A new constant expression interpreter for Clang

October 2024

Red Hat

2019 LLVM DEV MEETING

The new Clang constant interpreter

Constants

It's a good idea to compute them

```
int a = 2147483647 + 2147483647;
```

What?

- Compile constexpr/consteval functions to bytecode
- Interpret bytecode
- Profit
- Expressions still get interpreted directly, no bytecode involved.

Basics

- `clang::interp::Context` keeps a `Program` around, a reference to the `ASTContext`, etc.

- `isPotentialConstantExpr(const FunctionDecl *)`
- `evaluateAsRValue(const Expr *)`
- `evaluate(const Expr *)`
- `evaluateAsInitializer(const VarDecl *)`

Basics

```
constexpr unsigned sum() {
  unsigned S = 0;
  for (unsigned I : (unsigned[]){1,2,3})
    S += I;
  return S;
}
```

```
clang -c -std=c++20 test.cpp
```

Red Hat

Basics

1. `0, 1, 2, 3, 3, 2, 1`
2. `__range1`
3. `3`
4. `__range1 + 3`
5. `__begin1 != __end1`
6. `++__begin1`
7. `*__begin1`
8. `S += I`
9. `sum()`
10. `__begin1 != __end1 (fallthrough warning, via CFG)`

Basics

- `interp::Program` keeps global variables and functions
- `interp::Function` is bytecode + information about parameters, etc.
- `interp::Compiler<Emitter>` compiles functions to bytecode
- `interp::InterpStack`
- `interp::InterpFrame` holds function frame information, like position of the parameters on the stack, frame size, etc.

Types

- Primitives: PT_Bool, PT_Uint8, ….PT_Sint64, PT_Float, PT_Ptr, PT_MemberPtr, PT_FnPtr, …
- Vectors and _Complex types: arrays of primitives
- Classes/Structs: `Record` with information about field offsets, base offsets, etc.

(Block)Pointers

- Backed by an `interp::Block`
- Block: allocated data, either in the `InterpFrame` for local variables or in the `Program` for global variables.
- `Descriptor` describes the contents of a block
- `InlineDescriptor` precedes fields and saves metadata about the field (e.g. if it has been initialized)

Pointers: Example

```
struct Player {
   int x, y;
   float health;
};
```

- Player
  - Field x: Offset 16
  - FIeld y: Offset 40
  - Field health: Offset 64
  - Size: 96

Pointers: Example

- ⬛⬛ InlineDescriptor for x
- 🟦 x
- ⬛⬛ InlineDescriptor for y
- 🟦 y
- ⬛⬛ InlineDescriptor for health
- 🟦🟦🟦🟦 health

Red Hat

Bytecode: Example

```
struct Player {
  int x, y;
  float health;

  consteval float getHealth() const {
    return health;
  }
};

constexpr Player P{10, 50, 30.0};
static_assert(P.getHealth() == 30);
```

Bytecode: Example

```
Player::getHealth 0x7ccd65e30b80
[...]
       0  This
       8  GetPtrFieldPop      64
      24 LoadPopFloat
      32 RetFloat
      40 NoRet
```

- `(Block) 0x7d407acbdd28 {rootptr(8), 8, 104}`

Red Hat

Bytecode: Example

```
Player::getHealth 0x7ccd65e30b80
[...]
➡️      0  This
        8  GetPtrFieldPop      64
       24 LoadPopFloat
       32 RetFloat
       40 NoRet
```

- (Block) 0x7d407acbdd28 {rootptr(8), 8, 104}
- (Block) 0x7d407acbdd28 {rootptr(8), 8, 104}

Red Hat

Bytecode: Example

```
Player::getHealth 0x7ccd65e30b80
[...]
       0  This
➡️     8  GetPtrFieldPop      64
      24  LoadPopFloat
      32  RetFloat
      40  NoRet
```

- (Block) 0x7d407acbdd28 {72, 72, 104}
- (Block) 0x7d407acbdd28 {rootptr(8), 8, 104}

Red Hat

Bytecode: Example

```
Player::getHealth 0x7ccd65e30b80
[...]
       0  This
       8  GetPtrFieldPop      64
➡️     24 LoadPopFloat
       32 RetFloat
       40 NoRet
```

