Optimizing Switch Statements in GCC

Overview and What's New

Filip Kastl

SUSE Labs / FOSDEM 2025

1st Feb, 2025

Definitions

```
switch (i)
{
    case 0:
        // do thing A
        break;
    case 1:
        // do thing B
        break;
    case 2:
    case 3:
        // do thing C
        break;
    case 6:
        // do thing D
        break;
    default:
        // do thing E
}
```

- `i` is the index variable
- Here, the case range is 0 to 6
- Notice the "hole" between 3 and 6
  - The case range size is 7
  - but the number of cases is 5

The naive way

```
switch (i)
{
    case 0:
        // do thing A
        break;
    case 1:
        // do thing B
        break;
    case 2:
    case 3:
        // do thing C
        break;
    case 6:
        // do thing D
        break;
    default:
        // do thing E
}
```

```
if (i == 0)
    // do thing A
else if (i == 1)
    // do thing B
else if (i >= 2 && i <= 3)
    // do thing C
else if (i == 6)
    // do thing D
else
    // do thing E
```

What will happen to these switches?

Switch 1

```
switch (i)
{
    case 1: x = 13; break;
    case 2: x = 3; break;
    case 4: x = 4; break;
    case 8: x = 2; break;
    case 16: x = 1; break;
    case 32: x = 1; break;
    case 64: x = 8; break;
    default: x = -1;
}
```

Switch 2

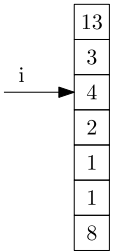
```
switch (i)
{
    case 1: x = 0; break;
    case 2: x = 1; break;
    case 4: x = 2; break;
    case 8: x = 3; break;
    case 16: x = 4; break;
    case 32: x = 5; break;
    case 64: x = 6; break;
    default: x = -1;
}
```

Switch conversion

- Contributed by Martin Jambor in 2008
- Prerequisites: Switch just assigns constants, not many holes

```
switch (i)
{
    case 0: x = 13; break;
    case 1: x = 3; break;
    case 2: x = 4; break;
    case 3: x = 2; break;
    case 4: x = 1; break;
    case 5: x = 1; break;
    case 6: x = 8; break;
    default: x = -1;
}
```

Optimized:




```
if (i >= 0 && i <= 6)
    x = CSWTCH.1[i];
else
    x = -1;
```

Switch conversion: linear transformation

```
switch (i)
{
    case 0: x = 5; break;
    case 1: x = 8; break;
    case 2: x = 11; break;
    case 3: x = 14; break;
    case 4: x = 17; break;
    case 5: x = 20; break;
    case 6: x = 23; break;
    default: x = -1;
}
```

Switch conversion: linear transformation

- Prerequisites: Switch just assigns constants, no holes, constants follow a linear function of `i`

```
switch (i)
{
    case 0: x = 5; break;
    case 1: x = 8; break;
    case 2: x = 11; break;
    case 3: x = 14; break;
    case 4: x = 17; break;
    case 5: x = 20; break;
    case 6: x = 23; break;
    default: x = -1;
}
```

- Expression $(i \cdot a) + b$
- Here $a = 3, b = 5$

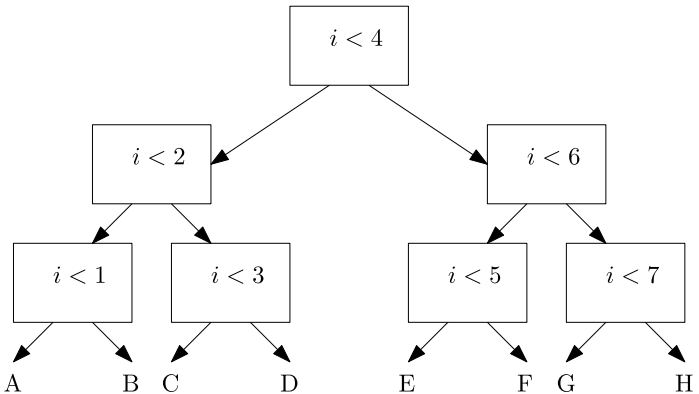
Optimized:

```
if (i >= 0 && i <= 6)
    _tmp1 = i * 3;
    x = _tmp1 + 5;
else
    x = -1;
```

Switch lowering: decision tree

```
switch (i)
{
    case 0:
        // do thing A
        break;
    case 1:
        // do thing B
        break;
    case 2:
        // do thing C
        break;
    case 3:
        // do thing D
        break;
    ...
    case 7:
        // do thing H
        break;
}
```

- Prerequisites: None
- The fallback technique



Switch lowering: jump tables

- Prerequisites: Not many holes

```
switch (i)
{
    case 0:
        // Do thing A
        break;
    case 1:
        // Do thing B
        break;
    case 2:
        // Do thing C
        break;
    case 3:
        // Do thing D
        break;
}
```

Lowered (optimized):

```
# Assume i between 0 and 3
    jmp     *.L10(,%rdi,8)
.L10:
    .quad   .L11
    .quad   .L12
    .quad   .L13
    .quad   .L14
.L11:
    # Do thing A
.L12:
    # Do thing B
.L13:
    # Do thing C
.L14:
    # Do thing D
```

