TYPE FREEZING: EXPLOITING ATTRIBUTE TYPE MONOMORPHISM IN TRACING JIT COMPILERS

Lin Cheng¹, Berkin Ilbeyi¹, Carl Friedrich Bolz-Tereick², Christopher Batten¹

¹Computer Systems Laboratory
Cornell University
²Heinrich-Heine-Universität Düsseldorf
## Dynamic Languages Are Popular and Slow

<table>
<thead>
<tr>
<th>Rank</th>
<th>Language</th>
<th>Type</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Python</td>
<td>🛡️</td>
<td>100.0</td>
</tr>
<tr>
<td>2</td>
<td>Java</td>
<td>🛠️</td>
<td>96.3</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>🧹</td>
<td>94.4</td>
</tr>
<tr>
<td>4</td>
<td>C++</td>
<td>🎠</td>
<td>87.5</td>
</tr>
<tr>
<td>5</td>
<td>R</td>
<td>🤖</td>
<td>81.5</td>
</tr>
<tr>
<td>6</td>
<td>JavaScript</td>
<td>🏡</td>
<td>79.4</td>
</tr>
<tr>
<td>7</td>
<td>C#</td>
<td>🎣</td>
<td>74.5</td>
</tr>
<tr>
<td>8</td>
<td>Matlab</td>
<td>🕵️</td>
<td>70.6</td>
</tr>
<tr>
<td>9</td>
<td>Swift</td>
<td>🌐</td>
<td>69.1</td>
</tr>
<tr>
<td>10</td>
<td>Go</td>
<td>🐍</td>
<td>68.0</td>
</tr>
</tbody>
</table>

### Type Polymorphism → Type Monomorphism

- An identifier can hold different types of data

```python
def foo(pt):
    x = pt.x
    y = pt.y
    return x + y

pt = Point()
pt.x = 14
pt.y = 28
foo(pt)  # 42

pt = Point()
pt.x = 14.0
pt.y = 28.0
foo(pt)  # 42.0

pt = Point()
pt.x = "14"
pt.y = "28"
foo(pt)  # "1428"
```

An identifier can hold different types of data

At least 79% of the identifiers in real world Python applications are type monomorphic [1]

---

**TYPE POLYMORPHISM ➔ TYPE MONOMORPHISM**

- An identifier can hold different types of data
- At least 79% of the identifiers in real world Python applications are type monomorphic [1]

```python
def foo( pt):
    x = pt.x
    y = pt.y
    return x + y
```

```plaintext
>>> pt = Point()
>>> pt.x = 14
>>> pt.y = 28
>>> foo( pt )
42

>>> pt = Point()
>>> pt.x = 14.0
>>> pt.y = 28.0
>>> foo( pt )
42.0

>>> pt = Point()
>>> pt.x = "14"
>>> pt.y = "28"
>>> foo( pt )
"1428"
```

---

An identifier can hold different types of data

At least 79% of the identifiers in real world Python applications are type monomorphic [1]

**Attribute type monomorphism** is a special kind of type monomorphism, in which a certain attribute of a user-defined type only holds a single type of data

