Twopy : un JIT pour python

Julien Pagès

28 Février 2018

Université de Montréal

Sommaire

Doctorat à l'université de Montpellier

Une machine virtuelle en héritage multiple basée sur le hachage parfait avec Roland Ducournau

Contributions

- Concentration sur les mécanismes objets
- Réalisation d'une machine virtuelle pour un langage en héritage multiple et en typage statique
- Analyse de types dynamiques pour augmenter la proportion d'appels statiques

On obtient une grande proportion d'appels statiques

ATER à l'équipe INRIA Camus à Strasbourg

De plus en plus de programmes gourmands en calcul en JavaScript, sans parallélisme dans le code généré

Modèle polyédrique et machine virtuelle JavaScript

- Dynamisme de JavaScript
- Optimisations des outils polyédriques utilisés classiquement en C
- Adaptation de JavaScriptCore (Apple) pour le parallélisme

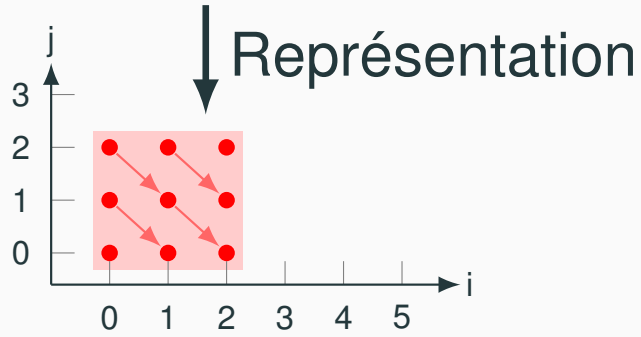
Modèle Polyédrique

```
for (int i = 0; i < 3; i++)
  for (int j = 0; j < 3; j++)
    z[i+j] += x[i] * y[j];
```

Modèle Polyédrique

```
for (int i = 0; i < 3; i++)
  for (int j = 0; j < 3; j++)
    z[i+j] += x[i] * y[j];
```

i	j	instances
0	0	z[0] += x[0] * y[0];
0	1	z[1] += x[0] * y[1];
0	2	z[2] += x[0] * y[2];
1	0	z[1] += x[1] * y[0];
1	1	z[2] += x[1] * y[1];
1	2	z[3] += x[1] * y[2];
2	0	z[2] += x[2] * y[0];
2	1	z[3] += x[2] * y[1];
2	2	z[4] += x[2] * y[2];

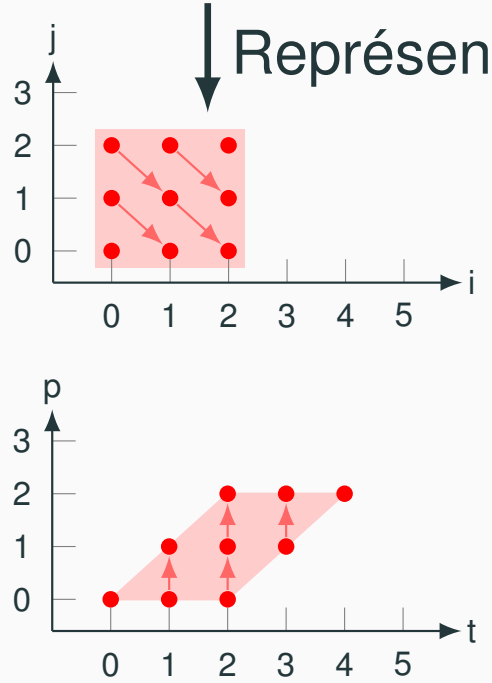


Modèle Polyédrique

```
for (int i = 0; i < 3; i++)
  for (int j = 0; j < 3; j++)
    z[i+j] += x[i] * y[j];
```

i	j	instances
0	0	z[0] += x[0] * y[0];
0	1	z[1] += x[0] * y[1];
0	2	z[2] += x[0] * y[2];
1	0	z[1] += x[1] * y[0];
1	1	z[2] += x[1] * y[1];
1	2	z[3] += x[1] * y[2];
2	0	z[2] += x[2] * y[0];
2	1	z[3] += x[2] * y[1];
2	2	z[4] += x[2] * y[2];

