);

}

for (i = b′; i < b; i++)

    statements(i);
```

Overhead:

- Epilogue (remainder) loop
- Calculate b′

  v1 = b - s

  v2 = v1 % 4

  b′ = b - v2

Prolog + epilogue loop

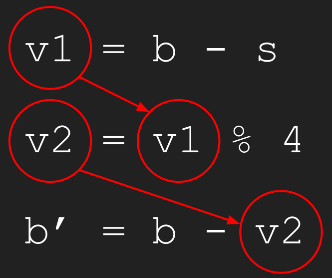
```
b' = b - (b - s) % 4;

for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}

for (i = b'; i < b; i++)
    statements(i);
```

Overhead:

- Epilogue (remainder) loop
- Calculate b'



stall!

(latency)

Prolog + epilogue loop

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

Overhead:

- Epilogue (remainder) loop
- Calculate b'

```
v1 = b - s
v2 = v1 % 4
b' = b - v2
```

Prolog + epilogue loop

- `V1 % UNROLLED_STEP` is expensive in general case
- But …

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s
v2 = v1 % 4
b′ = b - v2
```

Prolog + epilogue loop

- `V1 % UNROLLED_STEP is expensive in general case`
- `But …`
  - `If V1 >= 0`

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s

v2 = v1 % 4

b′ = b - v2
```

Prolog + epilogue loop

- `V1 % UNROLLED_STEP` is expensive in general case
- But …
  - If `V1 >= 0`
  - If `UNROLLED_STEP == 2**K`

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s
v2 = v1 % 4
b′ = b - v2
```

Prolog + epilogue loop

- V1 % UNROLLED_STEP is expensive in general case
- But …
  - If V1 >= 0
  - If UNROLLED_STEP == 2**K

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s
v2 = v1 % 4
b′ = b - v2
```

V1	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Prolog + epilogue loop

- V1 % UNROLLED_STEP is expensive in general case
- But …
  - If V1 >= 0
  - If UNROLLED_STEP == 2**K

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s
v2 = v1 % 4
b′ = b - v2
```

V1	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Prolog + epilogue loop

- V1 % UNROLLED_STEP is expensive in general case
- But …
  - If V1 >= 0
  - If UNROLLED_STEP == 2**K

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s

v2 = v1 % 4

b′ = b - v2
```

V1	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2**K - 1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Prolog + epilogue loop

- V1 % UNROLLED_STEP is expensive in general case
- But …
  - If V1 >= 0
  - If UNROLLED_STEP == 2**K

V1 % 2**K == V1 & (2**K - 1)

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
v1 = b - s
v2 = v1 % 4
b′ = b - v2
```

V1	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2**K - 1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Prolog + epilogue loop

- If s == 0

```
for (i = 0; i < b; ++i)
```

Overhead:

- Epilogue (remainder) loop
- Calculate b′

```
b′ = b - (b - s) % 4
```

Prolog + epilogue loop

- If s == 0
  `for (i = 0; i < b; ++i)`
- If b >= 0

Overhead:

- Epilogue (remainder) loop
- Calculate b′

  b′ = b − (b − s) % 4

b	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Prolog + epilogue loop

- If s == 0
  `for (i = 0; i < b; ++i)`
- If b >= 0
- If UNROLLED_STEP == 2**K

Overhead:

- Epilogue (remainder) loop
- Calculate b'

  b' = b - (b - s) % 4

b	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Prolog + epilogue loop

- If s == 0
  `for (i = 0; i < b; ++i)`
- If b >= 0
- If UNROLLED_STEP == 2**K
- b′ = b - b % 2**K

Overhead:

- Epilogue (remainder) loop
- Calculate b′

  b′ = b - (b - s) % 4

b	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
b′	0	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0

Prolog + epilogue loop

- If s == 0
  `for (i = 0; i < b; ++i)`
- If b >= 0
- If UNROLLED_STEP == 2**K
- b′ = b - b % 2**K

`b - b % 2**K == b & ~(2**K - 1)`

Overhead:

- Epilogue (remainder) loop
- Calculate b′

b′ = b - (b - s) % 4

b	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
b′	0	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0
~(2**K - 1)	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

Prolog + epilogue loop

- If s == 0
  `for (i = 0; i < b; ++i)`
- If b >= 0
- If UNROLLED_STEP == 2**K
- b′ = b - b % 2**K

`b - b % 2**K == b & ~(2**K - 1)`

Overhead:

- Epilogue (remainder) loop
- Calculate b′

b′ = b - (b - s) % 4

b	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
2**K	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
b′	0	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0
~(2**K - 1)	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

single instruction prologue!

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

Case: b - s == 100
Overhead:

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 100
Overhead:

```
N_jumps  = 99
N_checks = 101
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 100
Overhead:

```
N_jumps  = 99
N_checks = 101
```

```
N_jumps  = 25
N_checks = 27
extra prologue
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 100
Overhead:

origin loop:

```
N_jumps  = 99
N_checks = 101
```

unroll + runtime remainder:

```
N_jumps  = 25
N_checks = 27
extra prologue
```

unroll + no remainder guarantee:

```
N_jumps  = 24
N_checks = 26
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 100
Overhead:

origin loop:

```
N_jumps  = 99
N_checks = 101
```

unroll + runtime remainder:

```
N_jumps  = 25
N_checks = 27
extra prologue
```

unroll + no remainder guarantee:

```
N_jumps  = 24
N_checks = 26
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b’ = b - (b - s) % 4;
for (i = s; i < b’; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b’; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 4
Overhead:

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 4
Overhead:

```
N_jumps  = 3
N_checks = 5
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 4
Overhead:

```
N_jumps  = 3
N_checks = 5
```

```
N_jumps  = 1
N_checks = 4
extra prologue
```

origin loop

```
for (i = s; i < b; i++)
    statements(i);
```

unroll + runtime remainder

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

unroll + no remainder guarantee

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 4
Overhead:

```
N_jumps  = 3
N_checks = 5
```

```
N_jumps  = 1
N_checks = 4
extra prologue
```

```
N_jumps  = 0
N_checks = 2
```

origin loop	unroll + runtime remainder	unroll + no remainder guarantee

```
for (i = s; i < b; i++)
    statements(i);
```

```
b' = b - (b - s) % 4;
for (i = s; i < b'; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
for (i = b'; i < b; i++)
    statements(i);
```

```
for (i = s; i < b; i += 4) {
    statements(i);
    statements(i + 1);
    statements(i + 2);
    statements(i + 3);
}
```

Case: b - s == 4
Overhead:

```
N_jumps  = 3
N_checks = 5
```
+

```
N_jumps  = 1
N_checks = 4
extra prologue
```

```
N_jumps  = 0
N_checks = 2
```
+*

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

GCC 12.2
-O2
-march=skylake
-fno-tree-vectorize

CLANG 15.0
-O2
-march=skylake
-fno-vectorize
-fno-slp-vectorize

Examples. Loop (GCC vs CLANG)

```
int sum(int *a, int n) {
    int x = 0;
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

Examples. Loop (GCC vs CLANG)

x86-64 clang 15.0.0 | -O2 -march=skylake -fno-vectorize -fno-

A▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):                         # @sum(int*, int)
        test    esi, esi
        jle     .LBB0_1
        mov     esi, esi
        lea     rax, [rsi - 1]
        mov     ecx, esi
        and     ecx, 7
        cmp     rax, 7
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        and     esi, -8
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     eax, dword ptr [rdi + 4*rdx + 20]
        add     eax, dword ptr [rdi + 4*rdx + 24]
        add     eax, dword ptr [rdi + 4*rdx + 28]
        add     rdx, 8
        cmp     rsi, rdx
        jne     .LBB0_9
.LBB0_4:
        test    rcx, rcx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        xor     esi, esi
.LBB0_6:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
        cmp     rcx, rsi
        jne     .LBB0_6
.LBB0_7:
        ret
```

Examples. Loop (GCC vs CLANG)

x86-64 clang 15.0.0 | -O2 -march=skylake -fno-vectorize -fno-

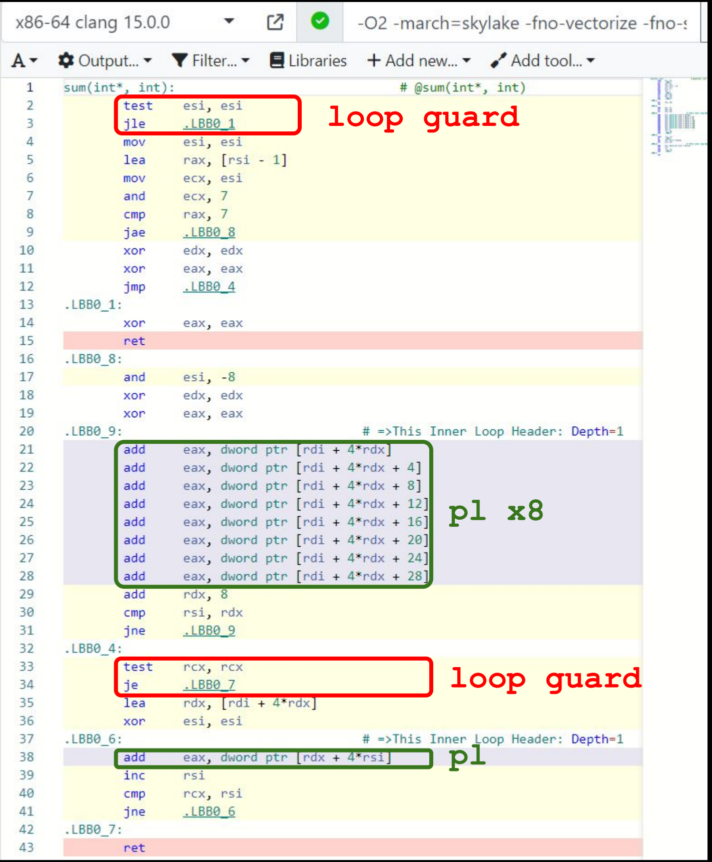
A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):                                # @sum(int*, int)
        test    esi, esi
        jle     .LBB0_1
        mov     esi, esi
        lea     rax, [rsi - 1]
        mov     ecx, esi
        and     ecx, 7
        cmp     rax, 7
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        and     esi, -8
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     eax, dword ptr [rdi + 4*rdx + 20]
        add     eax, dword ptr [rdi + 4*rdx + 24]
        add     eax, dword ptr [rdi + 4*rdx + 28]
        add     rdx, 8
        cmp     rsi, rdx
        jne     .LBB0_9
.LBB0_4:
        test    rcx, rcx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        xor     esi, esi
.LBB0_6:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
        cmp     rcx, rsi
        jne     .LBB0_6
.LBB0_7:
        ret
```