---

### Attribute Type Monomorphism → 75% of All Reads

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total</th>
<th>Monomorphic Primitive (%)</th>
<th>Monomorphic User-Defined (%)</th>
<th>Polymorphic (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deltablue</td>
<td>524.10 M</td>
<td>56.6</td>
<td>41.7</td>
<td>1.7</td>
</tr>
<tr>
<td>raytrace</td>
<td>5.01 B</td>
<td>2.7</td>
<td>9.6</td>
<td>87.4</td>
</tr>
<tr>
<td>richards</td>
<td>808.08 M</td>
<td>64.5</td>
<td>7.5</td>
<td>24.6</td>
</tr>
<tr>
<td>eparse</td>
<td>20.34 M</td>
<td>51.6</td>
<td>0.1</td>
<td>48.4</td>
</tr>
<tr>
<td>telco</td>
<td>376.50 M</td>
<td>70.9</td>
<td>1.6</td>
<td>27.5</td>
</tr>
<tr>
<td>float</td>
<td>150.01 M</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>html5lib</td>
<td>21.17 M</td>
<td>70.0</td>
<td>5.7</td>
<td>24.4</td>
</tr>
<tr>
<td>chaos</td>
<td>538.39 M</td>
<td>86.1</td>
<td>0.0</td>
<td>13.9</td>
</tr>
<tr>
<td>pickle</td>
<td>55.93 M</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>django</td>
<td>32.48 M</td>
<td>71.5</td>
<td>14.2</td>
<td>14.3</td>
</tr>
<tr>
<td>sympy</td>
<td>6.20 M</td>
<td>87.3</td>
<td>0.0</td>
<td>12.6</td>
</tr>
<tr>
<td>sympy-opt</td>
<td>6.20 M</td>
<td>86.8</td>
<td>0.0</td>
<td>13.1</td>
</tr>
<tr>
<td>gcbench</td>
<td>37.14 M</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>genshi-xml</td>
<td>5.84 M</td>
<td>98.0</td>
<td>1.7</td>
<td>0.3</td>
</tr>
<tr>
<td>chameleon</td>
<td>451.46 K</td>
<td>84.0</td>
<td>0.4</td>
<td>15.6</td>
</tr>
<tr>
<td>mako</td>
<td>260.12 K</td>
<td>73.7</td>
<td>17.5</td>
<td>8.8</td>
</tr>
<tr>
<td>meteor-contest</td>
<td>11.52 K</td>
<td>77.9</td>
<td>0.6</td>
<td>21.5</td>
</tr>
<tr>
<td>nbbody-modified</td>
<td>7.88 K</td>
<td>68.3</td>
<td>0.8</td>
<td>30.9</td>
</tr>
<tr>
<td>fib</td>
<td>7.88 K</td>
<td>68.3</td>
<td>0.8</td>
<td>30.9</td>
</tr>
</tbody>
</table>
# Attribute Type Monomorphism $\rightarrow$ 75% of All Reads

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total</th>
<th>Monomorphic</th>
<th>Polymorphic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Primitive (%)</td>
<td>User-Defined (%)</td>
</tr>
<tr>
<td>delta blue</td>
<td>524.10 M</td>
<td>56.6</td>
<td>41.7</td>
</tr>
<tr>
<td>raytrace</td>
<td>5.01 B</td>
<td>2.7</td>
<td>9.6</td>
</tr>
<tr>
<td>richards</td>
<td>808.08 M</td>
<td>64.5</td>
<td>7.5</td>
</tr>
<tr>
<td>eparse</td>
<td>20.34 M</td>
<td>51.6</td>
<td>0.1</td>
</tr>
<tr>
<td>telco</td>
<td>376.50 M</td>
<td>70.9</td>
<td>1.6</td>
</tr>
<tr>
<td>float</td>
<td>150.01 M</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>html5lib</td>
<td>21.17 M</td>
<td>70.0</td>
<td>5.7</td>
</tr>
<tr>
<td>chaos</td>
<td>538.39 M</td>
<td>86.1</td>
<td>0.0</td>
</tr>
<tr>
<td>pickle</td>
<td>55.93 M</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>django</td>
<td>32.48 M</td>
<td>71.5</td>
<td>14.2</td>
</tr>
<tr>
<td>sympy</td>
<td>6.20 M</td>
<td>87.3</td>
<td>0.0</td>
</tr>
<tr>
<td>sympy-opt</td>
<td>6.20 M</td>
<td>86.8</td>
<td>0.0</td>
</tr>
<tr>
<td>gcbench</td>
<td>37.14 M</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>genshi-xml</td>
<td>5.84 M</td>
<td>98.0</td>
<td>1.7</td>
</tr>
<tr>
<td>chameleon</td>
<td>451.46 K</td>
<td>84.0</td>
<td>0.4</td>
</tr>
<tr>
<td>mako</td>
<td>260.12 K</td>
<td>73.7</td>
<td>17.5</td>
</tr>
<tr>
<td>meteor-contest</td>
<td>11.52 K</td>
<td>77.9</td>
<td>0.6</td>
</tr>
<tr>
<td>nbody-modified</td>
<td>7.88 K</td>
<td>68.3</td>
<td>0.8</td>
</tr>
<tr>
<td>fib</td>
<td>7.88 K</td>
<td>68.3</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Simple Type Freezing**