Représentation

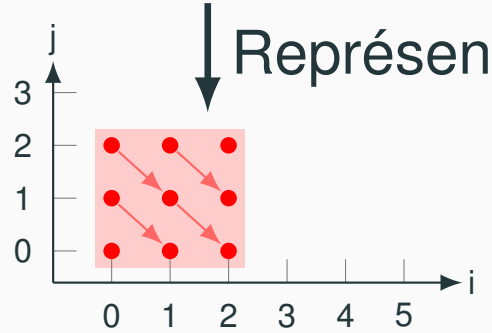


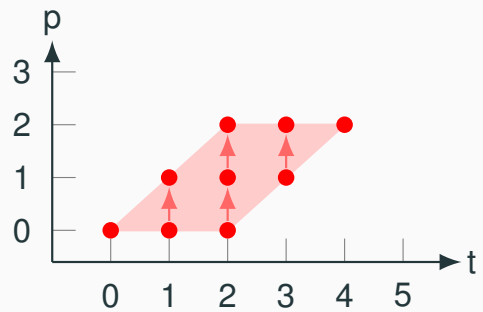
Modèle Polyédrique

```
for (int i = 0; i < 3; i++)
  for (int j = 0; j < 3; j++)
    z[i+j] += x[i] * y[j];
```

i	j	instances
0	0	z[0] += x[0] * y[0];
0	1	z[1] += x[0] * y[1];
0	2	z[2] += x[0] * y[2];
1	0	z[1] += x[1] * y[0];
1	1	z[2] += x[1] * y[1];
1	2	z[3] += x[1] * y[2];
2	0	z[2] += x[2] * y[0];
2	1	z[3] += x[2] * y[1];
2	2	z[4] += x[2] * y[2];

Représentation






```
#pragma omp parallel for
for (int t = 0; p < 5; p++)
  for (int p = max(0,t-2); p <= min(2,t); p++)
    z[t] += x[p] * y[t-p];
```

Modèle polyédrique

Transformations = Optimisations

- Localité des données
- Parallélisation

Static Control Parts (SCoPs)

- Bornes de boucles affines
- Branchements conditionnels affines
- Accès mémoire affines
- Alias connus

Objectifs du projet

Une machine virtuelle paresseuse pour python

- Python est un langage dynamique populaire
- Et assez lent

Objectif : développer un compilateur pour python basé sur le *Basic-Block Versioning* en exploitant le parallélisme

Contexte

La concurrence

- cpython : Interpréteur officiel, mauvaises performances
- pypy : principal concurrent, compilation JIT
- pyston : JIT développé par DropBox, projet abandonné en 2017
- Jython/IronPython : basés sur JVM et .NET CLR

Choix techniques

1. Écrit en python : apprentissage du langage, éventuel bootstrap
2. D'abord un interpréteur python pour étudier le langage
3. Travailler à partir du *bytecode* python, plus bas-niveau
4. Un JIT paresseux qui produit du code assembleur x86