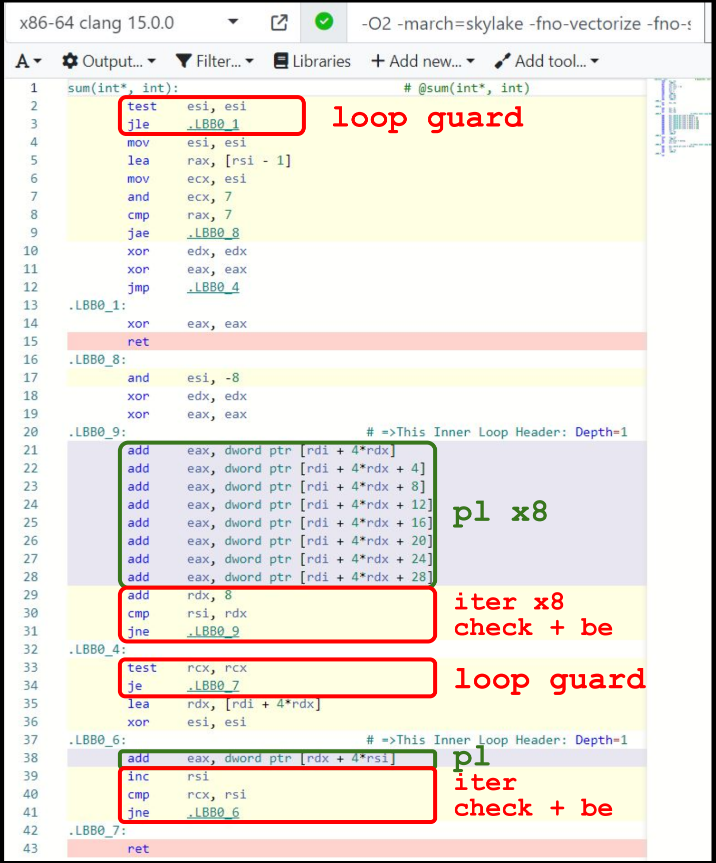
loop guard (test esi, esi / jle .LBB0_1)

loop guard (test rcx, rcx / je .LBB0_7)

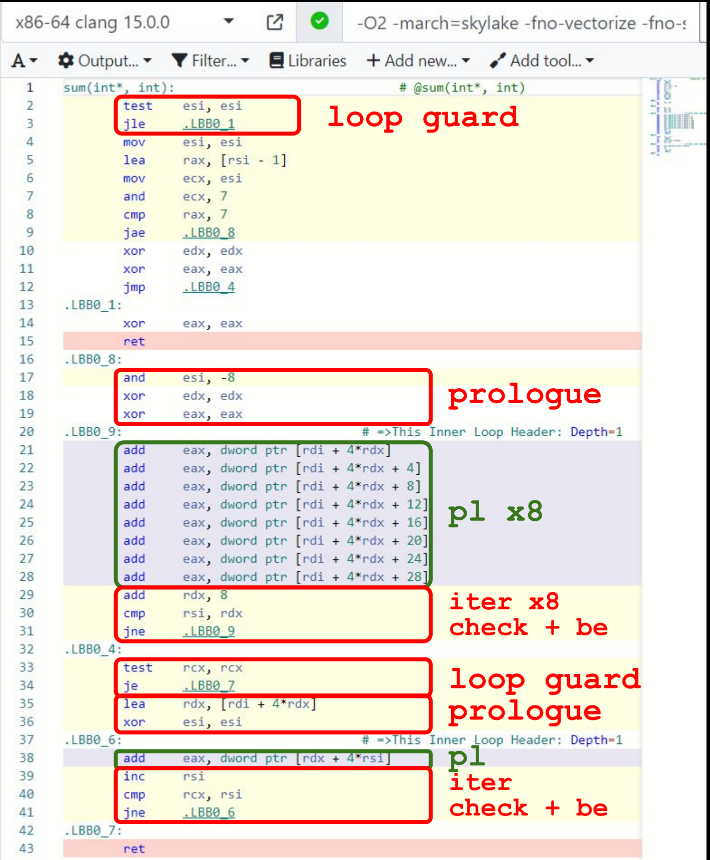
Examples. Loop (GCC vs CLANG)



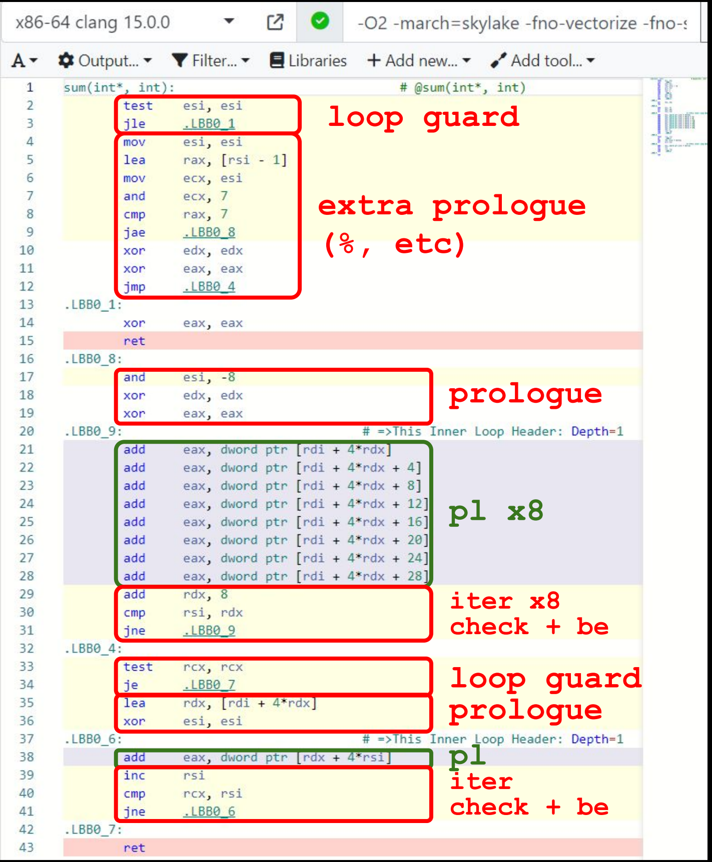
Examples. Loop (GCC vs CLANG)



Examples. Loop (GCC vs CLANG)



Examples. Loop (GCC vs CLANG)

