Python インタプリタの多倍長整数演算の改良

IISEC 堀川 清司, 小崎 俊二, 松尾 和人

2009 年 3 月 7 日

研究背景

Python：高水準プログラミング言語、オープンソースのインタプリタ

Python の特徴：

- 簡潔な文法 → プログラミングが容易
- インタプリタは C 言語で開発 → 様々なプラットフォームに対応

  ⇒ 幅広く利用

- 標準で多倍長整数演算に対応

  → 処理速度が不十分

Python インタプリタの多倍長整数演算を高速化

⇒ 数論アルゴリズム計算・セキュリティ技術実装の高速化

Python における多倍長整数

多倍長整数に関するプログラムファイル：

- `Include/longintrepr.h`
  - **多倍長整数の構造体定義**
- `Include/longobject.h`
  - **多倍長整数演算の C/API 関数宣言**
- `Objects/longobject.c`
  - **多倍長整数演算を行う関数記述**

⇒ **多倍長整数：`PyLongObject` 型 (構造体) として表現**

Python の多倍長整数表現

$B = 2^{\mathrm{SHIFT}}$ を基数とした多倍長整数の表現：

$$x = (-1)^s \sum_{0 \le i < n} x_i B^i,\ 0 \le x_i \le B-1,\ s \in \{0, 1\}$$

`PyLongObject`：

- `ob_size` (`signed int` 型)：$(-1)^s \cdot n$
- `ob_digit` (`unsigned short` 型)：$x_i,\ (0 \le {}^{\forall} i < n)$
  16 ビットを仮定

Python → `SHIFT`=15

- 32bit/word を仮定 (gcc 準拠)
- 冪乗算の k-ary 法における窓幅: 5bit

Python 多倍長整数演算の改良方針

- 多倍長表現桁サイズ変更：15bit → 16bit
  - 桁数減少による高速化
  - 15bit へのマスク処理省略による簡略化

  ⇒ 乗算・除算・冪乗算の高速化

プログラムファイル変更部分

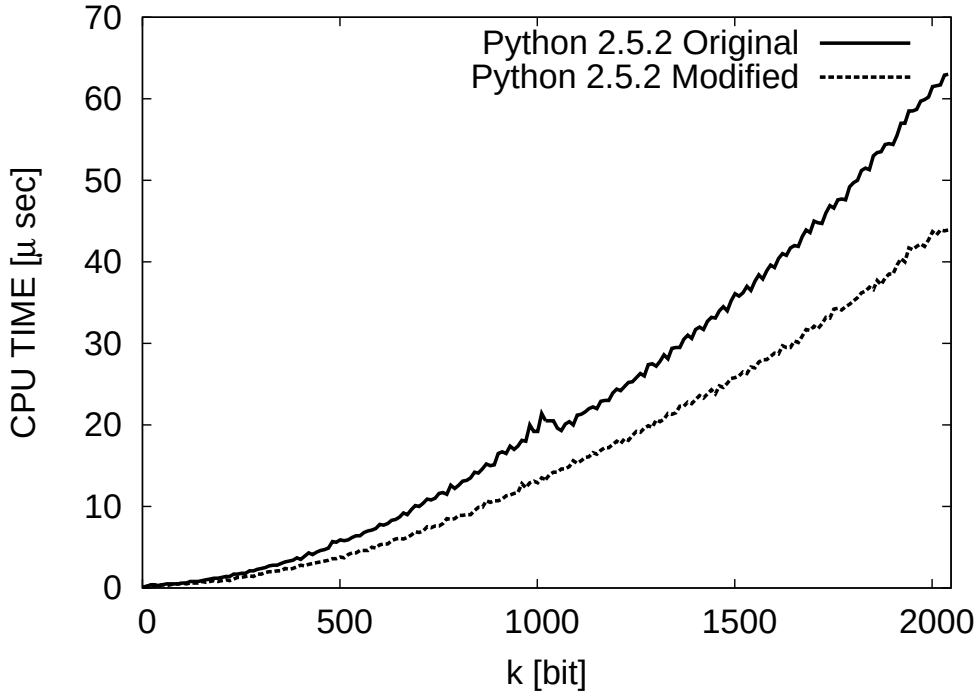
- `Include/longintrepr.h`
  - **桁サイズ**: `SHIFT`=15 → 16
- `Objects/longobject.c`
  - **マスク処理等の省略**: `t&`$(2^{15}-1)$ → `t`
  - **冪乗算における k-ary 法の窓幅**: 5bit → 4bit
  - **乗算における自乗計算部分削除**
- `Python/marshal.c`
  - `unsigned short` **型を扱う関数の追加**

