Building a JIT compiler for CPython

Brandt Bucher (May 19th, 2024)

Building a JIT compiler for CPython

Brandt Bucher (May 19th, 2024)

Brandt Bucher

Brandt Bucher

- 2017: Started using Python.
- 2018: Contributed code to CPython.
- 2019: Joined Python's Triage Team.
- 2020: Joined Python's Core Development Team.
- 2021: Joined Microsoft's CPython Performance Engineering Team.
- 2022: Helped make CPython 3.11 25% faster!
- 2023: Implemented CPython's new JIT compiler.

Background

Background

- CPython 3.11:
  - Specializing adaptive interpreter profiles and optimizes programs on-the-fly
- CPython 3.12:
  - Interpreter generator allows analysis and modification from a DSL spec
- CPython 3.13:
  - Second "micro-op" interpreter detects, optimizes, and executes "hot" code

Background

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

Background

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

Background

```
for _ in range(n):
    a, b = b, a + b
```

Background

Bytecode

```
for _ in range(n):      FOR_ITER
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

Background

Bytecode

```
for _ in range(n):    FOR_ITER
    a, b = b, a + b   STORE_FAST
                      LOAD_FAST_LOAD_FAST
                      LOAD_FAST
                      BINARY_OP
                      STORE_FAST_STORE_FAST
                      JUMP_BACKWARD                stack
```

Background

Bytecode

```
for _ in range(n):    FOR_ITER
    a, b = b, a + b   STORE_FAST
                      LOAD_FAST_LOAD_FAST
                      LOAD_FAST
                      BINARY_OP
                      STORE_FAST_STORE_FAST
                      JUMP_BACKWARD
```

stack

iterator

Background

Bytecode

```
for _ in range(n):      FOR_ITER
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

stack

next(iterator)

iterator

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

stack

next(iterator)

iterator

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

stack

next(iterator)

iterator

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

stack

next(iterator)

iterator

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST       stack
                        JUMP_BACKWARD             iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP                     stack
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD               iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                       stack
                        BINARY_OP                         a
                        STORE_FAST_STORE_FAST             b
                        JUMP_BACKWARD                 iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST          stack
                        LOAD_FAST                      b
                        BINARY_OP                      a
                        STORE_FAST_STORE_FAST          b
                        JUMP_BACKWARD               iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                     stack
                        BINARY_OP                     a + b
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD                iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                     stack
                        BINARY_OP                     a + b
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD                iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                     stack
                        BINARY_OP_ADD_INT             a + b
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD                iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST              b = a + b
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP_ADD_INT           stack
                        STORE_FAST_STORE_FAST         b
                        JUMP_BACKWARD             iterator
```

Background

Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST              b = a + b
                        LOAD_FAST_LOAD_FAST     a = b
                        LOAD_FAST
                        BINARY_OP_ADD_INT
                        STORE_FAST_STORE_FAST       stack
                        JUMP_BACKWARD             iterator
```

Background

Specialized Bytecode

```
for _ in range(n):    FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b   STORE_FAST              b = a + b
                      LOAD_FAST_LOAD_FAST     a = b
                      LOAD_FAST
                      BINARY_OP_ADD_INT
                      STORE_FAST_STORE_FAST       stack
                      JUMP_BACKWARD             iterator
```

Background

Specialized Bytecode

```
FOR_ITER_RANGE
STORE_FAST
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

Micro-Op Traces

```
FOR_ITER_RANGE
STORE_FAST
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

Micro-Op Traces

```
FOR_ITER_RANGE

STORE_FAST

LOAD_FAST_LOAD_FAST
```

```
LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
FOR_ITER_RANGE

STORE_FAST

LOAD_FAST_LOAD_FAST
```

```
LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
LOAD_FAST
BINARY_OP_ADD_INT
STORE_FAST
STORE_FAST_STORE_FAST
LOAD_FAST_LOAD_FAST
JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

STORE_FAST

LOAD_FAST_LOAD_FAST
```

```
LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE
```

```
LOAD_FAST
```

```
BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST
```

```
LOAD_FAST_LOAD_FAST
```

```
JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

LOAD_FAST_LOAD_FAST
```

```
LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
```

```
LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE                    BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

                                    STORE_FAST_STORE_FAST
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST

                                    JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

STORE_FAST_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

STORE_FAST_STORE_FAST



JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT



JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
_STORE_FAST

JUMP_BACKWARD
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
_STORE_FAST
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_JUMP_TO_TOP
```