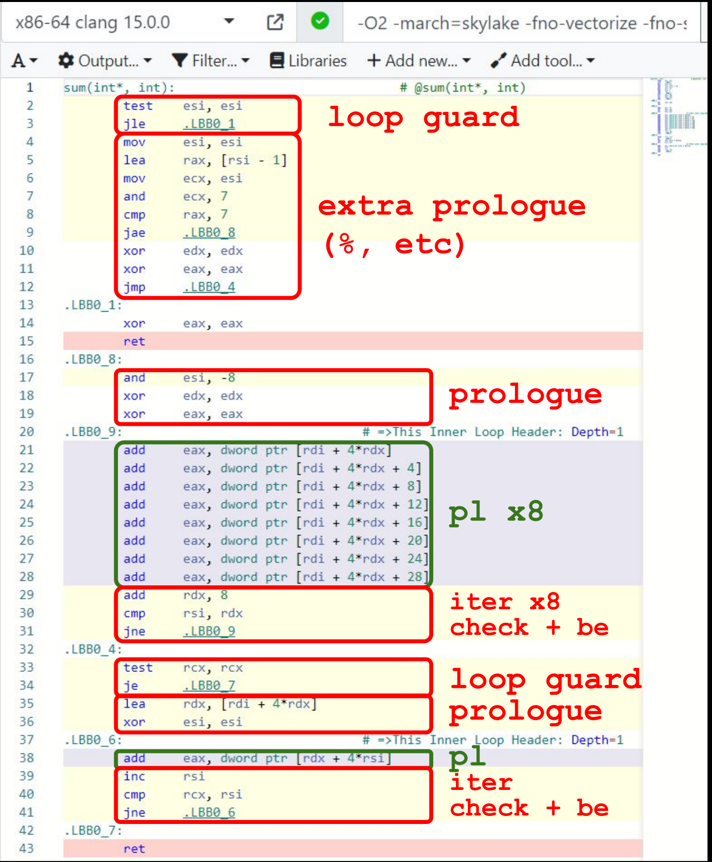


x86-64 clang 15.0.0 | -O2 -march=skylake -fno-vectorize -fno-:

```
sum(int*, int):                          # @sum(int*, int)
        test    esi, esi
        jle     .LBB0_1
        mov     esi, esi
        lea     rax, [rsi - 1]
        mov     ecx, esi
        and     ecx, 7
        cmp     rax, 7
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        and     esi, -8
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     eax, dword ptr [rdi + 4*rdx + 20]
        add     eax, dword ptr [rdi + 4*rdx + 24]
        add     eax, dword ptr [rdi + 4*rdx + 28]
        add     rdx, 8
        cmp     rsi, rdx
        jne     .LBB0_9
.LBB0_4:
        test    rcx, rcx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        xor     esi, esi
.LBB0_6:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
        cmp     rcx, rsi
        jne     .LBB0_6
.LBB0_7:
        ret
```

Examples. Loop (GCC vs CLANG)

What about GCC ?



Examples. Loop (GCC vs CLANG)

x86-64 gcc 12.2 | -O2 -march=skylake -fno-tree-

A▾ Output...▾ Filter...▾ Libraries + Add new...▾ Add tool...▾

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

x86-64 clang 15.0.0 | -O2 -march=skylake -fno-vectorize -fno-s

A▾ Output...▾ Filter...▾ Libraries + Add new...▾ Add tool...▾

```
sum(int*, int):                         # @sum(int*, int)
        test    esi, esi
        jle     .LBB0_1
        mov     esi, esi
        lea     rax, [rsi - 1]
        mov     ecx, esi
        and     ecx, 7
        cmp     rax, 7
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        and     esi, -8
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     eax, dword ptr [rdi + 4*rdx + 20]
        add     eax, dword ptr [rdi + 4*rdx + 24]
        add     eax, dword ptr [rdi + 4*rdx + 28]
        add     rdx, 8
        cmp     rsi, rdx
        jne     .LBB0_9
.LBB0_4:
        test    rcx, rcx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        xor     esi, esi
.LBB0_6:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
        cmp     rcx, rsi
        jne     .LBB0_6
.LBB0_7:
        ret
```

Annotations on the clang output:
- test/jle (lines 2–3): loop guard
- mov … jmp .LBB0_4 (lines 4–12): extra prologue (%, etc)
- and/xor/xor (lines 17–19): prologue
- 8 × add eax, dword ptr (lines 21–28): pl x8
- add rdx, 8 / cmp / jne (lines 29–31): iter x8 check + be
- test rcx, rcx / je (lines 33–34): loop guard
- lea / xor (lines 35–36): prologue
- add eax, dword ptr [rdx + 4*rsi] (line 38): pl
- inc / cmp / jne (lines 39–41): iter check + be

Examples. Loop (GCC vs CLANG)

x86-64 gcc 12.2 | -O2 -march=skylake -fno-tree-

A▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [rdi+rsi*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

Where is unrolling ?!

x86-64 clang 15.0.0 | -O2 -march=skylake -fno-vectorize -fno-s

A▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):                         # @sum(int*, int)
        test    esi, esi
        jle     .LBB0_1
        mov     esi, esi
        lea     rax, [rsi - 1]
        mov     ecx, esi
        and     ecx, 7
        cmp     rax, 7
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        and     esi, -8
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     eax, dword ptr [rdi + 4*rdx + 20]
        add     eax, dword ptr [rdi + 4*rdx + 24]
        add     eax, dword ptr [rdi + 4*rdx + 28]
        add     rdx, 8
        cmp     rsi, rdx
        jne     .LBB0_9
.LBB0_4:
        test    rcx, rcx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        xor     esi, esi
.LBB0_6:                                # =>This Inner Loop Header: Depth=1
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
        cmp     rcx, rsi
        jne     .LBB0_6
.LBB0_7:
        ret
```

Annotations on the clang listing:

- loop guard (test / jle .LBB0_1)
- extra prologue (%, etc) (mov … jmp .LBB0_4)
- prologue (and esi, -8 / xor / xor)
- pl x8 (the eight add instructions)
- iter x8 check + be (add rdx, 8 / cmp / jne)
- loop guard (test rcx, rcx / je)
- prologue (lea / xor)
- pl (add eax, dword ptr [rdx + 4*rsi])
- iter check + be (inc / cmp / jne)

GCC vs CLANG (-O2)

GCC:

- using PGO data or pragma

Clang:

- auto detect unroll and unroll count

GCC vs CLANG (-O2)

```
int sum(int *a, int n) {
    int x = 0;
    #pragma GCC unroll 4
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

force GCC to unroll loop

GCC vs CLANG (-O2)

x86-64 gcc 12.2 | -O2 -march=skylake -fno-tree-vec

A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [-4+rsi*4]
        shr     rdx, 2
        inc     rdx
        lea     rcx, [rdi+rsi*4]
        xor     eax, eax
        and     edx, 3
        je      .L3
        cmp     rdx, 1
        je      .L15
        cmp     rdx, 2
        je      .L16
        mov     eax, DWORD PTR [rdi]
        add     rdi, 4
.L16:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
.L15:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rcx
        je      .L22
.L3:
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        add     rdi, 16
        cmp     rdi, rcx
        jne     .L3
        ret
.L22:
        ret
.L4:
        xor     eax, eax
        ret
```

GCC vs CLANG (-O2)

x86-64 gcc 12.2 | -O2 -march=skylake -fno-tree-vec

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        movsx   rsi, esi
        lea     rdx, [-4+rsi*4]
        shr     rdx, 2
        inc     rdx
        lea     rcx, [rdi+rsi*4]
        xor     eax, eax
        and     edx, 3
        je      .L3
        cmp     rdx, 1
        je      .L15
        cmp     rdx, 2
        je      .L16
        mov     eax, DWORD PTR [rdi]
        add     rdi, 4
.L16:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
.L15:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rcx
        je      .L22
.L3:
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        add     rdi, 16
        cmp     rdi, rcx
        jne     .L3
        ret
.L22:
        ret
.L4:
        xor     eax, eax
        ret
```

x4

GCC vs CLANG (-O2)

GCC:

- using PGO data or pragma
- trick with jumps and body cloning

Clang:

- auto detect unroll and unroll count
- “naive”

GCC vs CLANG (-O2)

C++

```cpp
int sum(int *a, int n) {
    int x = 0;
    #pragma GCC unroll 5
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

force to unroll 5

x86-64 gcc 12.2 — -O2 -march=skylake -fno

```asm
        cmp     rdx, 2
        je      .L16
        mov     eax, DWORD PTR [rdi]
        add     rdi, 4
.L16:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
.L15:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rcx
        je      .L22
.L3:
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        add     rdi, 16
        cmp     rdi, rcx
        jne     .L3
        ret
.L22:
        ret
.L4:
        xor     eax, eax
        ret
```

x4 !

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-vector

Output... Filter... Libraries + Add new... Add tool...