**Nested Type Freezing**
Type Freezing

Motivation
Background
Simple Type Freezing
Nested Type Freezing
Evaluation
class Point( object ):  
    def __init__( self, x, y ):  
        self.x = x  
        self.y = y  

pt1 = Point( 42, 1 )  
pt2 = Point( 6, 7 )

- Attributes can be added to, or removed from, an instance dynamically: It is necessary to keep track of each instance’s attribute list

- CPython associates a complex and memory hungry Dict with each instance
class Point( object ):
    def __init__( self, x, y ):
        self.x = x
        self.y = y

pt1    = Point( 42, 1 )
...  
pt200k = Point( 6, 7 )

Attributes can be added to, or removed from, an instance dynamically: It is necessary to keep track of each instance’s attribute list

CPython associates a complex and memory hungry Dict with each instance
USER DEFINED TYPES - THE PyPy WAY

Instances are likely to have the same set of attributes: modern JIT compilers usually implement an optimization called Maps (also known as Hidden Classes or Shapes)

User-defined types are structural: an instance's map determines its type

```python
class Point( object ):
    def __init__( self, x, y ):
        self.x = x
        self.y = y

pt1 = Point( 42, 1 )
pt2 = Point( 6, 7 )
```
```python
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Line(object):
    def __init__(self, pt1, pt2):
        self.pt1 = pt1
        self.pt2 = pt2

def create_lines(n):
    lines = []
    for i in range(n):
        pt1 = Point(i, n-i)
        pt2 = Point(2*i-n, i-n)
        lines.append(Line(pt1, pt2))
    return lines

def total_length(n, lines):
    length = 0
    i = 0
    while (i < n):
        line = lines[i]
        pt1 = line.pt1
        pt2 = line.pt2
        a_side = (pt1.x - pt2.x) ** 2
        b_side = (pt1.y - pt2.y) ** 2
        length += math.sqrt(a_side + b_side)
    return length
```

def total_length( n, lines ):
    length = 0
    i = 0
    while( i < n ):
        line = lines[i]
        pt1 = line.pt1
        pt2 = line.pt2
        a_side = (pt1.x - pt2.x)**2
        b_side = (pt1.y - pt2.y)**2
        length += math.sqrt( a_side + b_side )
    return length

p7 = get_array_item( p0, i1 )    # line = lines[i]
guard_class( p7, W_ObjectObject ) #
```python
def total_length( n, lines ):
    length = 0
    i = 0
    while( i < n ):
        line = lines[i]
        pt1 = line.pt1
        pt2 = line.pt2
        a_side = ( pt1.x - pt2.x ) ** 2
        b_side = ( pt1.y - pt2.y ) ** 2
        length += math.sqrt( a_side + b_side )
    return length
```

```python
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #
p8 = get( p7, Map )               # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
guard_class( p9, W_ObjectObject ) #
```
def total_length( n, lines ):
    length = 0
    i = 0
    while( i < n ):
        line = lines[i]
        pt1 = line.pt1
        pt2 = line.pt2
        a_side = ( pt1.x - pt2.x ) ** 2
        b_side = ( pt1.y - pt2.y ) ** 2
        length += math.sqrt( a_side + b_side )
    return length