BBV : Basic Block Versioning

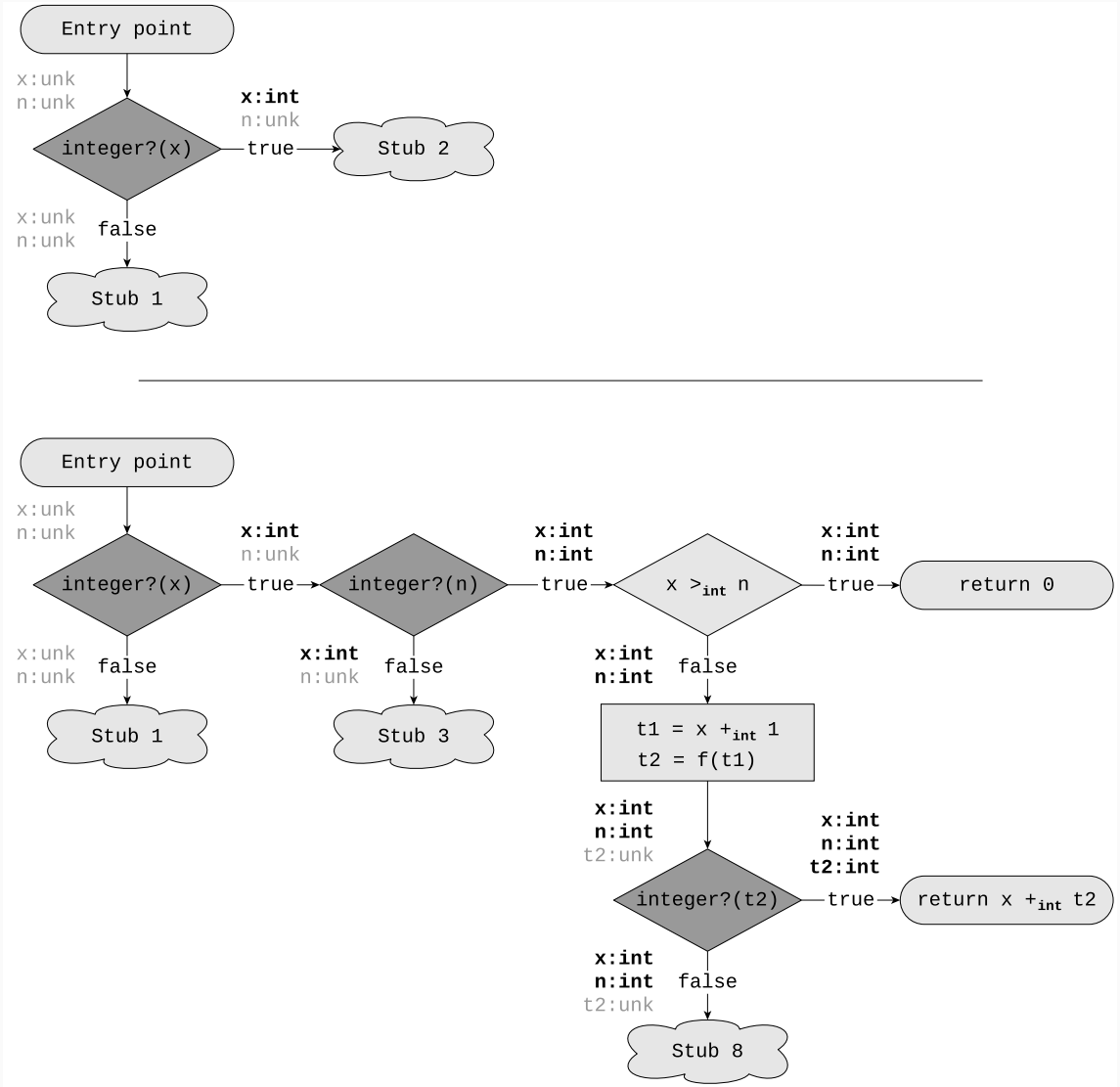


Schéma de fonctionnemment BBV depuis [SF17]

Twopy

Code source python :

```
def fib(n):
    if n<2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Twopy

Compilation en bytecode python :