- 30.0
- (Block) 0x7d407acbdd28 {rootptr(8), 8, 104}

Red Hat

Bytecode: Example

```
Player::getHealth 0x7ccd65e30b80
[...]
       0  This
       8  GetPtrFieldPop     64
      24  LoadPopFloat
➡️    32  RetFloat
      40  NoRet
```

- 30.0

Red Hat

Bytecode: Example

```
static_assert(P.getHealth() == 30);
```

- 30.0
- 30.0

Red Hat

Bytecode: Example

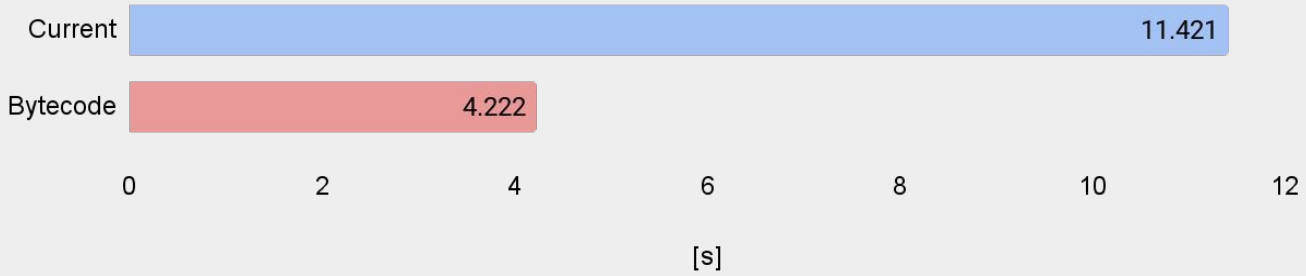
```
static_assert(P.getHealth() == 30);
```

- Final result is `true`
- Gets converted to an `APValue`
- … and returned from `evaluateAsRValue()`

(Silly) Performance Measurements

Empty loops (#61425)

```
constexpr unsigned A = []() {
  for (unsigned int n = 0; n != 10'000'000; ++n {}
}();
```



Red Hat

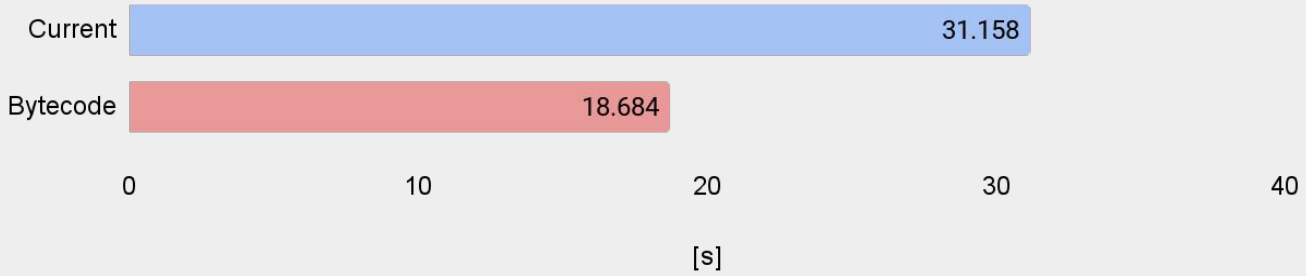
fib(40)

```
constexpr unsigned fib(unsigned N) {
  if (N < 2)
    return 1;
  return fib(N - 2) + fib(N - 1);
}
constexpr unsigned Fib100 = fib(40);
```

Red Hat

checksums

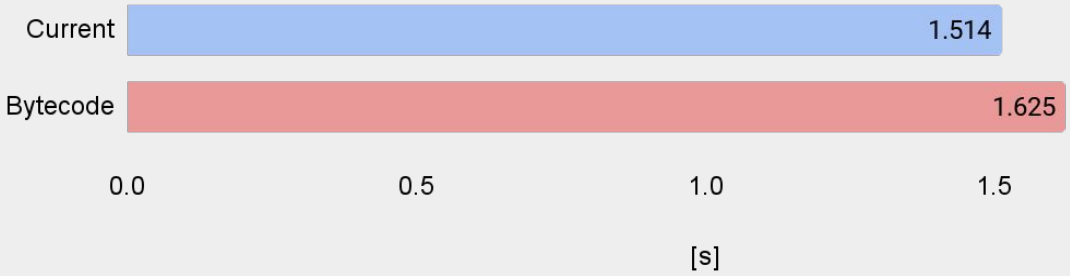
```
constexpr char str[] = {
#embed "sqlite3.c" suffix(,0)
};
consteval unsigned checksum(const char *s) {
  unsigned result = 0;
  for (const char *p = s; *p != '\0'; ++p)
    result += *p;
  return result;
}
constexpr auto sqliteChecksum = checksum(str);
```



