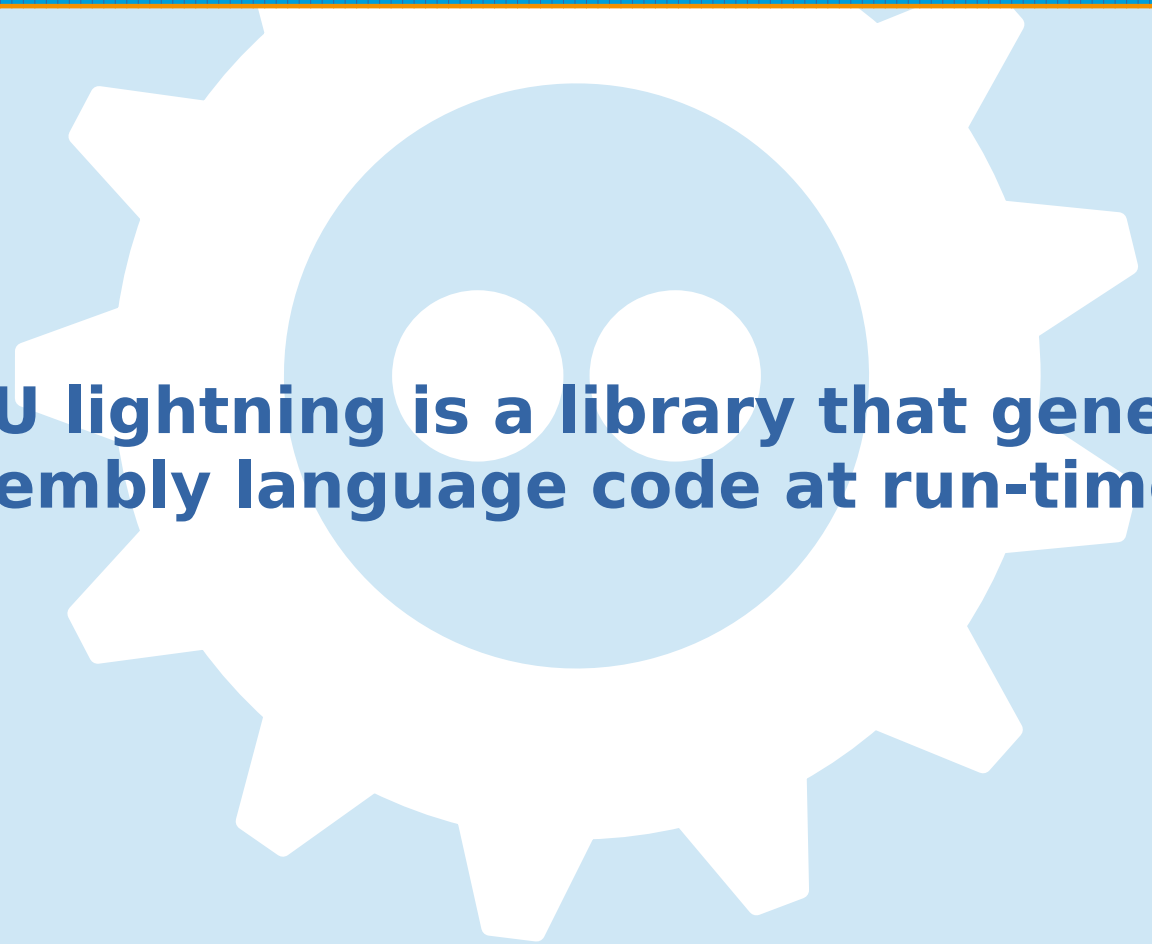
Cross-platform compilers with
GNU Lightning