```
 0 LOAD_FAST                0 (n)
 2 LOAD_CONST               1 (2)
 4 COMPARE_OP               0 (<)
 6 POP_JUMP_IF_FALSE       12
 8 LOAD_CONST               2 (1)
10 RETURN_VALUE
12 LOAD_GLOBAL              0 (fib)
14 LOAD_FAST                0 (n)
16 LOAD_CONST               2 (1)
18 BINARY_SUBTRACT
20 CALL_FUNCTION            1
22 LOAD_GLOBAL              0 (fib)
24 LOAD_FAST                0 (n)
26 LOAD_CONST               1 (2)
28 BINARY_SUBTRACT
30 CALL_FUNCTION            1
32 BINARY_ADD
34 RETURN_VALUE
36 LOAD_CONST               0 (None)
38 RETURN_VALUE
```

Twopy

Génération des blocs de base :

```
Block 1:
    0 LOAD_FAST                0 (n)
    2 LOAD_CONST               1 (2)
    4 COMPARE_OP               0 (<)
    6 POP_JUMP_IF_FALSE       12

Block 2:
    8 LOAD_CONST               2 (1)
    10 RETURN_VALUE

Block 3:
    12 LOAD_GLOBAL             0 (fib)
    14 LOAD_FAST               0 (n)
