Как ускорить Python?

В. В. Ащепкова

В рамках дисциплины «Дискретные структуры» перед нами стояла задача реализовать приложение с играми-задачами по разделу «Булевы функции». Создать его было решено на основе Python. Однако при увеличении размера входных данных скорости работы программ оказалось недостаточно для приложения, в котором пользователь будет запускать проверяющие алгоритмы не единожды. Таким образом, возникла необходимость ускорить их, например, через подходы модульного программирования.

Модульное программирование — организация программы как совокупности небольших независимых блоков (модулей). К ним относятся плагины и библиотеки. Первые — это программное обеспечение, которое добавляет функции в приложение. Вторые — наборы классов и функций, используемые компьютерной программой. Кроме того, существует два типа библиотек: статические и динамические. Программа обращается к статической — в процессе сборки, к динамической — во время выполнения.

Плюсы:

- Многократное использование;
- Независимая разработка;
- Удобная отладка и замена.

Минусы:

- Трудоемкость разработки;
- Расходы памяти больше;
- Время ЦП выше.

После выбора подхода разработчику следует подобрать язык программирования, оптимальный для решения поставленной задачи. В нашем случае — C++.

Для подключения C++ к Python созданы библиотеки: Boost.Python, pybind11, cpppy и др. Рассмотрим работу первой библиотеки (см. листинг 1-3).

Листинг 1. Создание функции

```
char const* greet()
{
    return “hello, world”;
}
```

Листинг 2. Объявление функции в качестве модуля Boost

```
#include <boost/python.hpp>
BOOST_PYTHON_MODULE(hello_ext)
{ using namespace boost::python;
  def(“greet”, greet);}
```

Листинг 3. Подключение сгенерированного модуля

```
>> import hello_ext
>> print hello_ext.greet()
hello, world
```

Для работы данной библиотеки достаточно компилятора C++ и интерпретатора Python, поэтому воспользоваться ею не составит труда. В ходе реализации проекта было интересно изучить ее принцип работы и попробовать подключить функции вручную.

Python написан на языке C, поэтому программы, написанные на C, можно подключить практически напрямую, C++ же библиотека преобразовывает самостоятельно. Для внедрения C++ необходимо представить содержание программы в виде C: в заголовочном файле связать с ним функции (см. листинг 4), в исходном коде заменить типы данных согласно C (см. листинг 5). Изменения коснутся и обертки, написанной на Python: с помощью библиотеки ctypes нужно подключить ранее скомпилированный файл и описать типы принимаемых и возвращаемых значений (см. листинг 6).

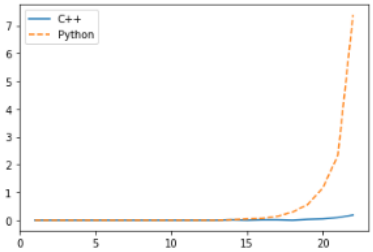
Листинг 4. Файл.hpp (частично)

```
std::string Fict(std::string line);
#ifdef __cplusplus
extern "C" {#endif
    char * Fict(char *line);
#ifdef __cplusplus
}#endif
```

Листинг 5. Файл.cpp (частично)

```
#include “*.hpp”
std::string Fict(std::string line)
//реализация
char * test5_Fict(test5 *test, char *line) {
std::string str = Fict(std::string(line));
char *result = new char[str.length()+1];
strcpy(result, str.c_str());
return result; }
```

Рисунок 1. Скорость работы модуля C++ и функции Python



Листинг 6. Файл.py (частично)

```
test_lib5=ctypes.CDLL(‘Полный путь
до файла’)
test_lib5.Fict.restype=ctypes.c_char_p
test_lib5.Fict.argtypes=[ctypes.c_char_p]
```

Готовое приложение работает правильно, подключение выполнено успешно. Для большого набора данных модуль на C++ ускорил программу более чем в 2 раза (см. рис 1).