→ **乗算・除算・冪乗算に対する効果を確認**

乗算の実行時間比較

実験結果：

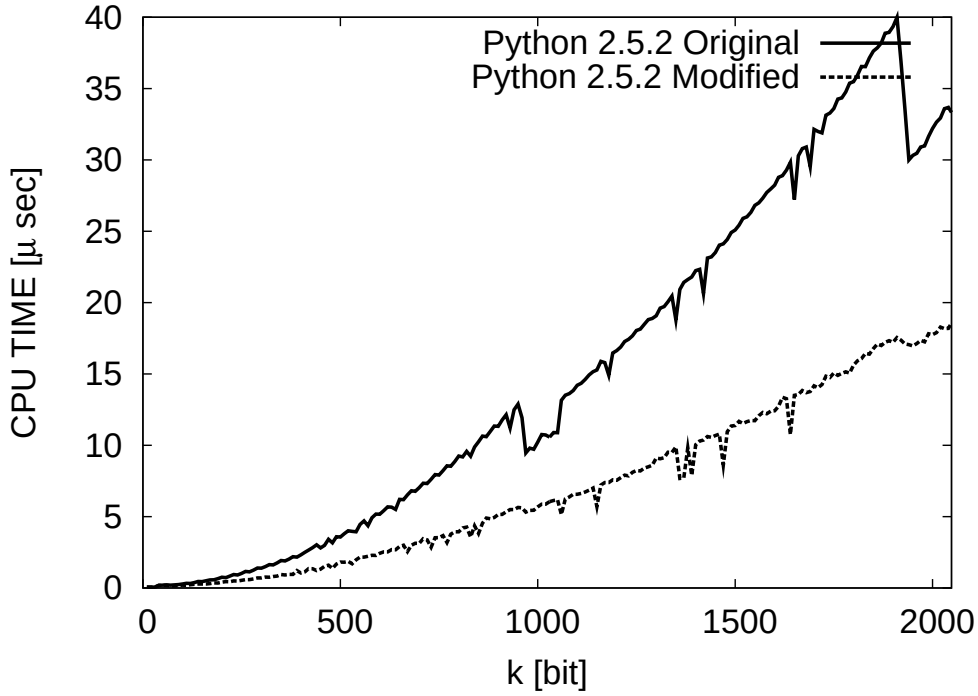
32bit 環境



Pentium 4 (2.66GHz)

gcc-4.2.1 on SUSE Linux

64bit 環境



Xeon 5365 (3.00GHz)