    16 LOAD_CONST              2 (1)
    18 BINARY_SUBTRACT
    20 CALL_FUNCTION           1
    22 LOAD_GLOBAL             0 (fib)
    24 LOAD_FAST               0 (n)
    26 LOAD_CONST              1 (2)
    28 BINARY_SUBTRACT
    30 CALL_FUNCTION           1
    32 BINARY_ADD
    34 RETURN_VALUE
    36 LOAD_CONST              0 (None)
    38 RETURN_VALUE
```

Twopy

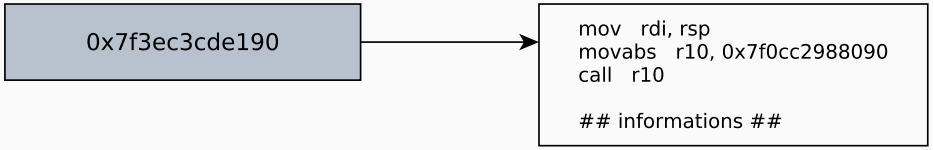
```
Function fib:
    Block 1:
    0x7f3ec3cde000: mov        r9, qword ptr [rsp + 8]
    0x7f3ec3cde005: push r9
    0x7f3ec3cde007: push 8
    0x7f3ec3cde009: pop        r8
    0x7f3ec3cde00b: pop        r9
    0x7f3ec3cde00d: cmp        r9, r8
    0x7f3ec3cde010: jl 0x7f3ec3cde190
    0x7f3ec3cde016: movabs r10, 0x7f3ec3cde1cf
    0x7f3ec3cde020: jmp        r10
```

Twopy

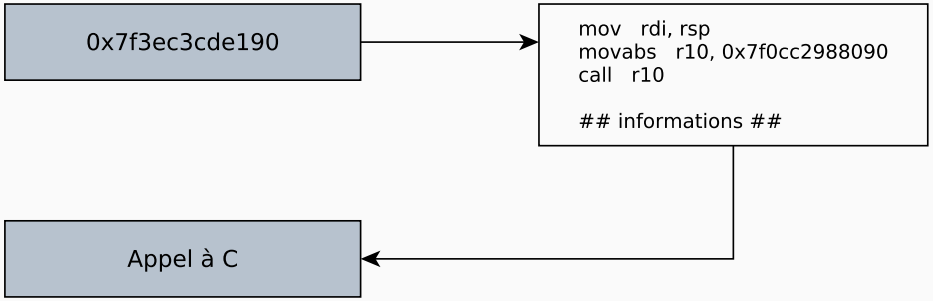
```
Function fib:
    Block 1:
    0x7f3ec3cde000: mov        r9, qword ptr [rsp + 8]