```asm
        cmp     eax, 4
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        sub     esi, ecx
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inr
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     rdx, 5
        cmp     esi, edx
        jne     .LBB0_9
.LBB0_4:
        test    ecx, ecx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        mov     ecx, ecx
        xor     esi, esi
.LBB0_6:                                # =>This Inr
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
```

x5

GCC vs CLANG (-O2)

C++

```cpp
int sum(int *a, int n) {
    int x = 0;
    #pragma GCC unroll 5
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

force to unroll 5

No GCC warning!

x86-64 gcc 12.2 | -O2 -march=skylake -fno

```asm
        cmp     rdx, 2
        je      .L16
        mov     eax, DWORD PTR [rdi]
        add     rdi, 4
.L16:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
.L15:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rcx
        je      .L22
.L3:
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        add     rdi, 16
        cmp     rdi, rcx
        jne     .L3
        ret
.L22:
        ret
.L4:
        xor     eax, eax
        ret
```

x4 !

x86-64 clang 15.0.0 | -O2 -march=skylake -fno-vector

```asm
        cmp     eax, 4
        jae     .LBB0_8
        xor     edx, edx
        xor     eax, eax
        jmp     .LBB0_4
.LBB0_1:
        xor     eax, eax
        ret
.LBB0_8:
        sub     esi, ecx
        xor     edx, edx
        xor     eax, eax
.LBB0_9:                                # =>This Inr
        add     eax, dword ptr [rdi + 4*rdx]
        add     eax, dword ptr [rdi + 4*rdx + 4]
        add     eax, dword ptr [rdi + 4*rdx + 8]
        add     eax, dword ptr [rdi + 4*rdx + 12]
        add     eax, dword ptr [rdi + 4*rdx + 16]
        add     rdx, 5
        cmp     esi, edx
        jne     .LBB0_9
.LBB0_4:
        test    ecx, ecx
        je      .LBB0_7
        lea     rdx, [rdi + 4*rdx]
        mov     ecx, ecx
        xor     esi, esi
.LBB0_6:                                # =>This Inr
        add     eax, dword ptr [rdx + 4*rsi]
        inc     rsi
```

x5

GCC vs CLANG (-O2)

C++

```cpp
int sum(int *a, int n) {
    int x = 0;
    #pragma GCC unroll 4
    for (int i = 0; i < n; i += 3)
        x += a[i];
    return x;
}
```

non-power-of 2 loop step

x86-64 gcc 12.2 — -O2 -march=skylake -fno

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        xor     eax, eax
        xor     edx, edx
.L3:
        lea     rcx, [rax+3]
        add     edx, DWORD PTR [rdi+rax*4]
        cmp     esi, ecx
        jle     .L1
        lea     rax, [rcx+3]
        add     edx, DWORD PTR [rdi+rcx*4]
        cmp     esi, eax
        jle     .L1
        add     edx, DWORD PTR [rdi+rax*4]
        lea     rax, [rcx+6]
        cmp     esi, eax
        jle     .L1
        add     edx, DWORD PTR [rdi+rax*4]
        lea     rax, [rcx+9]
        cmp     esi, eax
        jg      .L3
.L1:
        mov     eax, edx
        ret
.L4:
        xor     edx, edx
        mov     eax, edx
        ret
```

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-vector

```
        ret
.LBB0_8:
        mov     edx, r8d
        and     edx, 2147483644
        xor     esi, esi
        xor     eax, eax
.LBB0_9:                                # =>This Inr
        add     eax, dword ptr [rdi + 4*rsi]
        add     eax, dword ptr [rdi + 4*rsi + 12]
        add     eax, dword ptr [rdi + 4*rsi + 24]
        add     eax, dword ptr [rdi + 4*rsi + 36]
        add     rsi, 12
        add     edx, -4
        jne     .LBB0_9
.LBB0_4:
        test    r8b, 3
        je      .LBB0_7
        lea     rdx, [rdi + 4*rsi]
        inc     cl
        movzx   ecx, cl
        and     ecx, 3
        shl     rcx, 2
        lea     rcx, [rcx + 2*rcx]
        xor     esi, esi
.LBB0_6:                                # =>This Inr
        add     eax, dword ptr [rdx + rsi]
        add     rsi, 12
        cmp     ecx, esi
        jne     .LBB0_6
.LBB0_7:
```

GCC vs CLANG (-O2)

C++

```
int sum(int *a, int n) {
    int x = 0;
    #pragma GCC unroll 4
    for (int i = 0; i < n; i += 3)
        x += a[i];
    return x;
}
```

non-power-of 2 loop step

x86-64 gcc 12.2 — -O2 -march=skylake -fno

```
sum(int*, int):
        test    esi, esi
        jle     .L4
        xor     eax, eax
        xor     edx, edx
.L3:
        lea     rcx, [rax+3]
        add     edx, DWORD PTR [rdi+rax*4]
        cmp     esi, ecx
        jle     .L1
        lea     rax, [rcx+3]
        add     edx, DWORD PTR [rdi+rcx*4]
        cmp     esi, eax
        jle     .L1
        add     edx, DWORD PTR [rdi+rax*4]
        lea     rax, [rcx+6]
        cmp     esi, eax
        jle     .L1
        add     edx, DWORD PTR [rdi+rax*4]
        lea     rax, [rcx+9]
        cmp     esi, eax
        jg      .L3
.L1:
        mov     eax, edx
        ret
.L4:
        xor     edx, edx
        mov     eax, edx
        ret
```

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-vector

```
        ret
.LBB0_8:
        mov     edx, r8d
        and     edx, 2147483644
        xor     esi, esi
        xor     eax, eax
.LBB0_9:                                # =>This Inr
        add     eax, dword ptr [rdi + 4*rsi]
        add     eax, dword ptr [rdi + 4*rsi + 12]
        add     eax, dword ptr [rdi + 4*rsi + 24]
        add     eax, dword ptr [rdi + 4*rsi + 36]
        add     rsi, 12
        add     edx, -4
        jne     .LBB0_9
.LBB0_4:
        test    r8b, 3
        je      .LBB0_7
        lea     rdx, [rdi + 4*rsi]
        inc     cl
        movzx   ecx, cl
        and     ecx, 3
        shl     rcx, 2
        lea     rcx, [rcx + 2*rcx]
        xor     esi, esi
.LBB0_6:                                # =>This Inr
        add     eax, dword ptr [rdx + rsi]
        add     rsi, 12
        cmp     ecx, esi
        jne     .LBB0_6
.LBB0_7:
```

GCC vs CLANG (-O2)

GCC:

- using PGO data or pragma
- trick with jumps and body cloning
- Power-of-2 step && unroll count
  (rejection / body cloning)

Clang:

- auto detect unroll and unroll count
- “naive”
- any step / unroll count
  (prologue cost)

Outline of the talk

- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

New opportunities

- Loop vectorization
- SLP vectorization
- Full unroll / loop deletion
- (Mem2Reg) Promote Memory To Register / (SROA) Scalar Replacement Of Aggregates
- …

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    for (int i = 0; i < 4; i++)
        x += a[i];
    return x;
}
```

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*):
        mov     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        ret
```

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```asm
sum(int*):                              # @sum(ir
        mov     eax, dword ptr [rdi + 4]
        add     eax, dword ptr [rdi]
        add     eax, dword ptr [rdi + 8]
        add     eax, dword ptr [rdi + 12]
        ret
```

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    for (int i = 0; i < 4; i++)
        x += a[i];
    return x;
}
```

compile time bounds

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```
sum(int*):
        mov     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        ret
```

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```
sum(int*):                              # @sum(ir
        mov     eax, dword ptr [rdi + 4]
        add     eax, dword ptr [rdi]
        add     eax, dword ptr [rdi + 8]
        add     eax, dword ptr [rdi + 12]
        ret
```

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    for (int i = 0; i < 4; i++)
        x += a[i];
    return x;
}
```

compile time bounds

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*):
        mov     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        ret
```

loop is deleted

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```asm
sum(int*):                              # @sum(in
        mov     eax, dword ptr [rdi + 4]
        add     eax, dword ptr [rdi]
        add     eax, dword ptr [rdi + 8]
        add     eax, dword ptr [rdi + 12]
        ret
```

loop is deleted

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    for (int i = 0; i < 8; i++)
        x += a[i];
    return x;
}
```

compile time bounds

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*):
        lea     rdx, [rdi+32]
        xor     eax, eax
.L2:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L2
        ret
```

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```asm
sum(int*):                              # @sum(i
        mov     eax, dword ptr [rdi + 4]
        add     eax, dword ptr [rdi]
        add     eax, dword ptr [rdi + 8]
        add     eax, dword ptr [rdi + 12]
        add     eax, dword ptr [rdi + 16]
        add     eax, dword ptr [rdi + 20]
        add     eax, dword ptr [rdi + 24]
        add     eax, dword ptr [rdi + 28]
        ret
```

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    for (int i = 0; i < 8; i++)
        x += a[i];
    return x;
}
```

compile time bounds

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*):
        lea     rdx, [rdi+32]
        xor     eax, eax
.L2:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L2
        ret
```

here is the loop
(no loop guard)

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```asm
sum(int*):                              # @sum(i
        mov     eax, dword ptr [rdi + 4]
        add     eax, dword ptr [rdi]
        add     eax, dword ptr [rdi + 8]
        add     eax, dword ptr [rdi + 12]
        add     eax, dword ptr [rdi + 16]
        add     eax, dword ptr [rdi + 20]
        add     eax, dword ptr [rdi + 24]
        add     eax, dword ptr [rdi + 28]
        ret
```

loop is deleted

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    #pragma GCC unroll 8
    for (int i = 0; i < 8; i++)
        x += a[i];
    return x;
}
```

compile time bounds + pragma hint

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*):
        mov     eax, DWORD PTR [rdi+4]
        add     eax, DWORD PTR [rdi]
        add     eax, DWORD PTR [rdi+8]
        add     eax, DWORD PTR [rdi+12]
        add     eax, DWORD PTR [rdi+16]
        add     eax, DWORD PTR [rdi+20]
        add     eax, DWORD PTR [rdi+24]
        add     eax, DWORD PTR [rdi+28]
        ret
```