Background

Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_CHECK_VALIDITY_AND_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST
_LOAD_FAST
```

```
_CHECK_VALIDITY_AND_SET_IP
_LOAD_FAST

_CHECK_VALIDITY_AND_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_CHECK_VALIDITY_AND_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY_AND_SET_IP
_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_STORE_FAST

_LOAD_FAST
_LOAD_FAST
```

```
_LOAD_FAST

_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_STORE_FAST
_STORE_FAST

_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_STORE_FAST

_LOAD_FAST
_LOAD_FAST
```

```
_LOAD_FAST

_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_STORE_FAST
_STORE_FAST

_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_STORE_FAST

_LOAD_FAST
_LOAD_FAST
```

```
_LOAD_FAST

_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_STORE_FAST
_STORE_FAST

_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_STORE_FAST

_LOAD_FAST
_LOAD_FAST
```

```
_LOAD_FAST

_GUARD_BOTH_INT
_BINARY_OP_ADD_INT

_STORE_FAST
_STORE_FAST

_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_STORE_FAST

_LOAD_FAST
_LOAD_FAST
```

```
_LOAD_FAST

_BINARY_OP_ADD_INT

_STORE_FAST
_STORE_FAST

_JUMP_TO_TOP
```

Background

Optimized Micro-Op Traces

```
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE
_STORE_FAST
_LOAD_FAST
_LOAD_FAST
_LOAD_FAST
_BINARY_OP_ADD_INT
_STORE_FAST
_STORE_FAST
_JUMP_TO_TOP
```

Just-In-Time Compilation

Just-In-Time Compilation

- Technical goals:
  - Remove interpretive overhead
  - Compile optimized traces to machine code
  - Reduce indirection:
    - "Burn in" constants, caches, and arguments
    - Move data off of frames and into registers
    - Bring hot code paths in-line

- Deployment goals:
  - Broad platform support
  - Few runtime dependencies
  - Low implementation complexity

Just-In-Time Compilation

- Technical goals:
  - Remove interpretive overhead
  - Compile optimized traces to machine code
  - Reduce indirection:
    - "Burn in" constants, caches, and arguments
    - Move data off of frames and into registers
    - Bring hot code paths in-line

- Deployment goals:
  - Broad platform support
  - Few runtime dependencies
  - Low implementation complexity

Just-In-Time Compilation

Copy-And-Patch Compilation

Copy-And-Patch Compilation

- Haoran Xu and Fredrik Kjolstad. 2021. Copy-and-Patch Compilation: A Fast Compilation Algorithm for High- Level Languages and Bytecode. Proc. ACM Program. Lang. 5, OOPSLA, Article 136 (October 2021), 30 pages. https://doi.org/10.1145/3485513

- Haoran Xu. 2023. Building a baseline JIT for Lua automatically. (12 March 2023). Retrieved from https://sillycross.github.io/2023/05/12/2023-05-12/.

- A way of automatically turning a C interpreter into a fast template JIT compiler

Copy-And-Patch Compilation

- Compared to WebAssembly baseline compiler (`Liftoff`):
  - 5x faster code generation
  - 50% faster code

- Compared to traditional JIT toolchain (`LLVM -O0`):
  - 100x faster code generation
  - 15% faster code

- Compared to an optimizing JIT with hand-written assembly (`LuaJIT`):
  - Faster on 13/44 benchmarks
  - Only 35% slower overall

Copy-And-Patch Compilation

- At runtime, walk over a sequence of bytecode instructions.
- For each:
  - Copy some static, pre-compiled machine code into executable memory
  - Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

- At runtime, walk over a sequence of bytecode instructions.
- For each:
  - Copy some static, pre-compiled machine code into executable memory
  - Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

- Copy some static, pre-compiled machine code into executable memory
- Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

- When linking or loading a relocatable object file (ELF, COFF, Mach-O, etc.):
  - Copy some static, pre-compiled machine code into executable memory
  - Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