    0x7f3ec3cde005: push r9
    0x7f3ec3cde007: push 8
    0x7f3ec3cde009: pop        r8
    0x7f3ec3cde00b: pop        r9
    0x7f3ec3cde00d: cmp        r9, r8
    0x7f3ec3cde010: jl 0x7f3ec3cde190
    0x7f3ec3cde016: movabs r10, 0x7f3ec3cde1cf
    0x7f3ec3cde020: jmp        r10
```

Adresses vers des *stubs*

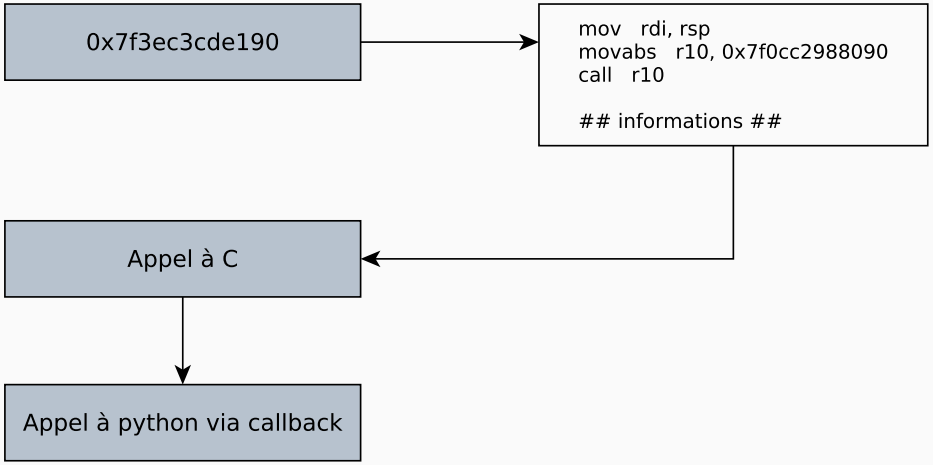
Twopy


0x7f3ec3cde190
mov rdi, rsp
movabs r10, 0x7f0cc2988090
call r10
informations

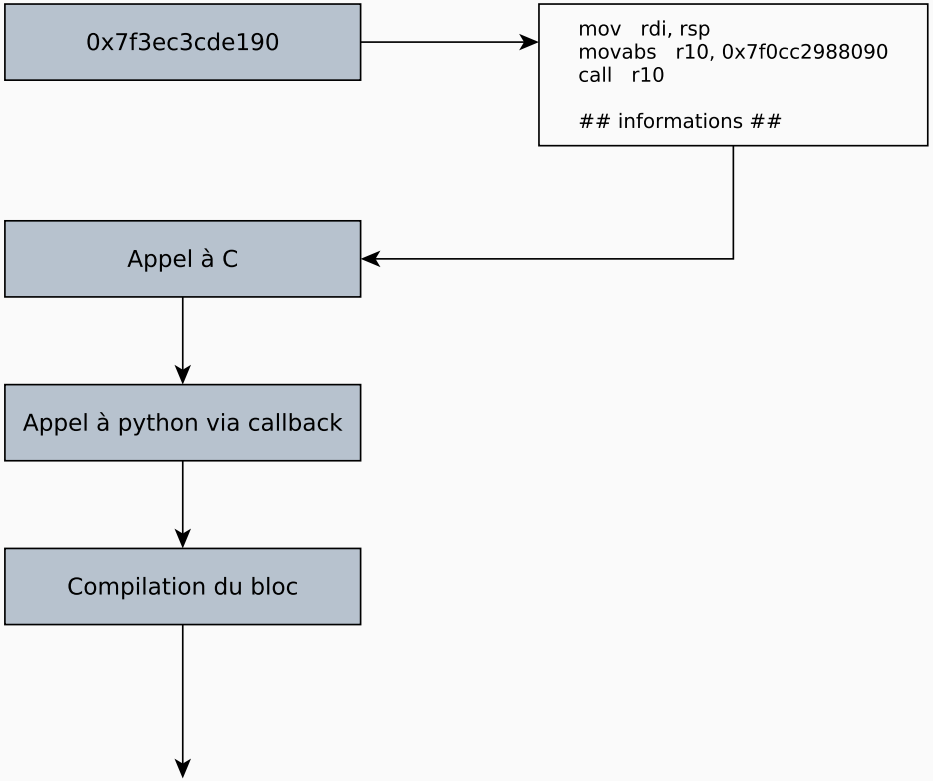
Twopy


0x7f3ec3cde190
mov rdi, rsp
movabs r10, 0x7f0cc2988090
call r10
informations
Appel à C

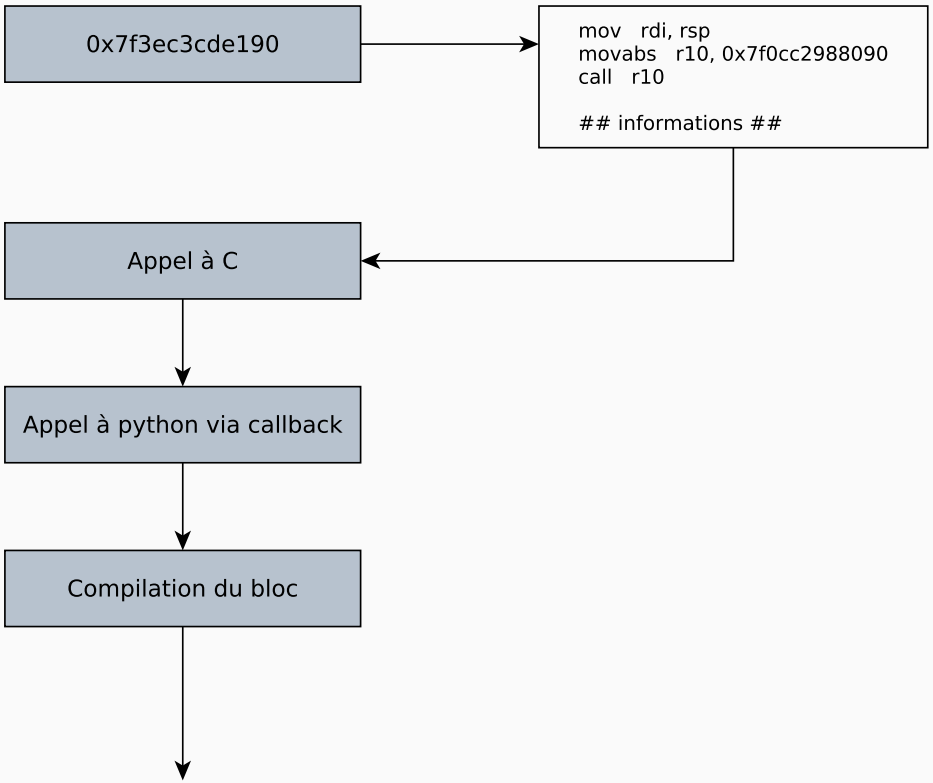
Twopy


0x7f3ec3cde190
mov rdi, rsp
movabs r10, 0x7f0cc2988090
call r10
informations
Appel à C
Appel à python via callback

Twopy

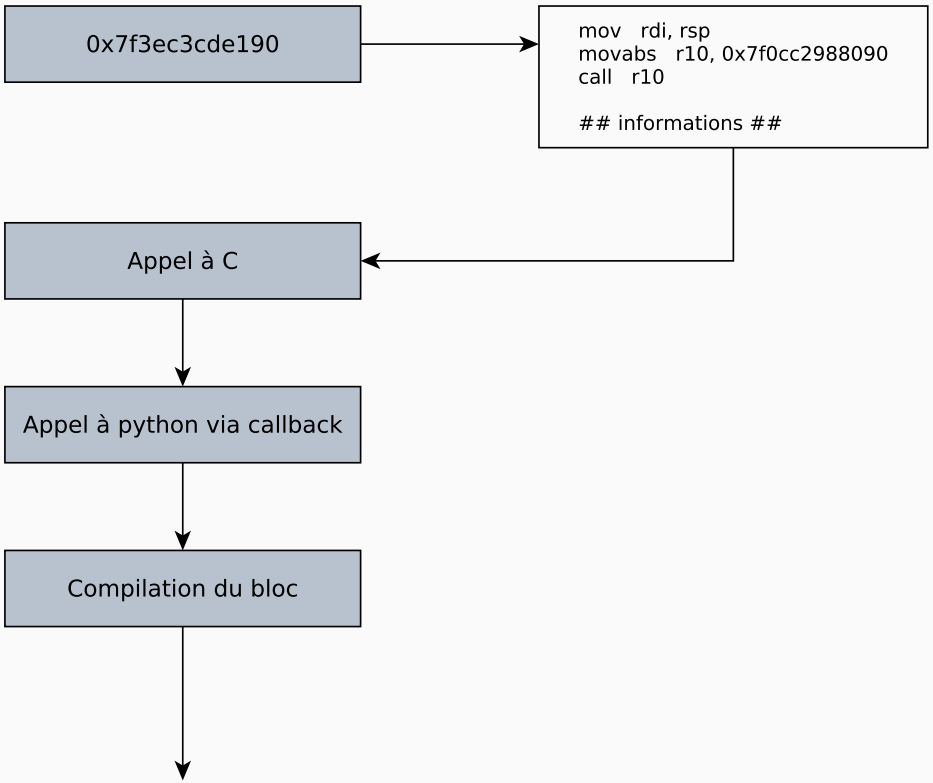

0x7f3ec3cde190
mov rdi, rsp
movabs r10, 0x7f0cc2988090
call r10
informations
Appel à C
Appel à python via callback
Compilation du bloc

Twopy