gcc-4.2.1 on SUSE Linux

$\Rightarrow$ 1.5〜2 倍程度高速化

除算の改良

Python：Knuth の除算アルゴリズムを利用するが、冗長な処理が多い

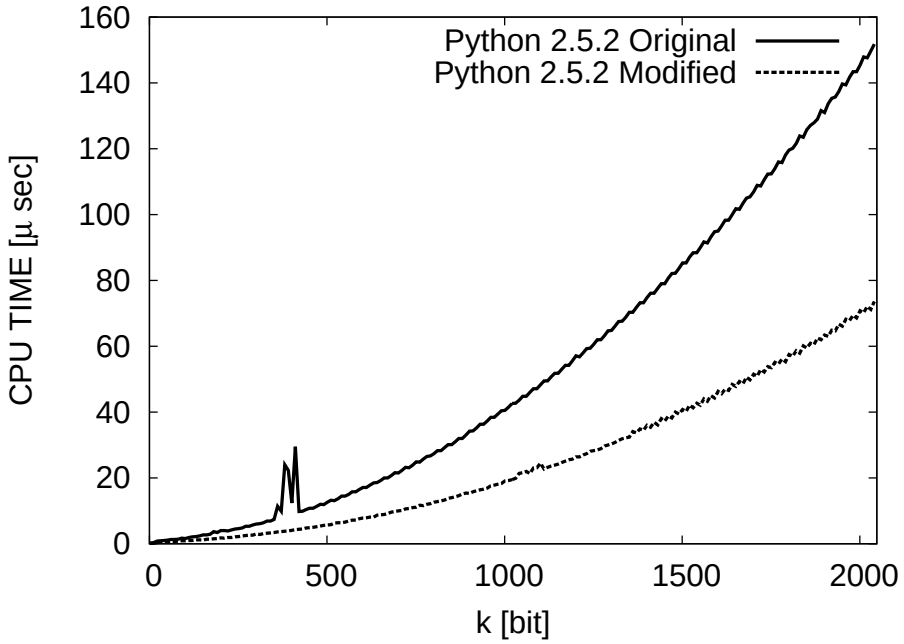
$\Rightarrow$ Knuth の除算アルゴリズムを再実装

- 32bit/word を考慮した実装

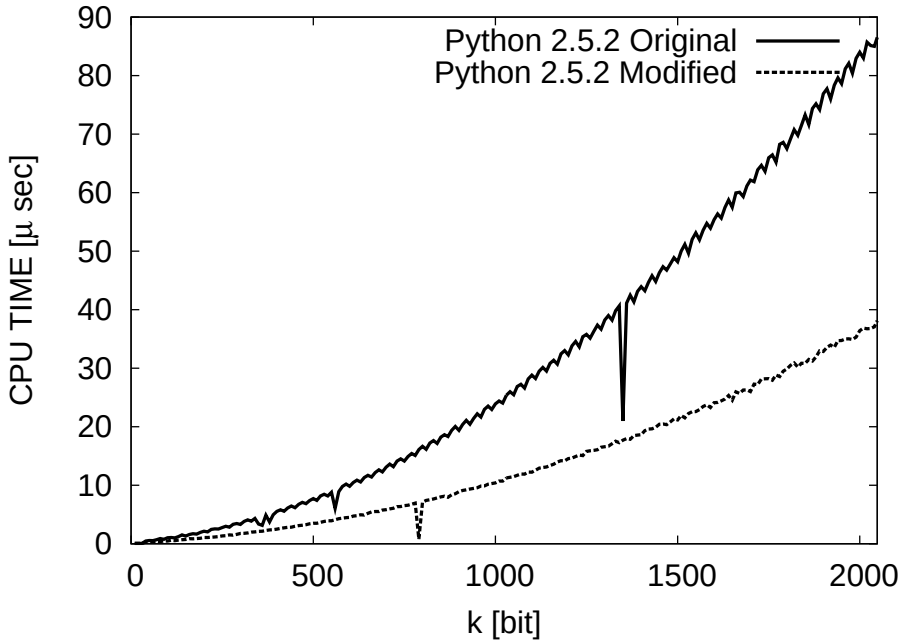
除算の実行時間比較

実験結果：

32bit 環境



64bit 環境



⇒ 2 倍以上高速化

法冪乗算に対する効果