Paul Cercueil

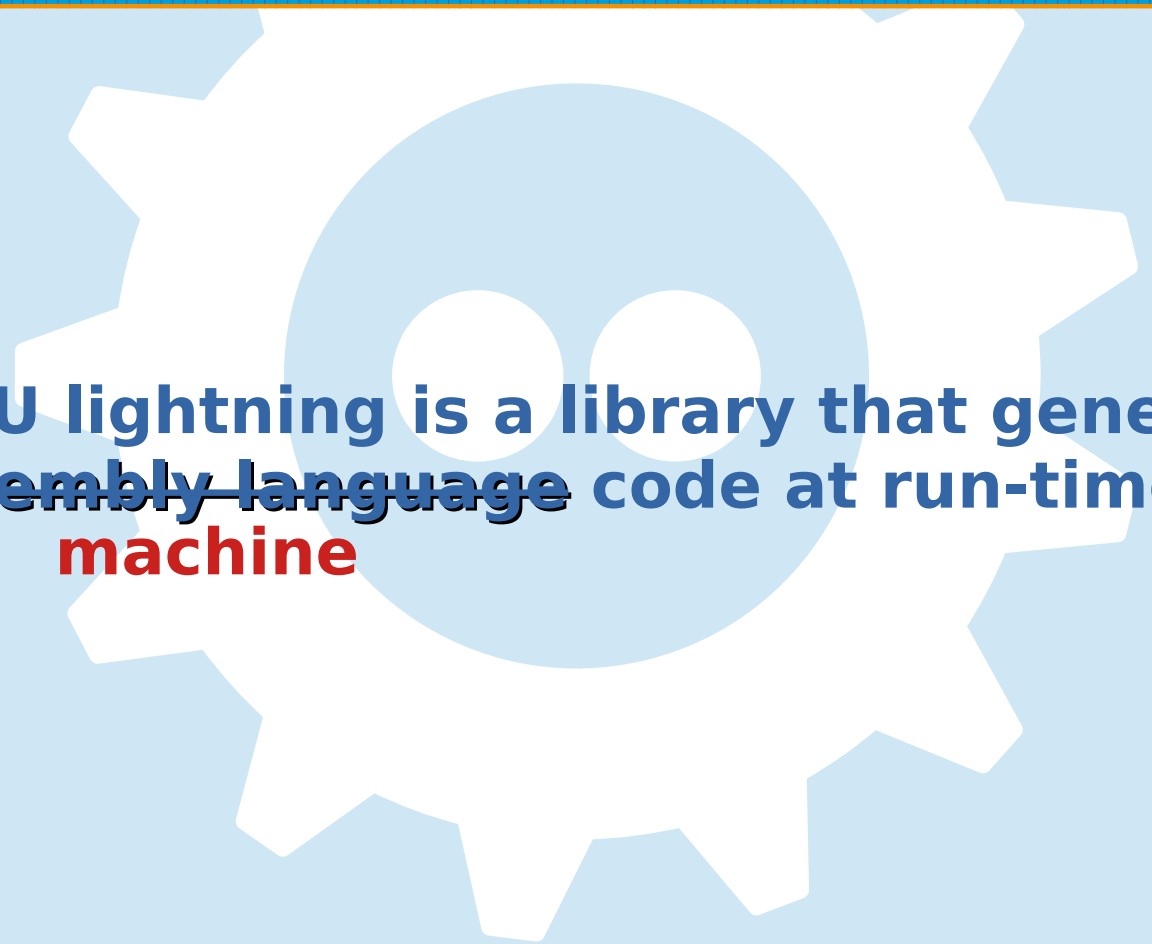
FOSDEM '25

https://gnu.org/software/lightning/



« GNU lightning is a library that generates assembly language code at run-time ».

https://gnu.org/software/lightning/



« GNU lightning is a library that generates ~~assembly language~~ machine code at run-time ».

What is GNU Lightning?

- Code generator part of a JIT engine
- Virtual CPU, with virtual registers and virtual opcodes
- Cross-platform:
  - ARM, Alpha, HPPA, IA64, Loongarch, MIPS, PowerPC, RISC-V, s390, SPARC, SuperH, x86
  - 32-bit and 64-bit
  - Little-endian and big-endian
- LGPLv3, GNU C, no dependencies

What is GNU Lightning?

- Created in 2000 by Paolo Bonzini
- JIT engine for GNU Smalltalk, and CLisp
- Moved to git in 2008
- Maintainer since 2012 is Paulo Cesar Pereira de Andrade
- Currently at version 2.2.3

Lightrec

- MIPS-to-everything JIT engine for Playstation 1 emulators
- Uses GNU Lightning

- <ad>See my talk tomorrow:

  « Writing a dynarec, step by step »</ad>

Why Lightning?