```
case _LOAD_FAST:
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    break;
```

Copy-And-Patch Compilation

```
case _LOAD_FAST:
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    break;
```

Copy-And-Patch Compilation

```
case _LOAD_FAST:
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    break;
```

Copy-And-Patch Compilation

```
case _LOAD_FAST:
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    break;
```

Copy-And-Patch Compilation

```
case _LOAD_FAST:
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    break;
```

Copy-And-Patch Compilation

```
_LOAD_FAST

    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(void)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(void)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_RUN_NEXT_MICRO_OP(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_RUN_NEXT_MICRO_OP(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_RUN_NEXT_MICRO_OP(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
extern int MAGICALLY_INSERT_OPARG;
extern int MAGICALLY_RUN_NEXT_MICRO_OP(_PyInterpreterFrame *, PyObject **);

int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_RUN_NEXT_MICRO_OP(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
extern int MAGICALLY_INSERT_OPARG;
extern int MAGICALLY_RUN_NEXT_MICRO_OP(_PyInterpreterFrame *, PyObject **);

int
_LOAD_FAST(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *value = frame->localsplus[MAGICALLY_INSERT_OPARG];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_RUN_NEXT_MICRO_OP(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  &MAGICALLY_INSERT_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  &MAGICALLY_INSERT_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  &MAGICALLY_INSERT_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  &MAGICALLY_INSERT_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  &MAGICALLY_INSERT_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  &MAGICALLY_INSERT_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
      b8 00 00 00 00 mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x12
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

01: R_X86_64_32          MAGICALLY_INSERT_OPARG
1b: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
      b8 00 00 00 00 mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x12
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

01: R_X86_64_32          MAGICALLY_INSERT_OPARG
1b: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
      b8 00 00 00 00 mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x12
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

01: R_X86_64_32          MAGICALLY_INSERT_OPARG
1b: R_X86_64_GOTPCRELX MAGICALLY_RUN_NEXT_MICRO_OP - 0x4
```

Copy-And-Patch Compilation

```
      b8 00 00 00 00 mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x12
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi


01: R_X86_64_32         MAGICALLY_INSERT_OPARG
```

Copy-And-Patch Compilation

```
static void
emit__LOAD_FAST(unsigned char *code, _PyUOpInstruction *uop)
{
    const unsigned char code_body[] = {
          b8,   00,   00,   00,   00,   48,   8b,   44,
          c7,   48,   8b,   08,   ff,   c1,   74,   02,
          89,   08,   48,   89,   06,   48,   83,   c6,
          08,
    };
    memcpy(code, code_body, sizeof(code_body));
    memcpy(code + 1, &uop->oparg, 4);
}
```

Copy-And-Patch Compilation

```
static void
emit__LOAD_FAST(unsigned char *code, _PyUOpInstruction *uop)
{
    const unsigned char code_body[] = {
        0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
        0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
        0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
        0x08,
    };
    memcpy(code, code_body, sizeof(code_body));
    memcpy(code + 1, &uop->oparg, 4);
}
```

Copy-And-Patch Compilation

```
static void
emit__LOAD_FAST(unsigned char *code, _PyUOpInstruction *uop)
{
    const unsigned char code_body[] = {
        0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
        0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
        0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
        0x08,
    };
    memcpy(code, code_body, sizeof(code_body));
    memcpy(code + 1, &uop->oparg, 4);
}
```

Copy-And-Patch Compilation

```
static void
emit__LOAD_FAST(unsigned char *code, _PyUOpInstruction *uop)
{
    const unsigned char code_body[] = {
        0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
        0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
        0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
        0x08,
    };
    memcpy(code, code_body, sizeof(code_body));
    memcpy(code + 1, &uop->oparg, 4);
}
```

Copy-And-Patch Compilation