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```asm
sum(int*):                              # @sum(i
        mov     eax, dword ptr [rdi + 4]
        add     eax, dword ptr [rdi]
        add     eax, dword ptr [rdi + 8]
        add     eax, dword ptr [rdi + 12]
        add     eax, dword ptr [rdi + 16]
        add     eax, dword ptr [rdi + 20]
        add     eax, dword ptr [rdi + 24]
        add     eax, dword ptr [rdi + 28]
        ret
```

Full unroll / loop deletion

C++

```cpp
int sum(int *a) {
    int x = 0;
    for (int i = 0; i < 1024; i++)
        x += a[i];
    return x;
}
```

compile time bounds

x86-64 gcc 12.2 — -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*):
        lea     rdx, [rdi+4096]
        xor     eax, eax
.L2:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L2
        ret
```

no unroll
no loop guard

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectoriz

```asm
sum(int*):                              # @sum(int*)
        xor     ecx, ecx
        xor     eax, eax
.LBB0_1:                                # =>This Inner
        add     eax, dword ptr [rdi + 4*rcx]
        add     eax, dword ptr [rdi + 4*rcx + 4]
        add     eax, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     eax, dword ptr [rdi + 4*rcx + 16]
        add     eax, dword ptr [rdi + 4*rcx + 20]
        add     eax, dword ptr [rdi + 4*rcx + 24]
        add     eax, dword ptr [rdi + 4*rcx + 28]
        add     eax, dword ptr [rdi + 4*rcx + 32]
        add     eax, dword ptr [rdi + 4*rcx + 36]
        add     eax, dword ptr [rdi + 4*rcx + 40]
        add     eax, dword ptr [rdi + 4*rcx + 44]
        add     eax, dword ptr [rdi + 4*rcx + 48]
        add     eax, dword ptr [rdi + 4*rcx + 52]
        add     eax, dword ptr [rdi + 4*rcx + 56]
        add     eax, dword ptr [rdi + 4*rcx + 60]
        add     rcx, 16
        cmp     rcx, 1024
        jne     .LBB0_1
        ret
```

unroll x16
no loop guard
no extra prologue

GCC vs CLANG (-O2)

GCC:

- using PGO data or pragma
- trick with jumps and body cloning
- Power-of-2 step && unroll count (rejection / body cloning)
- Less aggressive full unroll

Clang:

- auto detect unroll and unroll count
- “naive”
- any step / unroll count (prologue cost)
- More aggressive full unroll

Mem2Reg / SROA (scalar replacement of aggregates)

C++ | x86-64 clang 15.0.0 | -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

```asm
sum(int*, int):                                # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

Mem2Reg / SROA (scalar replacement of aggregates)

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

write to mem (stack)

x86-64 clang 15.0.0 — -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

```asm
sum(int*, int):                         # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

write to mem (stack)

x86-64 clang 15.0.0 | -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

```asm
sum(int*, int):                         # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

read

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

write to mem (stack)

x86-64 clang 15.0.0 -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

```asm
sum(int*, int):                         # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

read

read

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

write to mem (stack)

x86-64 clang 15.0.0 -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

```asm
sum(int*, int):                         # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

read

read

read

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

write to mem (stack)

x86-64 clang 15.0.0   -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

```
sum(int*, int):                          # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                 # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

read read read read

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

write to mem (stack)

x86-64 clang 15.0.0 -O2 -march=skylake -fno-slp-vectorize -fno-vectorize

```asm
sum(int*, int):                          # @sum(int*
        test    esi, esi
        jle     .LBB0_1
        mov     r8d, esi
        xor     r9d, r9d
        xor     esi, esi
        xor     edx, edx
        xor     eax, eax
        xor     ecx, ecx
.LBB0_3:                                 # =>This Inner L
        add     r9d, dword ptr [rdi + 4*rcx]
        add     esi, dword ptr [rdi + 4*rcx + 4]
        add     edx, dword ptr [rdi + 4*rcx + 8]
        add     eax, dword ptr [rdi + 4*rcx + 12]
        add     rcx, 4
        cmp     rcx, r8
        jb      .LBB0_3
        add     esi, r9d
        add     edx, esi
        add     eax, edx
        ret
.LBB0_1:
        xor     eax, eax
        ret
```

read

read

read

read

no writes!

Mem2Reg / SROA (scalar replacement of aggregates)

```c
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

```
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

```c
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; k += 4) {
            sums[0] += a[i];
            sums[1] += a[i + 1];
            sums[2] += a[i + 2];
            sums[3] += a[i + 3];
        }
    }

    int x = 0;
    for (int k = 0; k < 4; k += 4) {
        x += sums[0];
        x += sums[1];
        x += sums[2];
        x += sums[3];
    }
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

```c
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; k += 4) {
            sums[0] += a[i];
            sums[1] += a[i + 1];
            sums[2] += a[i + 2];
            sums[3] += a[i + 3];
        }
    }

    int x = 0;
    for (int k = 0; k < 4; k += 4) {
        x += sums[0];
        x += sums[1];
        x += sums[2];
        x += sums[3];
    }
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

```c
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    for (int i = 0; i < n; i += 4) {
        sums[0] += a[i];
        sums[1] += a[i + 1];
        sums[2] += a[i + 2];
        sums[3] += a[i + 3];
    }

    int x = 0;
    x += sums[0];
    x += sums[1];
    x += sums[2];
    x += sums[3];
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

```c
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    for (int i = 0; i < n; i += 4) {
        sums[0] += a[i];
        sums[1] += a[i + 1];
        sums[2] += a[i + 2];
        sums[3] += a[i + 3];
    }

    int x = 0;
    x += sums[0];
    x += sums[1];
    x += sums[2];
    x += sums[3];
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

```c
int sum(int *a, int n) {
    int sums_0 = 0;
    int sums_1 = 0;
    int sums_2 = 0;
    int sums_3 = 0;

    for (int i = 0; i < n; i += 4) {
        sums_0 += a[i];
        sums_1 += a[i + 1];
        sums_2 += a[i + 2];
        sums_3 += a[i + 3];
    }

    int x = 0;
    x += sums_0;
    x += sums_1;
    x += sums_2;
    x += sums_3;
    return x;
}
```

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

x86-64 gcc 12.2 | -O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*, int):
        mov     QWORD PTR [rsp-24], 0
        mov     QWORD PTR [rsp-16], 0
        xor     edx, edx
        xor     r9d, r9d
        lea     r8, [rsp-8]
        test    esi, esi
        jle     .L9
.L5:
        lea     rax, [rsp-24]
        mov     rcx, rdi
.L6:
        add     edx, DWORD PTR [rcx]
        mov     DWORD PTR [rax], edx
        add     rax, 4
        add     rcx, 4
        cmp     r8, rax
        je      .L10
        mov     edx, DWORD PTR [rax]
        jmp     .L6
.L10:
        add     r9d, 4
        mov     edx, DWORD PTR [rsp-24]
        add     rdi, 16
        cmp     esi, r9d
        jg      .L5
        mov     eax, DWORD PTR [rsp-20]
        add     eax, edx
        add     eax, DWORD PTR [rsp-16]
        add     eax, DWORD PTR [rsp-12]
        ret
.L9:
        xor     eax, eax
        ret
```

Mem2Reg / SROA (scalar replacement of aggregates)

C++

```cpp
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll 1
    for (int i = 0; i < n; i += 4) {
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];
    }

    int x = 0;
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

x86-64 gcc 12.2

-O2 -march=skylake -fno-tree-vectorize

```asm
sum(int*, int):
        mov     QWORD PTR [rsp-24], 0      ; write
        mov     QWORD PTR [rsp-16], 0      ; write
        xor     edx, edx
        xor     r9d, r9d
        lea     r8, [rsp-8]
        test    esi, esi
        jle     .L9
.L5:
        lea     rax, [rsp-24]
        mov     rcx, rdi
.L6:
        add     edx, DWORD PTR [rcx]
        mov     DWORD PTR [rax], edx       ; write
        add     rax, 4
        add     rcx, 4
        cmp     r8, rax
        je      .L10
        mov     edx, DWORD PTR [rax]
        jmp     .L6
.L10:
        add     r9d, 4
        mov     edx, DWORD PTR [rsp-24]
        add     rdi, 16
        cmp     esi, r9d
        jg      .L5
        mov     eax, DWORD PTR [rsp-20]
        add     eax, edx
        add     eax, DWORD PTR [rsp-16]
        add     eax, DWORD PTR [rsp-12]
        ret
.L9:
        xor     eax, eax
        ret
```