- LLVM / libgccjit don’t box in the same weight class
  - They were designed for langages
    - High-level concepts (e.g. variables)
    - Language-focused algorithms
  - They were designed for ahead-of-time
    - Powerful optimizations
    - Generate very fast code… slowly
    - Lightning generates unoptimized code… very fast
- Different tools for different applications.

API overview

- *Minimum* of 6 GP + 6 FP registers:
  - 3 caller-saved: JIT_R0 → JIT_R2 (up to JIT_R(JIT_R_NUM))
  - 3 callee-saved: JIT_V0 → JIT_V2 (up to JIT_V(JIT_V_NUM))
  - 6 floating-point: JIT_F0 → JIT_F5 (up to JIT_F(JIT_F_NUM))
- Simple register allocator (that you don’t need):
  - ```
    u8 reg = jit_get_reg(jit_class_gpr);
    jit_unget_reg(reg);
    ```
    ```

API overview

- « virtual opcodes » for binary operations, arithmetic, boolean, branching, memory I/O, floating-point math, function calls and function prolog/epilog

- Each instruction is composed of:
  - An operation, like « mul » or « sub »
  - Most times, a register/immediate flag (r or i)
  - A type identifier, when applicable.
    No type suffix = pointer-sized

API overview

```
jit_<op><r|i>[_<type>](O1, O2, O3)
```

addr	_f	_d	O1 = O2 + O3
addi	_f	_d	O1 = O2 + O3
addxr			O1 = O2 + (O3 + carry)
addxi			O1 = O2 + (O3 + carry)
addcr			O1 = O2 + O3, set carry
addci			O1 = O2 + O3, set carry
subr	_f	_d	O1 = O2 - O3
subi	_f	_d	O1 = O2 - O3
subxr			O1 = O2 - (O3 + carry)
subxi			O1 = O2 - (O3 + carry)
subcr			O1 = O2 - O3, set carry
subci			O1 = O2 - O3, set carry
rsbr	_f	_d	O1 = O3 - O1
rsbi	_f	_d	O1 = O3 - O1
mulr	_f	_d	O1 = O2 * O3
muli	_f	_d	O1 = O2 * O3
hmulr	_u		O1 = ((O2 * O3) >> WORDSIZE)
hmuli	_u		O1 = ((O2 * O3) >> WORDSIZE)
divr	_u	_f _d	O1 = O2 / O3
divi	_u	_f _d	O1 = O2 / O3
remr	_u		O1 = O2 % O3
remi	_u		O1 = O2 % O3
andr			O1 = O2 & O3
andi			O1 = O2 & O3
orr			O1 = O2 | O3
ori			O1 = O2 | O3
xorr			O1 = O2 ^ O3
xori			O1 = O2 ^ O3
lshr			O1 = O2 << O3
lshi			O1 = O2 << O3
rshr	_u		O1 = O2 >> O3(1)
rshi	_u		O1 = O2 >> O3(2)
lrotr			O1 = (O2 << O3) | (O3 >> (WORDSIZE - O3))
lroti			O1 = (O2 << O3) | (O3 >> (WORDSIZE - O3))
rrotr			O1 = (O2 >> O3) | (O3 << (WORDSIZE - O3))
rroti			O1 = (O2 >> O3) | (O3 << (WORDSIZE - O3))
movzr			O1 = O3 ? O1 : O2
movnr			O1 = O3 ? O2 : O1

API overview

Forward branch

```
jit_node_t *n;

n = jit_beqi(JIT_R0, 0);

...

jit_patch(n);
```

Backwards branch

```
jit_node_t *n1, *n2;

n1 = jit_label();

...