```
Block 2:
0x7f0cc8bac082:        push 4
0x7f0cc8bac084:        pop        rax
0x7f0cc8bac085:        pop        rbx
0x7f0cc8bac086:        pop        r10
0x7f0cc8bac088:        pop        r10
0x7f0cc8bac08a:        jmp        rbx
```

Twopy




```
Block 2:
0x7f0cc8bac082:        push 4
0x7f0cc8bac084:        pop        rax
0x7f0cc8bac085:        pop        rbx
0x7f0cc8bac086:        pop        r10
0x7f0cc8bac088:        pop        r10
0x7f0cc8bac08a:        jmp        rbx
```

Patch de pile (adresse de retour) et du saut vers le stub

Avancement du projet

- Mini-interpréteur du langage
- JIT paresseux à l'échelle du *basic block*
- Début de librairie standard

Premiers résultats

```python
def fib(n):
    if n<2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print(fib(40))
```

cpython	28,6s
pypy	2,5s
gcc -O1	0,5s
twopy	1,64s

Premiers résultats

```python
def fib(n):
    if n<2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print(fib(40))
```

cpython	28,6s
pypy	2,5s
gcc -O1	0,5s
twopy	1,64s

- Pas de test de types
- Pas d'allocation de registre
- Pas d'inlining

Premiers résultats

```
def fib(n):
    if n<2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print(fib(40))
```

cpython	28,6s
pypy	2,5s
gcc -O1	0,5s
twopy	1,64s

- Pas de test de types
- Pas d'allocation de registre
- Pas d'inlining
- Correspond à test de types + BBV

La suite

À court terme

- Implémenter les tests de types
- Versionnement du code avec le BBV en intra-procédural
- Gestion des flottants

À long terme

- Gestion mémoire
- BBV inter-procédural
- Allocation de registres
- Parallélisme : micro-threads pour appels de fonctions ?

Questions

Merci pour votre attention

Bibliographie

Baptiste Saleil and Marc Feeley.
Interprocedural specialization of higher-order dynamic languages without static analysis.
In 31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain, pages 23:1–23:23, 2017.