Outline of the talk

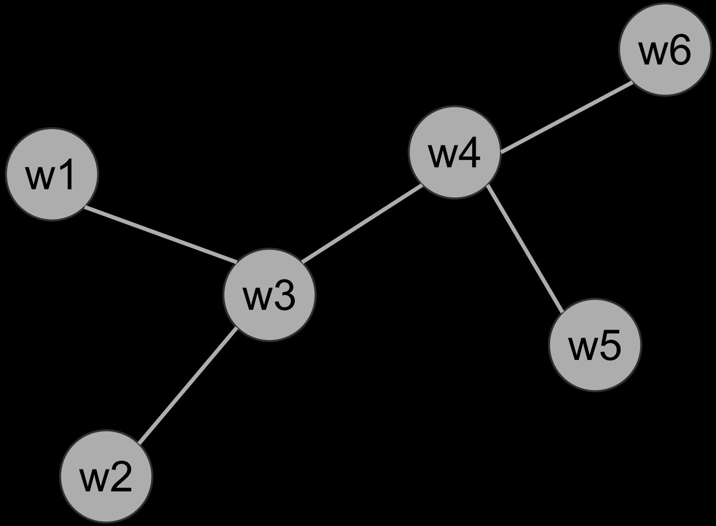
- Loop from compiler perspective
- Loop unrolling basics
- Loop unrolling overhead
- GCC and CLANG unroll details
- New optimization opportunities after unrolling
- Example

Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

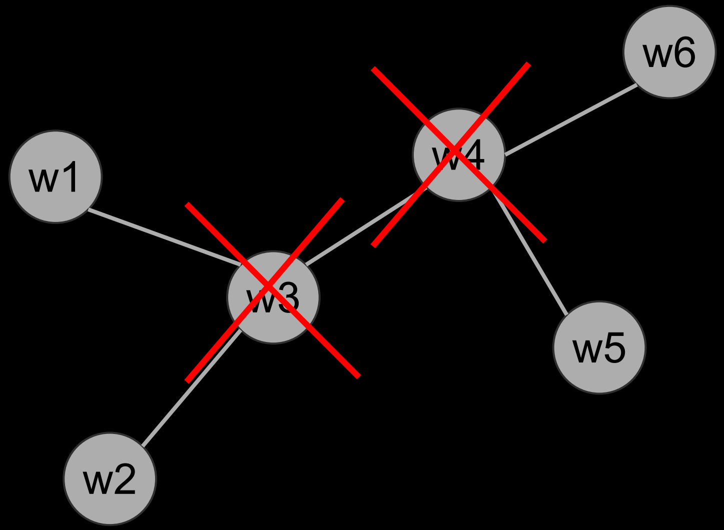
Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP



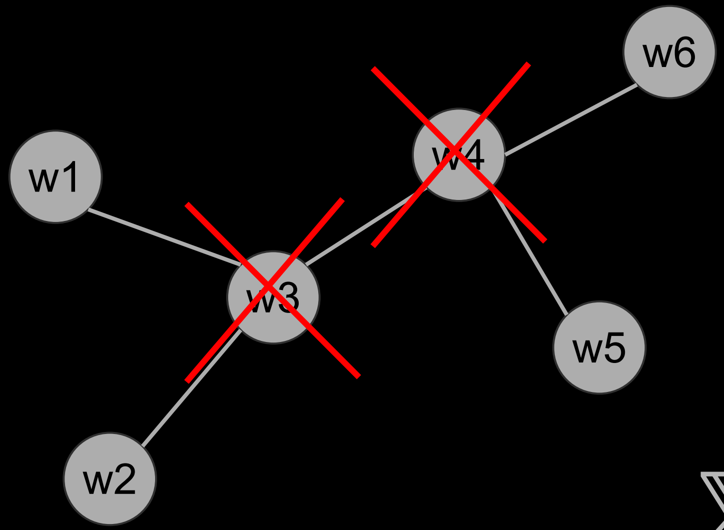
Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP



Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

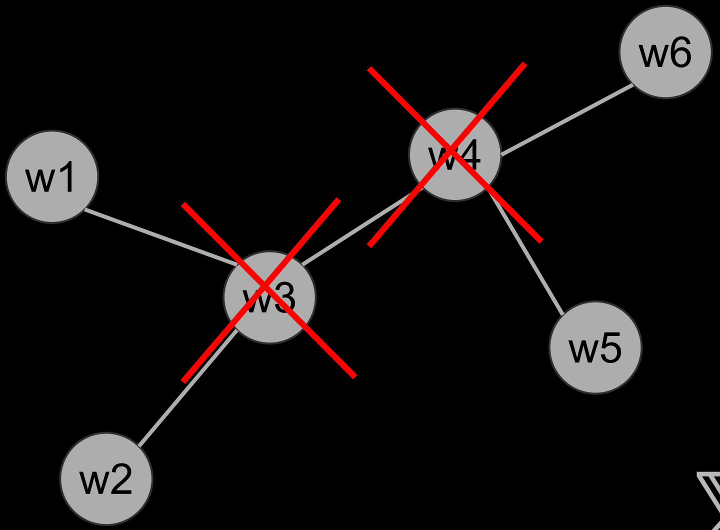


$$\sum_{\text{deleted}} wi \rightarrow \min$$

Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

NP-complete!



$$\sum_{\text{deleted}} wi \rightarrow \min$$

Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

But…

Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

But…

- Assume weights $\in$ N

Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

But…

- Assume weights $\in \mathbb{N}$
- Assume NumVertex <= 16

Example

Need to solve Minimum Weighted Vertex Cover Problem ASAP

But…

- Assume weights ∈ N
- Assume NumVertex <= 16
- Encode task as… bits:
  - vertices set is given as bitmask (uint32_t)

Example

```
uint32_t getBestSubset(
        const vector<uint32_t> &InW,
        const vector<uint32_t> &InM) {
    array<uint32_t, 16> Weights = {0};
    array<uint32_t, 16> AdjMasks = {0};
    copy(begin(InW), end(InW), begin(Weights));
    copy(begin(InM), end(InM), begin(AdjMasks));
```

Example

```
const auto GetSubsetWeight = [&](uint32_t Subset) {
  uint32_t RV = 0;
  for (size_t I = 0; I < 16; ++I)
    RV += (Subset & (1 << I)) ? Weights[I] : 0;
  return RV;
};

const auto IsCompatible = [&](uint32_t Subset) {
  uint32_t IncompatibleVerMask = 0;
  for (size_t I = 0; I < 16; ++I)
    IncompatibleVerMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
  return (Subset & IncompatibleVerMask) == 0;
};
```

Example

```
    uint32_t BestWeight = 0;
    uint32_t BestSubset = 0;
    const uint32_t NumSubsets = 1 << InW.size();
    for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {

        const uint32_t Weight = GetSubsetWeight(Subset);
        if (Weight <= BestWeight)
            continue;

        if (!IsCompatible(Subset))
            continue;

        BestWeight = Weight;
        BestSubset = Subset;
    }
    return BestSubset;
}
```

Example

C++

```cpp
        const uint32_t NumVtx) {
    array<uint32_t, N> Weights = {0};
    array<uint32_t, N> AdjMasks = {0};
    copy(begin(InW), end(InW), begin(Weights));
    copy(begin(InM), end(InM), begin(AdjMasks));

    const auto GetSubsetWeight = [&](uint32_t Subset) {
      uint32_t RV = 0;
      for (size_t I = 0; I < N; ++I)
        RV += (Subset & (1 << I)) ? Weights[I] : 0;
      return RV;
    };

    const auto IsCompatible = [&](uint32_t Subset) {
      uint32_t IncompatMask = 0;
      for (size_t I = 0; I < N; ++I)
        IncompatMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
      return (Subset & IncompatMask) == 0;
    };


    uint32_t BestWeight = 0;
    uint32_t BestSubset = 0;
    const uint32_t NumSubsets = 1 << NumVtx;
    for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {
        const uint32_t Weight = GetSubsetWeight(Subset);
        if (Weight <= BestWeight)
            continue;

        if (!IsCompatible(Subset))
            continue;

        BestWeight = Weight;
        BestSubset = Subset;
    }
    return BestSubset;
}
```

x86-64 clang 15.0.0 — -O2 -march=skylake

```asm
        vpxor   xmm6, xmm6, xmm6
        xor     eax, eax
        jmp     .LBB0_8
.LBB0_10:                               #   in Loop: Header=BB0_8 Depth=1
        inc     ecx
        shrx    esi, ecx, ebx
        test    esi, esi
        jne     .LBB0_6
.LBB0_8:                                # =>This Inner Loop Header: Depth=1
        vmovd   xmm7, ecx
        vpbroadcastd    ymm7, xmm7
        vpand   ymm8, ymm7, ymm4
        vpand   ymm7, ymm7, ymm5
        vpcmpeqd        ymm9, ymm7, ymm5
        vpand   ymm9, ymm9, ymm1
        vpcmpeqd        ymm10, ymm8, ymm4
        vpand   ymm10, ymm10, ymm0
        vpaddd  ymm9, ymm10, ymm9
        vextracti128    xmm2, ymm9, 1
        vpaddd  xmm2, xmm9, xmm2
        vpshufd xmm3, xmm2, 238                 # xmm3 = xmm2[2,3,2,3]
        vpaddd  xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85                  # xmm3 = xmm2[1,1,1,1]
        vpaddd  xmm2, xmm2, xmm3
        vmovd   esi, xmm2
        cmp     esi, edx
        jbe     .LBB0_10
        vpcmpgtd        ymm2, ymm8, ymm6
        vpand   ymm2, ymm11, ymm2
        vpcmpgtd        ymm3, ymm7, ymm6
        vpand   ymm3, ymm12, ymm3
        vpor    ymm2, ymm2, ymm3
        vextracti128    xmm3, ymm2, 1
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 238                 # xmm3 = xmm2[2,3,2,3]
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85                  # xmm3 = xmm2[1,1,1,1]
        vpor    xmm2, xmm2, xmm3
        vmovd   edi, xmm2
        test    edi, ecx
        cmove   eax, ecx
        cmove   edx, esi
        jmp     .LBB0_10
```