Switch lowering: bit tests

- Prerequisites: Many holes, many cases pointing to the same code
- Contributed by Roger Sayle in 2003, he credits Honza Hubička and Andi Kleen

```
switch (i)
{
    case 0:
    case 3:
    case 5:
    case 8:
    case 10:
    case 13:
    case 15:
        // Do thing A
        break;
    default:
        // Do thing B
}
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	0	0	1	0	1	0	0	1

$\rightarrow 0b1010010100101001 = 42281$

Optimized:

```
_tmp1 = 42281 >> i;
_tmp2 = _tmp1 & 1;
if (_tmp2)
    // Do thing A
else
    // Do thing B
```

Switch conversion: exponential transform

Switch 1

```
switch (i)
{
    case 1: x = 13; break;
    case 2: x = 3; break;
    case 4: x = 4; break;
    case 8: x = 2; break;
    case 16: x = 1; break;
    case 32: x = 1; break;
    case 64: x = 8; break;
    default: x = -1;
}
```

Switch conversion: exponential transform

- Prerequisites: Switch just assigns constants, case numbers are powers of 2

Switch 1

```
switch (i)
{
    case 1: x = 13; break;
    case 2: x = 3; break;
    case 4: x = 4; break;
    case 8: x = 2; break;
    case 16: x = 1; break;
    case 32: x = 1; break;
    case 64: x = 8; break;
    default: x = -1;
}
```

```
// Assuming i is a power of 2
_tmp = log2(i);
switch (_tmp)
{
    case 0: x = 13; break;
    case 1: x = 3; break;
    case 2: x = 4; break;
    case 3: x = 2; break;
    case 4: x = 1; break;
    case 5: x = 1; break;
    case 6: x = 8; break;
    default: x = -1;
}
```

Switch conversion: exponential transform

- Prerequisites: Switch just assigns constants, case numbers are powers of 2

Switch 1

```
switch (i)
{
    case 1: x = 13; break;
    case 2: x = 3; break;
    case 4: x = 4; break;
    case 8: x = 2; break;
    case 16: x = 1; break;
    case 32: x = 1; break;
    case 64: x = 8; break;
    default: x = -1;
}
```

```
// Assuming i is a power
// of 2 and log2(i) is
// between 0 and 6
_tmp = log2(i);
x = CSWTCH.1[_tmp];
```

Switch conversion: exponential transform + linear transform

Switch 2

```
switch (i)
{
    case 1: x = 0; break;
    case 2: x = 1; break;
    case 4: x = 2; break;
    case 8: x = 3; break;
    case 16: x = 4; break;
    case 32: x = 5; break;
    case 64: x = 6; break;
    default: x = -1;
}
```

Switch conversion: exponential transform + linear transform

Switch 2

```
switch (i)
{
    case 1: x = 0; break;
    case 2: x = 1; break;
    case 4: x = 2; break;
    case 8: x = 3; break;
    case 16: x = 4; break;
    case 32: x = 5; break;
    case 64: x = 6; break;
    default: x = -1;
}
```

```
// Assuming i is a power of 2
_tmp = log2(i);
switch (_tmp)
{
    case 0: x = 0; break;
    case 1: x = 1; break;
    case 2: x = 2; break;
    case 3: x = 3; break;
    case 4: x = 4; break;
    case 5: x = 5; break;
    case 6: x = 6; break;
    default: x = -1;
}
```

Switch conversion: exponential transform + linear transform

Switch 2

```
switch (i)
{
    case 1: x = 0; break;
    case 2: x = 1; break;
    case 4: x = 2; break;
    case 8: x = 3; break;
    case 16: x = 4; break;
    case 32: x = 5; break;
    case 64: x = 6; break;
    default: x = -1;
}
```

```
// Assuming i is a power
// of 2 and log2(i)
// between 0 and 6
x = log2(i);
```

Future work

1) Teach switch conversion about ranges

```
switch (i % 4) {
    case 0: x = 0; break;
    case 1: x = 1; break;
    case 2: x = 2; break;
    case 3: x = 3; break;
    default: x = -1;
}
```

...does not currently simplify to

```
x = i % 4;
```

2) Lowering algorithms

- Finding bit test opportunities in $O(n)$
- Working around $\Omega(n^2)$-ness of finding jump table opportunities (Bugzilla pr118353)

Recap

- Switch conversion (future work: value ranges)
  - Linear transform
  - Exponential transform (new)
- Switch lowering
  - Decision trees
  - Jump tables (future work: ~~$\Omega(n^2)$~~)
  - Bit tests (future work: $O(n)$)

Btw, you can find all of this in the GCC source file `gcc/tree-switch-conversion.cc`

Further reading

- https://xoranth.net/gcc-switch/
- Sayle, Roger. (2008). A Superoptimizer Analysis of Multiway Branch Code Generation.

You can reach me at filip.kastl@suse.com