n2 = jit_beqi(JIT_R0, 0);

jit_patch_at(n2, n1);
```

API overview

- Function prolog / epilog:
  `jit_prolog()`
  `jit_epilog()`

- Code generation:
  `void (*fn)() = jit_emit();`

- Code disassembly:
  `jit_disassemble()`

Example of generated code

« **jit_addr(JIT_R0, JIT_R1, JIT_R2);** »

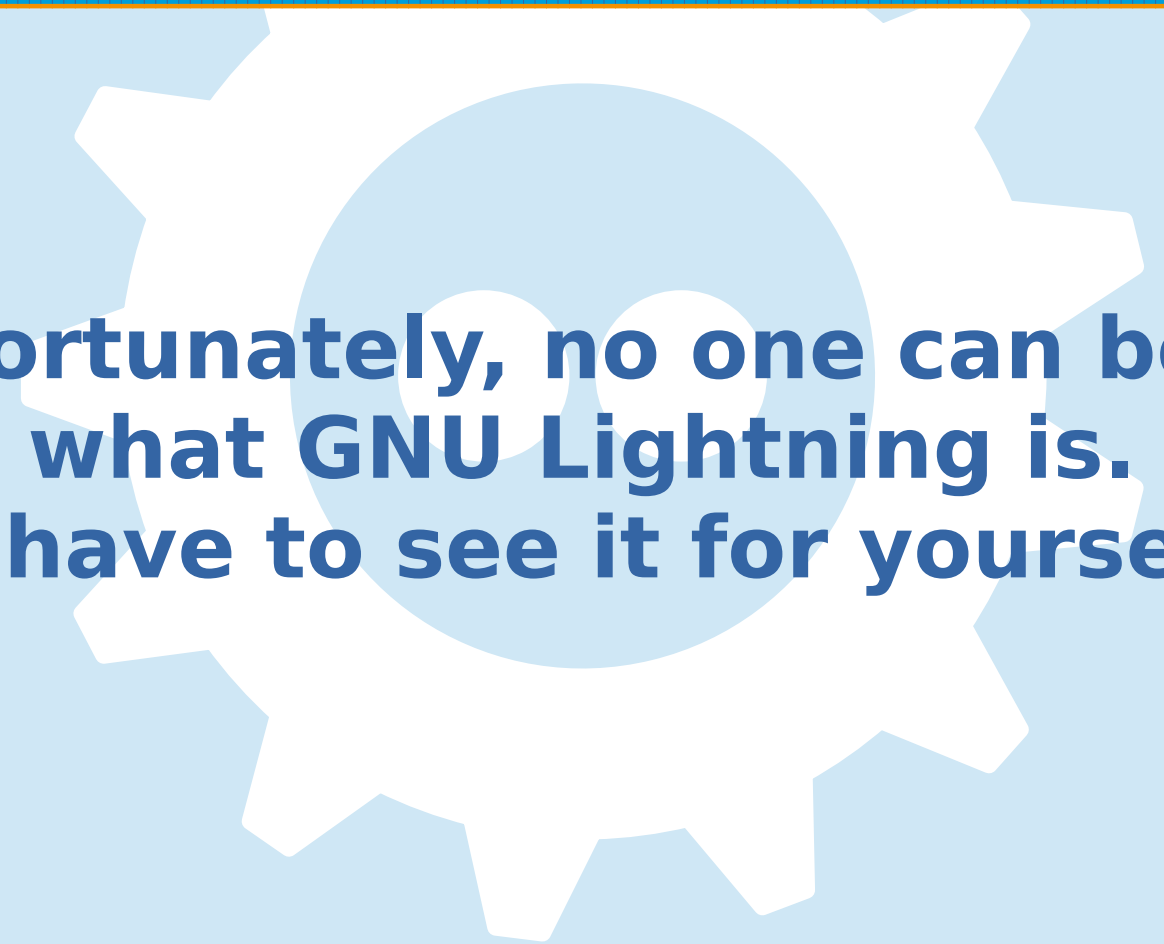
```
x86_64:
  lea   (%r10,%r11,1),%rax

MIPS64:
  daddu v0,v1,t4

PowerPC32:
  add   r28,r29,r30

SH4:
  mov   r2,r1
  add   r3,r1
```

Demo



« Unfortunately, no one can be told what GNU Lightning is. You have to see it for yourself. »

Demo

- Imagine a scripting langage called « MindBlown »
  - 32K memory cells, each signed 32-bit
  - < / > : switches to the left/right cell
  - + / - : increment/decrement value at current cell
  - [/] : repeat until the current cell is zero
  - . : Print the character of the current cell