p7 = get_array_item( p0, i1 )    # line = lines[i]
guard_class( p7, W_ObjectObject ) #
p8 = get( p7, Map )              # pt1 = line.pt1
guard_value( p8, Map of Line )  #
guard_not_invalidated()         #
p9 = get( p7, slot0 )            #
guard_class( p9, W_ObjectObject ) #
p10 = get( p7, slot1 )           # pt2 = line.pt2
guard_class( p7, W_ObjectObject ) #
def total_length( n, lines ):
    length = 0
    i = 0
    while( i < n ):
        line = lines[i]
        pt1 = line.pt1
        pt2 = line.pt2
        a_side = (pt1.x - pt2.x)**2
        b_side = (pt1.y - pt2.y)**2
        length += math.sqrt(a_side + b_side)
    return length
```python
def total_length( n, lines ):
    length = 0
    i = 0
    while( i < n ):
        line = lines[i]
        pt1 = line.pt1
        pt2 = line.pt2
        a_side = ( pt1.x - pt2.x ) ** 2
        b_side = ( pt1.y - pt2.y ) ** 2
        length += math.sqrt( a_side + b_side )
    return length
```
Type Freezing

Motivation

Background

Simple Type Freezing

Nested Type Freezing

Evaluation
To exploit attribute type monomorphism, we need to keep track of which attributes are type monomorphic.
**TECHNIQUE: SIMPLE TYPE FREEZING**

```python
class Point( object ):
    def __init__( self, x, y ):
        self.x = x
        self.y = y

pt1 = Point( 42, 1 )
pt2 = Point( 6, 7 )
```

- To exploit attribute type monomorphism, we need to keep track of which attributes are type monomorphic.
- We "freeze" the type information of attributes into the map with an auxiliary field, *known type*.
- With this extra info, unmodified PyPy JIT compiler is able to prove the type of these attributes at compiling time and eliminate type guards on them.
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = (pt1.x - pt2.x)**2
    b_side = (pt1.y - pt2.y)**2
    length += math.sqrt(a_side + b_side)
**CHALLENGE: ATTRIBUTES MAY BECOME POLYMORPHIC**

```python
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )

pt = Point( "42", 1 )
```

```
p7 = get_array_item( p0, i1 )  # line = lines[i]
guard_class( p7, W_ObjectObject ) #
p8 = get( p7, Map )  # pt1 = line.pt1
guard_value( p8, Map of Line )  #
guard_not_invalidated()  #
p9 = get( p7, slot0 )  #

p10 = get( p7, slot1 )  # pt2 = line.pt2
p11 = get( p9, Map )  # pt1.x
guard_value( p11, Map of Point)  #
p12 = get( p9, slot0 )  #

p13 = get( p10, Map )  # pt2.x
guard_value( p13, Map of Point)  #
p14 = get( p10, slot0 )  #

...

p19 = get( p9, slot1 )  # pt1.y
p20 = get( p10, slot1 )  # pt2.y
```