Example

C++

```cpp
        const uint32_t NumVtx) {
    array<uint32_t, N> Weights = {0};
    array<uint32_t, N> AdjMasks = {0};
    copy(begin(InW), end(InW), begin(Weights));
    copy(begin(InM), end(InM), begin(AdjMasks));

    const auto GetSubsetWeight = [&](uint32_t Subset) {
      uint32_t RV = 0;
      for (size_t I = 0; I < N; ++I)
        RV += (Subset & (1 << I)) ? Weights[I] : 0;
      return RV;
    };

    const auto IsCompatible = [&](uint32_t Subset) {
      uint32_t IncompatMask = 0;
      for (size_t I = 0; I < N; ++I)
        IncompatMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
      return (Subset & IncompatMask) == 0;
    };


    uint32_t BestWeight = 0;
    uint32_t BestSubset = 0;
    const uint32_t NumSubsets = 1 << NumVtx;
    for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {
        const uint32_t Weight = GetSubsetWeight(Subset);
        if (Weight <= BestWeight)
            continue;

        if (!IsCompatible(Subset))
            continue;

        BestWeight = Weight;
        BestSubset = Subset;
    }
    return BestSubset;
}
```

x86-64 clang 15.0.0 | -O2 -march=skylake

Output... Filter... Libraries Add new... Add tool...

```asm
        vpxor   xmm6, xmm6, xmm6
        xor     eax, eax
        jmp     .LBB0_8
.LBB0_10:                               #   in Loop: Header=BB0_8 Depth=1
        inc     ecx
        shrx    esi, ecx, ebx
        test    esi, esi
        jne     .LBB0_6
.LBB0_8:                                # =>This Inner Loop Header: Depth=1
        vmovd   xmm7, ecx
        vpbroadcastd    ymm7, xmm7
        vpand   ymm8, ymm7, ymm4
        vpand   ymm7, ymm7, ymm5
        vpcmpeqd        ymm9, ymm7, ymm5
        vpand   ymm9, ymm9, ymm1
        vpcmpeqd        ymm10, ymm8, ymm4
        vpand   ymm10, ymm10, ymm0
        vpaddd  ymm9, ymm10, ymm9
        vextracti128    xmm2, ymm9, 1
        vpaddd  xmm2, xmm9, xmm2
        vpshufd xmm3, xmm2, 238             # xmm3 = xmm2[2,3,2,3]
        vpaddd  xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85              # xmm3 = xmm2[1,1,1,1]
        vpaddd  xmm2, xmm2, xmm3
        vmovd   esi, xmm2
        cmp     esi, edx
        jbe     .LBB0_10
        vpcmpgtd        ymm2, ymm8, ymm6
        vpand   ymm2, ymm11, ymm2
        vpcmpgtd        ymm3, ymm7, ymm6
        vpand   ymm3, ymm12, ymm3
        vpor    ymm2, ymm2, ymm3
        vextracti128    xmm3, ymm2, 1
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 238             # xmm3 = xmm2[2,3,2,3]
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85              # xmm3 = xmm2[1,1,1,1]
        vpor    xmm2, xmm2, xmm3
        vmovd   edi, xmm2
        test    edi, ecx
        cmove   eax, ecx
        cmove   edx, esi
        jmp     .LBB0_10
```

Example

C++

```cpp
        const uint32_t NumVtx) {
    array<uint32_t, N> Weights = {0};
    array<uint32_t, N> AdjMasks = {0};
    copy(begin(InW), end(InW), begin(Weights));
    copy(begin(InM), end(InM), begin(AdjMasks));

    const auto GetSubsetWeight = [&](uint32_t Subset) {
      uint32_t RV = 0;
      for (size_t I = 0; I < N; ++I)
        RV += (Subset & (1 << I)) ? Weights[I] : 0;
      return RV;
    };

    const auto IsCompatible = [&](uint32_t Subset) {
      uint32_t IncompatMask = 0;
      for (size_t I = 0; I < N; ++I)
        IncompatMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
      return (Subset & IncompatMask) == 0;
    };


    uint32_t BestWeight = 0;
    uint32_t BestSubset = 0;
    const uint32_t NumSubsets = 1 << NumVtx;
    for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {
        const uint32_t Weight = GetSubsetWeight(Subset);
        if (Weight <= BestWeight)
            continue;

        if (!IsCompatible(Subset))
            continue;

        BestWeight = Weight;
        BestSubset = Subset;
    }
    return BestSubset;
}
```

x86-64 clang 15.0.0 | -O2 -march=skylake

Output... | Filter... | Libraries | Add new... | Add tool...

```asm
        vpxor   xmm6, xmm6, xmm6
        xor     eax, eax
        jmp     .LBB0_8
.LBB0_10:                               #   in Loop: Header=BB0_8 Depth=1
        inc     ecx
        shrx    esi, ecx, ebx
        test    esi, esi
        jne     .LBB0_6
.LBB0_8:                                # =>This Inner Loop Header: Depth=1
        vmovd   xmm7, ecx
        vpbroadcastd    ymm7, xmm7
        vpand   ymm8, ymm7, ymm4
        vpand   ymm7, ymm7, ymm5
        vpcmpeqd        ymm9, ymm7, ymm5
        vpand   ymm9, ymm9, ymm1
        vpcmpeqd        ymm10, ymm8, ymm4
        vpand   ymm10, ymm10, ymm0
        vpaddd  ymm9, ymm10, ymm9
        vextracti128    xmm2, ymm9, 1
        vpaddd  xmm2, xmm9, xmm2
        vpshufd xmm3, xmm2, 238         # xmm3 = xmm2[2,3,2,3]
        vpaddd  xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85          # xmm3 = xmm2[1,1,1,1]
        vpaddd  xmm2, xmm2, xmm3
        vmovd   esi, xmm2
        cmp     esi, edx
        jbe     .LBB0_10
        vpcmpgtd        ymm2, ymm8, ymm6
        vpand   ymm2, ymm11, ymm2
        vpcmpgtd        ymm3, ymm7, ymm6
        vpand   ymm3, ymm12, ymm3
        vpor    ymm2, ymm2, ymm3
        vextracti128    xmm3, ymm2, 1
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 238         # xmm3 = xmm2[2,3,2,3]
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85          # xmm3 = xmm2[1,1,1,1]
        vpor    xmm2, xmm2, xmm3
        vmovd   edi, xmm2
        test    edi, ecx
        cmove   eax, ecx
        cmove   edx, esi
        jmp     .LBB0_10
```

Example

C++

```cpp
    const uint32_t NumVtx) {
  array<uint32_t, N> Weights = {0};
  array<uint32_t, N> AdjMasks = {0};
  copy(begin(InW), end(InW), begin(Weights));
  copy(begin(InM), end(InM), begin(AdjMasks));

  const auto GetSubsetWeight = [&](uint32_t Subset) {
    uint32_t RV = 0;
    for (size_t I = 0; I < N; ++I)
      RV += (Subset & (1 << I)) ? Weights[I] : 0;
    return RV;
  };

  const auto IsCompatible = [&](uint32_t Subset) {
    uint32_t IncompatMask = 0;
    for (size_t I = 0; I < N; ++I)
      IncompatMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
    return (Subset & IncompatMask) == 0;
  };


  uint32_t BestWeight = 0;
  uint32_t BestSubset = 0;
  const uint32_t NumSubsets = 1 << NumVtx;
  for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {
      const uint32_t Weight = GetSubsetWeight(Subset);
      if (Weight <= BestWeight)
          continue;

      if (!IsCompatible(Subset))
          continue;

      BestWeight = Weight;
      BestSubset = Subset;
  }
  return BestSubset;
}
```

x86-64 clang 15.0.0 | -O2 -march=skylak

```asm
        vpxor   xmm6, xmm6, xmm6
        xor     eax, eax
        jmp     .LBB0_8
.LBB0_10:                               #   in L
        inc     ecx
        shrx    esi, ecx, ebx
        test    esi, esi
        jne     .LBB0_6
.LBB0_8:                                # =>This
        vmovd   xmm7, ecx
        vpbroadcastd    ymm7, xmm7
        vpand   ymm8, ymm7, ymm4
        vpand   ymm7, ymm7, ymm5
        vpcmpeqd        ymm9, ymm7, ymm5
        vpand   ymm9, ymm9, ymm1
        vpcmpeqd        ymm10, ymm8, ymm4
        vpand   ymm10, ymm10, ymm0
        vpaddd  ymm9, ymm10, ymm9
        vextracti128    xmm2, ymm9, 1
        vpaddd  xmm2, xmm9, xmm2
        vpshufd xmm3, xmm2, 238
        vpaddd  xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85
        vpaddd  xmm2, xmm2, xmm3
        vmovd   esi, xmm2
        cmp     esi, edx
        jbe     .LBB0_10
        vpcmpgtd        ymm2, ymm8, ymm6
        vpand   ymm2, ymm11, ymm2
        vpcmpgtd        ymm3, ymm7, ymm6
        vpand   ymm3, ymm12, ymm3
        vpor    ymm2, ymm2, ymm3
        vextracti128    xmm3, ymm2, 1
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 238
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85
        vpor    xmm2, xmm2, xmm3
        vmovd   edi, xmm2
        test    edi, ecx
        cmove   eax, ecx
        cmove   edx, esi
        jmp     .LBB0_10
```