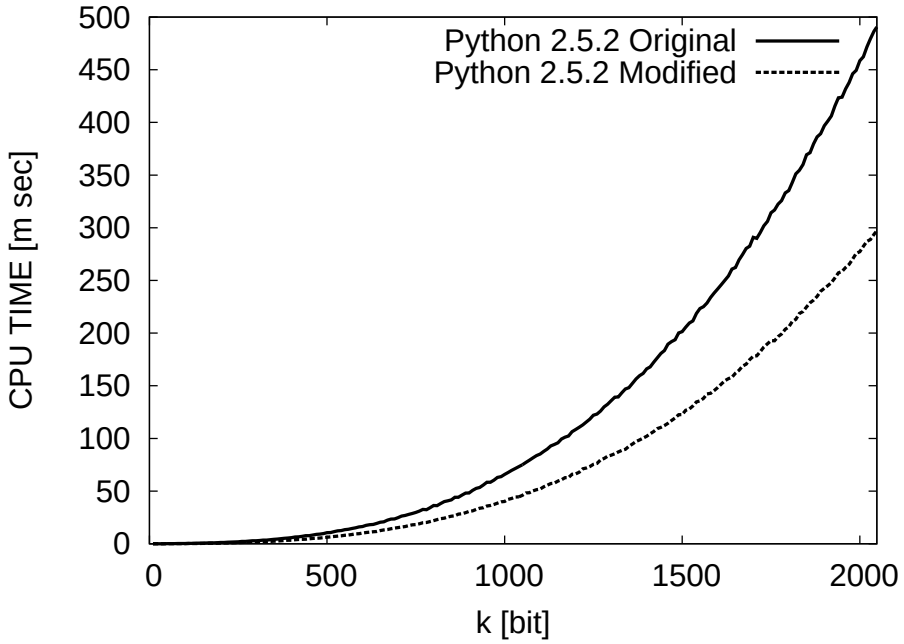
多倍長整数表現桁サイズの変更

- 乗算・除算の高速化 $\Rightarrow$ 法冪乗算の高速化
- k-ary 法の窓幅: 5bit → 4bit の影響

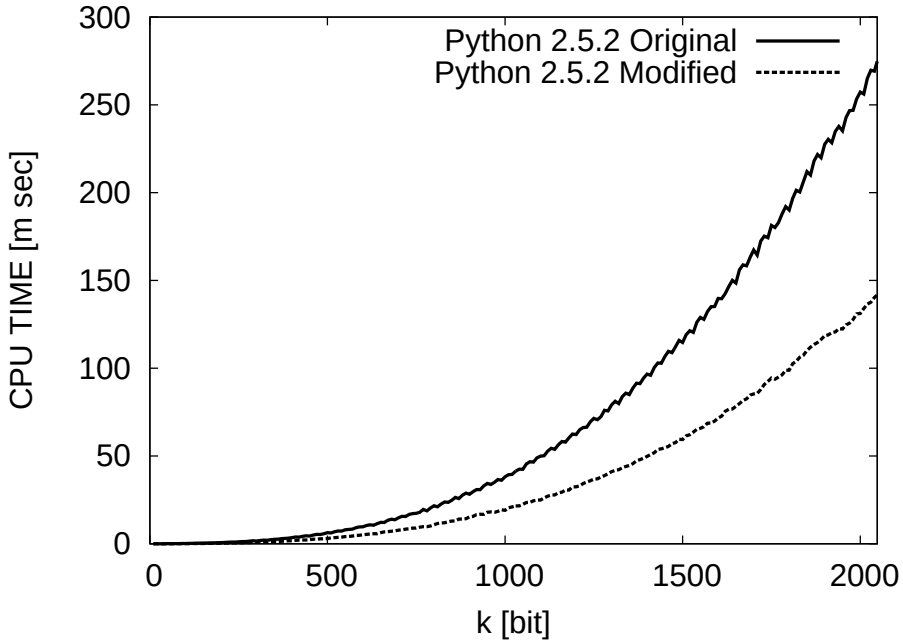
法冪乗算の実行時間比較

実験結果：

32bit 環境



64bit 環境



⇒ 約 1.5〜2 倍高速化

まとめ

- 多倍長整数表現の桁サイズ変更 15 → 16 ビット
- Knuth の除算アルゴリズムの効率的な実装

  $\Rightarrow$ 乗算・除算・法冪乗算の処理速度向上