```
static void
emit__LOAD_FAST(unsigned char *code, _PyUOpInstruction *uop)
{
    const unsigned char code_body[] = {
        0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
        0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
        0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
        0x08,
    };
    memcpy(code, code_body, sizeof(code_body));
    memcpy(code + 1, &uop->oparg, 4);
}
```

Copy-And-Patch Compilation

```
static void
emit__LOAD_FAST(unsigned char *code, _PyUOpInstruction *uop)
{
    const unsigned char code_body[] = {
        0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
        0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
        0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
        0x08,
    };
    memcpy(code, code_body, sizeof(code_body));
    memcpy(code + 1, &uop->oparg, 4);
}
```

Copy-And-Patch Compilation

- Build time:
  - ~1000 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple hand-written C

Copy-And-Patch Compilation

- Build time:
  - ~1000 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-*ish* hand-written C

Copy-And-Patch Compilation

- Build time:
  - ~1000 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-*ish* hand-written C
  - ~9000 lines of simple generated C

Copy-And-Patch Compilation

- Build time:
  - ~1000 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-*ish* hand-written C
  - ~9000 lines of simple generated C
  - No dependencies

Copy-And-Patch Compilation

- Build time:
  - ~1000 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-*ish* hand-written C
  - ~9000 lines of simple generated C
  - No dependencies

Platform Support

Platform Support

Platform Support

x86-64

- `x86_64-apple-darwin/clang`
- `x86_64-pc-windows-msvc/msvc`
- `x86_64-unknown-linux-gnu/clang`
- `x86_64-unknown-linux-gnu/gcc`

Platform Support

x86 and x86-64

- i686-pc-windows-msvc/msvc
- x86_64-apple-darwin/clang
- x86_64-pc-windows-msvc/msvc
- x86_64-unknown-linux-gnu/clang
- x86_64-unknown-linux-gnu/gcc

Platform Support

AArch64, x86, and x86-64

- aarch64-apple-darwin/clang
- aarch64-pc-windows-msvc/msvc
- aarch64-unknown-linux-gnu/clang
- aarch64-unknown-linux-gnu/gcc
- i686-pc-windows-msvc/msvc
- x86_64-apple-darwin/clang
- x86_64-pc-windows-msvc/msvc
- x86_64-unknown-linux-gnu/clang
- x86_64-unknown-linux-gnu/gcc

Platform Support

AArch64, x86, and x86-64

- `aarch64-apple-darwin/clang`
- `aarch64-pc-windows-msvc/msvc`
- `aarch64-unknown-linux-gnu/clang`
- `aarch64-unknown-linux-gnu/gcc`
- `i686-pc-windows-msvc/msvc`
- `x86_64-apple-darwin/clang`
- `x86_64-pc-windows-msvc/msvc`
- `x86_64-unknown-linux-gnu/clang`
- `x86_64-unknown-linux-gnu/gcc`

Performance

Performance

- Micro-op interpreter:
  - ~20% slower
  - ~1% more memory

- JIT:
  - ~0% slower
  - ~10% more memory

PEP 659: Specializing Adaptive Interpreter

PEP 659: Specializing Adaptive Interpreter

PEP 744: JIT Compilation

faster-cpython/ideas

faster-cpython/ideas

faster-cpython/benchmarking-public

```
> PCbuild/build.bat
```

```
> PCbuild/build.bat --experimental-jit
```

```
> PCbuild/build.bat --experimental-jit

$ ./configure
```

```
> PCbuild/build.bat --experimental-jit

$ ./configure --enable-experimental-jit
```

Specialist

brandtbucher/specialist

Specialist

Specialist uses fine-grained location information to create visual representations of exactly *where* and *how* CPython's new specializing, adaptive interpreter optimizes your code.

```
def encode_decode(key: str, text: str) -> str:
    out = []
    for i, t in enumerate(text):
        k = key[i % len(key)]
        out.append(chr(ord(t) ^ ord(k)))
    return "".join(out)
```

Getting Started

Specialist supports CPython 3.11+ on all platforms.

To install, just run:

```
$ pip install specialist
```

Thank you!

@brandtbucher

Thank you!

@brandtbucher | brandt@python.org