---

**Diagram:**

- **pt1** to **Point instance**
  - **"y"**
    - 1
    - INT
  - **42**
  - 1
- **pt2** to **Storage**
  - **"x"**
    - 0
    - INT
  - **Known Type**
  - **Next Entry**
**Challenge: Attributes May Become Polymorphic**

```python
pt = Point( "42", 1 )
p7 = get_array_item( p0, i1 )  # line = lines[i]
guard_class( p7, W_ObjectObject )  #
p8 = get( p7, Map )  # pt1 = line.pt1
guard_value( p8, Map of Line )  #
guard_not_invalidated()  #
p9 = get( p7, slot0 )  #

pt1 = line.pt1
p10 = get( p7, slot1 )  # pt2 = line.pt2
p11 = get( p9, Map )  # pt1.x
guard_value( p11, Map of Point )  #
p12 = get( p9, slot0 )  #

pt2 = line.pt2
p13 = get( p10, Map )  # pt2.x
guard_value( p13, Map of Point )  #
p14 = get( p10, slot0 )  #

... p19 = get( p9, slot1 )  # pt1.y
p20 = get( p10, slot1 )  # pt2.y
...```

Diagram:
- **pt** = Point("42", 1)
- **pt1** = Line.pt1
- **pt2** = Line.pt2
- **Storage**
  - "42" 1
  - 42 1
  - "y" 1
  - "x" 0
  - "x" INT
  - "y" INT
  - Known Type
  - Next Entry

- **Guard Class**
  - `guard_class(p7, W_ObjectObject)`
- **Guard Value**
  - `guard_value(p8, Map of Line)`
  - `guard_value(p11, Map of Point)`
- **Guard Not Invalidated**
  - `guard_not_invalidated()`
pt = Point("42", 1)

pt

Point instance

Storage

"42"
1

"y"
1

Attribute Name

Storage Slot

"x"
0

Mutated

Known Type

Next Entry

pt7 = get_array_item( p0, i1 ) # line = lines[i]
guard_class( pt7, W_ObjectObject ) #
p8 = get( pt7, Map ) # pt1 = line.pt1
guard_value( p8, Map of Line ) #
guard_not_invalidated() #
p9 = get( pt7, slot0 ) #
p10 = get( pt7, slot1 ) #
p11 = get( p9, Map ) # pt1.x
guard_value( p11, Map of Line ) #
p12 = get( p9, slot0 ) #
p13 = get( p10, Map ) # pt2.x
guard_value( p13, Map of Line ) #
p14 = get( p10, slot0 ) #

... p19 = get( p9, slot1 ) # pt1.y
p20 = get( p10, slot1 ) # pt2.y

...
Type Freezing

Motivation
Background
Simple Type Freezing
Nested Type Freezing
Evaluation
class Line( object ):
    def __init__( self, pt1, pt2 ):
        self.pt1 = pt1
        self.pt2 = pt2

line = Line( Point(42, 1), Point(6, 7) )

- If a type monomorphic attribute stores another kind of user-defined objects, knowing the type of top-level object implicitly tells the layout of nested objects.
- Like the case in simple type freezing, we associate another quasi-immutable auxiliary field, known map, with each map entry.
while (i < n):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = (pt1.x - pt2.x) ** 2
    b_side = (pt1.y - pt2.y) ** 2
    length += math.sqrt(a_side + b_side)

    p7 = get_array_item(p0, i1)  # line = lines[i]
    guard_class(p7, W_ObjectObject)  #

    p8 = get(p7, Map)  # pt1 = line.pt1
    guard_value(p8, Map of Line)  #
    guard_not_invalidated()  #
    p9 = get(p7, slot0)  #

    p10 = get(p7, slot1)  # pt2 = line.pt2
    p11 = get(p9, Map)  # pt1.x
    guard_value(p11, Map of Point)  #
    p12 = get(p9, slot0)  #

    p13 = get(p10, Map)  # pt2.x
    guard_value(p13, Map of Point)  #
    p14 = get(p10, slot0)  #

    . . .
    p19 = get(p9, slot1)  # pt1.y

    p20 = get(p10, slot1)  # pt2.y
    . . .
while (i < n):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = (pt1.x - pt2.x)**2
    b_side = (pt1.y - pt2.y)**2
    length += math.sqrt(a_side + b_side)

    p7 = get_array_item(p0, i1) # line = lines[i]
    guard_class(p7, W_ObjectObject) #
    p8 = get(p7, Map) # pt1 = line.pt1
    guard_value(p8, Map of Line) #
    guard_not_invalidated() #
    p9 = get(p7, slot0) #

    p10 = get(p7, slot1) # pt2 = line.pt2
    p11 = get(p9, Map) # pt1.x
    guard_value(p11, Map of Line) #
    p12 = get(p9, slot0) #
    p13 = get(p10, Map) # pt2.x
    guard_value(p13, Map of Line) #
    p14 = get(p10, slot0) #

    p19 = get(p9, slot1) # pt1.y
    p20 = get(p10, slot1) # pt2.y

    . . .
**Challenge: Two References to a Single Object**

- \( pt = \text{line}[42].\text{pt2} \)
- \( \text{delattr}(\text{pt2}, "x") \)