Red Hat

sqlite3

```
$ bin/clang -c sqlite3.c
$ bin/clang -c sqlite3.c -fexperimental-new-constant-interpreter
```

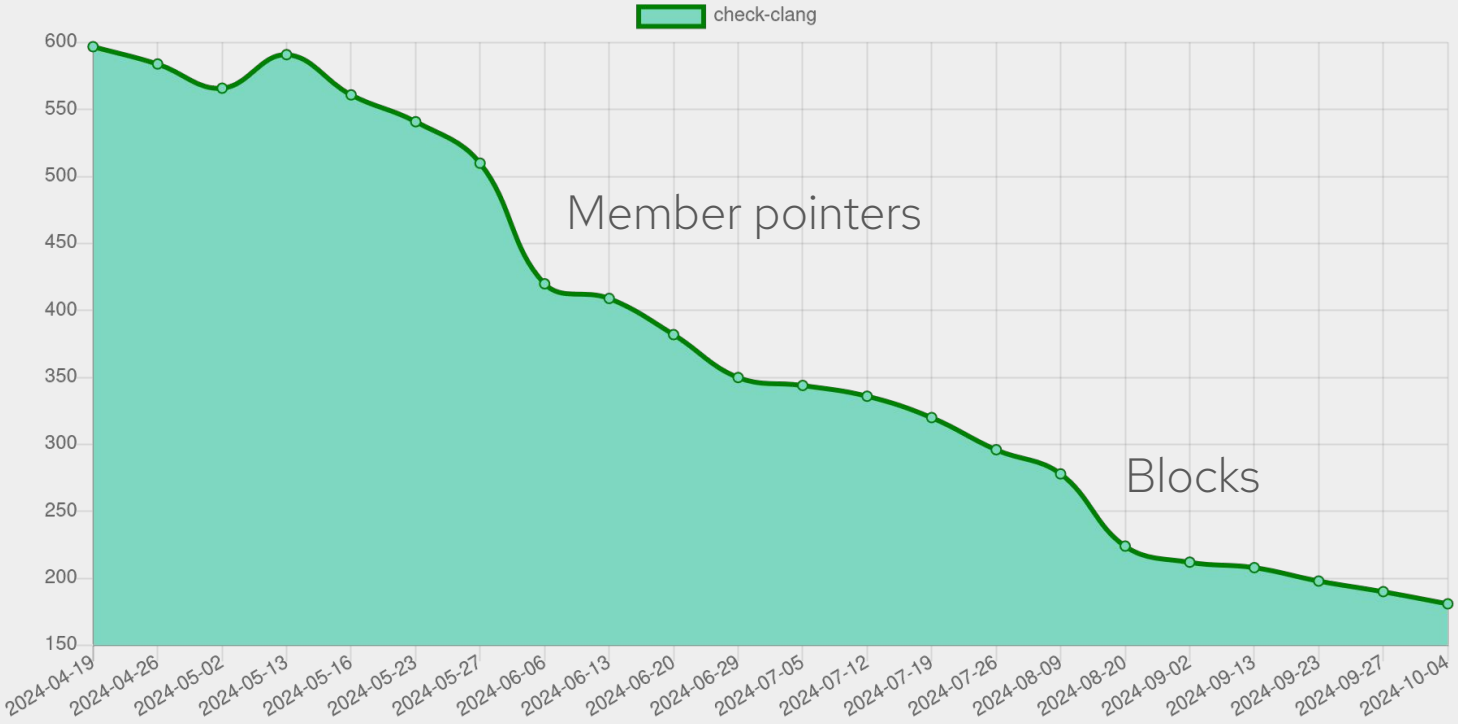


Red Hat

Testing

- `clang/test/AST/ByteCode`
- `RUN` lines in existing test files with `-fexperimental-new-constant-interpreter`
- Periodic testing of the entire clang test suite

Testing



https://tbaederr.github.io/stats/

Red Hat

Future Work

- `__builtin_constant_p`
- `typeid` pointers
- `__builtin_bit_cast`
- Array Fillers

QUESTIONS

THANKS