no loops

Example

C++

```cpp
        const uint32_t NumVtx) {
    array<uint32_t, N> Weights = {0};
    array<uint32_t, N> AdjMasks = {0};
    copy(begin(InW), end(InW), begin(Weights));
    copy(begin(InM), end(InM), begin(AdjMasks));

    const auto GetSubsetWeight = [&](uint32_t Subset) {
      uint32_t RV = 0;
      for (size_t I = 0; I < N; ++I)
        RV += (Subset & (1 << I)) ? Weights[I] : 0;
      return RV;
    };

    const auto IsCompatible = [&](uint32_t Subset) {
      uint32_t IncompatMask = 0;
      for (size_t I = 0; I < N; ++I)
        IncompatMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
      return (Subset & IncompatMask) == 0;
    };


    uint32_t BestWeight = 0;
    uint32_t BestSubset = 0;
    const uint32_t NumSubsets = 1 << NumVtx;
    for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {
        const uint32_t Weight = GetSubsetWeight(Subset);
        if (Weight <= BestWeight)
            continue;

        if (!IsCompatible(Subset))
            continue;

        BestWeight = Weight;
        BestSubset = Subset;
    }
    return BestSubset;
}
```

x86-64 clang 15.0.0 | -O2 -march=skylak

Output... | Filter... | Libraries | Add new...

```asm
        vpxor   xmm6, xmm6, xmm6
        xor     eax, eax
        jmp     .LBB0_8
.LBB0_10:                               #   in L
        inc     ecx
        shrx    esi, ecx, ebx
        test    esi, esi
        jne     .LBB0_6
.LBB0_8:                                # =>This
        vmovd   xmm7, ecx
        vpbroadcastd    ymm7, xmm7
        vpand   ymm8, ymm7, ymm4
        vpand   ymm7, ymm7, ymm5
        vpcmpeqd        ymm9, ymm7, ymm5
        vpand   ymm9, ymm9, ymm1
        vpcmpeqd        ymm10, ymm8, ymm4
        vpand   ymm10, ymm10, ymm0
        vpaddd  ymm9, ymm10, ymm9
        vextracti128    xmm2, ymm9, 1
        vpaddd  xmm2, xmm9, xmm2
        vpshufd xmm3, xmm2, 238
        vpaddd  xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85
        vpaddd  xmm2, xmm2, xmm3
        vmovd   esi, xmm2
        cmp     esi, edx
        jbe     .LBB0_10
        vpcmpgtd        ymm2, ymm8, ymm6
        vpand   ymm2, ymm11, ymm2
        vpcmpgtd        ymm3, ymm7, ymm6
        vpand   ymm3, ymm12, ymm3
        vpor    ymm2, ymm2, ymm3
        vextracti128    xmm3, ymm2, 1
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 238
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85
        vpor    xmm2, xmm2, xmm3
        vmovd   edi, xmm2
        test    edi, ecx
        cmove   eax, ecx
        cmove   edx, esi
        jmp     .LBB0_10
```

no loops
no memory reads

Example

C++

```cpp
        const uint32_t NumVtx) {
    array<uint32_t, N> Weights = {0};
    array<uint32_t, N> AdjMasks = {0};
    copy(begin(InW), end(InW), begin(Weights));
    copy(begin(InM), end(InM), begin(AdjMasks));

    const auto GetSubsetWeight = [&](uint32_t Subset) {
      uint32_t RV = 0;
      for (size_t I = 0; I < N; ++I)
        RV += (Subset & (1 << I)) ? Weights[I] : 0;
      return RV;
    };

    const auto IsCompatible = [&](uint32_t Subset) {
      uint32_t IncompatMask = 0;
      for (size_t I = 0; I < N; ++I)
        IncompatMask |= (Subset & (1 << I)) ? AdjMasks[I] : 0;
      return (Subset & IncompatMask) == 0;
    };


    uint32_t BestWeight = 0;
    uint32_t BestSubset = 0;
    const uint32_t NumSubsets = 1 << NumVtx;
    for (uint32_t Subset = 1; Subset < NumSubsets; ++Subset) {
        const uint32_t Weight = GetSubsetWeight(Subset);
        if (Weight <= BestWeight)
            continue;

        if (!IsCompatible(Subset))
            continue;

        BestWeight = Weight;
        BestSubset = Subset;
    }
    return BestSubset;
}
```

x86-64 clang 15.0.0 | -O2 -march=skylak

Output... | Filter... | Libraries | Add new...

```asm
        vpxor   xmm6, xmm6, xmm6
        xor     eax, eax
        jmp     .LBB0_8
.LBB0_10:                               #   in L
        inc     ecx
        shrx    esi, ecx, ebx
        test    esi, esi
        jne     .LBB0_6
.LBB0_8:                                # =>This
        vmovd   xmm7, ecx
        vpbroadcastd    ymm7, xmm7
        vpand   ymm8, ymm7, ymm4
        vpand   ymm7, ymm7, ymm5
        vpcmpeqd        ymm9, ymm7, ymm5
        vpand   ymm9, ymm9, ymm1
        vpcmpeqd        ymm10, ymm8, ymm4
        vpand   ymm10, ymm10, ymm0
        vpaddd  ymm9, ymm10, ymm9
        vextracti128    xmm2, ymm9, 1
        vpaddd  xmm2, xmm9, xmm2
        vpshufd xmm3, xmm2, 238
        vpaddd  xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85
        vpaddd  xmm2, xmm2, xmm3
        vmovd   esi, xmm2
        cmp     esi, edx
        jbe     .LBB0_10
        vpcmpgtd        ymm2, ymm8, ymm6
        vpand   ymm2, ymm11, ymm2
        vpcmpgtd        ymm3, ymm7, ymm6
        vpand   ymm3, ymm12, ymm3
        vpor    ymm2, ymm2, ymm3
        vextracti128    xmm3, ymm2, 1
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 238
        vpor    xmm2, xmm2, xmm3
        vpshufd xmm3, xmm2, 85
        vpor    xmm2, xmm2, xmm3
        vmovd   edi, xmm2
        test    edi, ecx
        cmove   eax, ecx
        cmove   edx, esi
        jmp     .LBB0_10
```

no loops
no memory reads
vectorization (xmm/ymm)

Example

N = 14	time
clang O2 • unroll: off • vectorization: off	
clang O2 • unroll: on • vectorization: off	
clang O2 • unroll: on • vectorization: on	

Example

N = 14	time
clang O2 ● unroll: off ● vectorization: off	668 mcs
clang O2 ● unroll: on ● vectorization: off	
clang O2 ● unroll: on ● vectorization: on	

Example

N = 14	time
clang O2 ● unroll: off ● vectorization: off	668 mcs
clang O2 ● unroll: on ● vectorization: off	93 mcs
clang O2 ● unroll: on ● vectorization: on	

x 7

Example

N = 14	time	
clang O2 ● unroll: off ● vectorization: off	668 mcs	
clang O2 ● unroll: on ● vectorization: off	93 mcs	x 7
clang O2 ● unroll: on ● vectorization: on	26 mcs	x 26

Outcome

- Unroll is generally:
  - useful with large trip count
  - harmful with small trip count
- Trust defaults… but…
  - Clang likely unrolls by default (small trip count case suffers)
  - GCC (O2) likely does not unroll by default. Use PGO / pragma.
- Step and unroll count should be power of 2
- Provide a hint for full unroll for GCC (O2)

Thank you

```c
int sum(int *a, int s, int b) {

    int x = 0;

    // #pragma GCC unroll 8

    for (int i = s; i < b; ++i)

        x += a[i];

    return x;

}
```

```c
int sum(int *a, int n) {
    int x = 0;
    // #pragma GCC unroll 8
    for (int i = 0; i < n; ++i)
        x += a[i];
    return x;
}
```

```c
int sum(int *a, int n) {
    int sums[4] = {0, 0, 0, 0};

    #pragma GCC unroll(1)
    for (int i = 0; i < n; i += 4) {
        #pragma GCC unroll(4)
        for (int k = 0; k < 4; ++k)
            sums[k] += a[k + i];

    }

    int x = 0;
    #pragma GCC unroll(4)
    for (int k = 0; k < 4; ++k)
        x += sums[k];
    return x;
}
```

-O2 -march=skylake -fno-tree-vectorize

-O2 -march=skylake -fno-vectorize -fno-slp-vectorize