```
pt7 = get_array_item( p0, i1 )  # line = lines[i]
guard_class( p7, W_ObjectObject )  #
p8 = get( p7, Map )             # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
p10 = get( p7, slot1 )            # pt2 = line.pt2

p12 = get( p9, slot0 )            #
p14 = get( p10, slot0 )           #
...                               #
p19 = get( p9, slot1 )            # pt1.y
p20 = get( p10, slot1 )           # pt2.y
...                               #
```
A map is said to be terminal, if and only if none of the instances that point to this map have added or removed any attributes.
A map is said to be **terminal**, if and only if none of the instances that point to this map have added or removed any attributes.
We move runtime type checking from read time to write time for two reasons

- There are generally more reads than writes
- Certain write time type checking can be optimized away, since type of the value to be written is known at JIT compile time

We used the running example as a micro-benchmark

- Type Freezing saves up to 30% of dynamic instructions
- Type Freezing improves performance by up to 6%

```python
def create_lines(n):
    lines = []
    for i in range(n):
        pt1 = Point(i, n-i)
        pt2 = Point(2*i-n, i-n)
        lines.append(Line(pt1, pt2))
    return lines
```
Type Freezing

Motivation
Background
Simple Type Freezing
Nested Type Freezing
Evaluation
### Evaluation Environment Setup

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Xeon E5620</td>
</tr>
<tr>
<td>Base Frequency</td>
<td>2.40GHz</td>
</tr>
<tr>
<td>Turbo Frequency</td>
<td>2.66GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>48GB</td>
</tr>
<tr>
<td>GC Nursery</td>
<td>6MB</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 7</td>
</tr>
<tr>
<td>Kernel Version</td>
<td>3.10.0-957.21.2.el7.x86_64</td>
</tr>
<tr>
<td>Baseline PyPy</td>
<td>PyPy 7.2.0</td>
</tr>
</tbody>
</table>

**Simple Type Freezing**
**Evaluation Environment Setup**

- **Processor**: Xeon E5620
- **Base Frequency**: 2.40GHz
- **Turbo Frequency**: 2.66GHz
- **Memory**: 48GB
- **GC Nursery**: 6MB
- **OS**: CentOS 7
- **Kernel Version**: 3.10.0-957.21.2.el7.x86_64
- **Baseline PyPy**: PyPy 7.2.0

- **Simple Type Freezing**
- **Simple and Nested Type Freezing**
**MAKE APPS TYPE FREEZING FRIENDLY**

- sympy is the only case where type freezing constantly performs worse than baseline
  - We took a deep dive and found the performance cliff is related to class Rational
  - Create attribute type polymorphism on purpose

- Many type mutation is due to straightforward casting [1]. This is also the case in Raytrace
  - Classes like Point usually hold Floats, but they are initialized to Integers
  - Convert init values to floats so they don't introduce attribute type polymorphism

```python
class Rational(Number):
    def __new__(cls, p, q):
        ...
        obj.p = p
        obj.q = q  # Small Int → Long Int
        ...
    dummy = Rational(None, None)

class Point(object):
    def __init__(cls, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        ...
s.addLight( Point(30, 30, 10) )
```

MAKE APPS TYPE FREEZING FRIENDLY

![Graph showing normalized performance and normalized number of dynamic instructions for various benchmarks.

- Simple Type Freezing
- Simple and Nested Type Freezing

Benchmark categories include deltahoe, raytrace, raytrace-opt, richards, epubse, telco, float, html5lib, chaos, pickle, gbench, meteor-contest, fib, sympy, simple-opt.](image-url)
TAKE-AWAY POINTS

- Attribute type monomorphism exists in real world applications
- We propose simple and nested type freezing to exploit attribute type monomorphism
- As a pure software technique, type freezing achieves most of the performance benefit of a prior SW/HW co-design work[1]