Санкт-Петербургский государственный университет

Информационно-аналитические системы

Группа 22.Б10-мм

Связь моделей на языке Python с моделирующим ядром на C++

Баранов Артём Артурович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
Старший преподаватель кафедры ИАС, К. К. Смирнов

Консультант:
Разработчик ПО ООО «Софтком» И. А. Андреев

Санкт-Петербург
2024

Оглавление

Введение

Моделирующая система реального времени — это программное обеспечение, которое используется для создания моделей и симуляции процессов, происходящих в реальном времени. Она позволяет анализировать и предсказывать поведение системы на основе входных данных и параметров. Моделирующие системы реального времени являются важной частью разработки систем [7], применяемых в различных областях, включая управление различного транспорта, медицина [8], космическая отрасль, военно-промышленный комплекс и так далее.

В учебной практике работа связана с проектом “smm-2ng” от компании “ООО Софтком”, в основе которого лежит моделирующая система реального времени, способная работать в математическом и полунатурном режиме моделирования. Математический режим — режим, в котором моделируется и тестируется алгоритм, но не учитываются временные требования. Полунатурный режим — режим, который используется для тестирования аппаратного обеспечения с помощью использования различных алгоритмов. Моделирующая система реального времени написана на языке C++. В свою очередь, создание алгоритмов на C++ требует значительных усилий и времени [1].

Высокоуровневые языки, такие как Python, имеющие большую экосистему библиотек [4], способны ускорить прототипирование различного рода алгоритмов [5]. Однако у Python присутствует существенный недостаток — он медленный [9]. Однако несмотря на недостатки языка Python, заказчик требует возможности написания алгоритмов на нём. Кроме того, в ходе учебной практики рассматривается работа только с математическим режимом моделирования, который акцентирует внимание на алгоритмах, а не на скорости выполнения моделирования, что позволяет использовать Python.

Добавление возможности взаимодействия с алгоритмами на Python помогает ускорить решение необходимых задач. Это также позволяет заранее увидеть возможности разработанного алгоритма, чем в свою очередь облегчает написание алгоритма на C++ в будущем.

1. Постановка задачи

Целью работы является добавление возможности работы ядра smm-2ng с алгоритмами на Python. Для её выполнения были поставлены следующие задачи:

1. сделать обзор способов вызова методов на Python из C++;
2. разработать и реализовать решение для работы с кодом на Python;
3. измерить производительность добавленной функциональности.

2. Обзор

2.1. Вызов методов на Python из C++

На данный момент существует множество различных инструментов для вызова методов на Python из C++. Перед обзором этих инструментов также следует выделить некоторые критерии для выбора подходящего варианта, а именно:

1. легковесность — относительно небольшой размер библиотеки и ее зависимостей, не считая CPython;
2. минимизация boilerplate-кода — это код, который не несёт смысловой нагрузки, а является типовым кодом;
3. сведение к минимуму зависимостей от сторонних проектов;

Именно эти критерии являются наиболее необходимыми при создании необходимой функциональности на C++ для работы с Python в контексте работы с smm-2ng. Далее будет представлены наиболее известные и поддерживаемые инструменты для вызова Python методов из C++.

2.1.1. Boost.Python

Boost.Python — это библиотека, требующая предварительной установки [6]. Библиотека позволяет работать одновременно с двумя языками: Python и C++. Если говорить о преимуществах библиотеки, то стоит упомянуть широкую функциональность [3], описанную в документации[1], среди которой есть автоматическое преобразование типов. Что касается минусов, то это большой размер репозитория библиотеки, а именно 3.6 Мб. Кроме того, отсутствие регулярных обновлений, последнее обновление было год назад на github[2].Небольшой пример использования продемонстрирован в приложении 1, который показывает базовые концепции использования инструмента, необходимые в учебной практике.

[1]Документация Boost.Python: https://www.boost.org/doc/libs/1_84_0/*libs/python/doc*

[2]Github Boost.Python: https://github.com/boostorg/python

2.1.2. Pybind11

Pybind11 — это заголовочная библиотека, которая не требует каких-либо дополнительных установок, как Boost.Python. Авторы позиционируют Pybind11, как более легковесную библиотеку [2], чем Boost.Python, об этом они заявляют на github[3], у неё также имеется автоматическое преобразование типов. Размер репозитория библиотеки составляет 3.1 Мб. Кроме того, библиотека поддерживает C++20 стандарт, хотя изначально поддерживала только C++11. Из немногочисленных минусов можно выделить, что библиотека в основном нацелена на разработку Python кода с вызовом C++. Из недостатков: менее обширная функциональность, чем Boost.Python. Небольшой пример использования продемонстрирован в приложении 2, который показывает базовые концепции использования инструмента, необходимые в учебной практике.

2.1.3. CPython

CPython — это интерпретатор языка Python. Ключевым преимуществом CPython является отсутствие каких-либо зависимостей от внешних проектов. Несравненным плюсом CPython можно выделить, что идёт взаимодействие с API Python на низком уровне, то есть нет никаких “обёрток”. Если говорить о недостатках, то это скудность функциональности, особенно отсутствие автоматического преобразования данных между Python и C++, в отличие от Boost.Python и Pybind11. Небольшой пример использования Небольшой пример использования продемонстрирован в приложении 3, который показывает базовые концепции использования инструмента, необходимые в учебной практике.

По результатам обзора была составлена таблица 1

[3]Github Pybind11: https://github.com/pybind/pybind11

Таблица 1: Преимущества и недостатки инструментов

	Boost.Python	Pybind11	CPython
Поддержка	Сторонняя библиотека	Сторонняя библиотека	Часть Python
Легковесность	3.6 Мб	3.1 Мб	Часть Python
Минимизация boilerplate-кода	+	+	-

Из приведенных выше высказываний становится ясно, что использование Pybind11 является наилучшим выбором, так как проигрывает только в том, что проект является сторонним к CPython. В свою очередь у других вариантов тоже имеются неоспоримые преимущества. Однако широкая функциональность Boost.Python нам не нужна, а сложности разработки на CPython видны уже на старте, они выражаются малой функциональностью и сложным освоением. Однако в ходе разработки вариант Pybind11 показал свои скрытые недостатки, которые выражаются в том, что большая часть документации[4] посвящена вызову C++ кода из Python, однако в учебной практике необходимо обратное. В связи с этим был выбран CPython.

2.2. Работа с плагинами в моделирующей системе реального времени smm-2ng

В основе моделирующей системы лежит ядро, написанное на C++, оно способно работать в нескольких режимах. Однако научная практика строится только на одном — математическом. В основе этого режима лежит тестирование алгоритма, но не его скорости исполнения. Для того, чтобы работать с алгоритмами в ядре реализован базовый класс — AbstractAlgorithm. От него пользователь реализует свой собственный алгоритм, который представляет собой плагин. Кроме этого, пользователь пишет конфигурацию алгоритма и его параметров, а именно название аргументов и их конкретные значение, всё это передаётся ядру

[4]Документация Pybind11 https://pybind11.readthedocs.io/en/latest/

в виде json файла. Со стороны ядра также реализован специальный макрос, который позволяет получить тип передаваемых аргументов в реализованный алгоритм.

Далее на рисунке 1 представлено устройство smm-2ng в контексте работы с алгоритмами

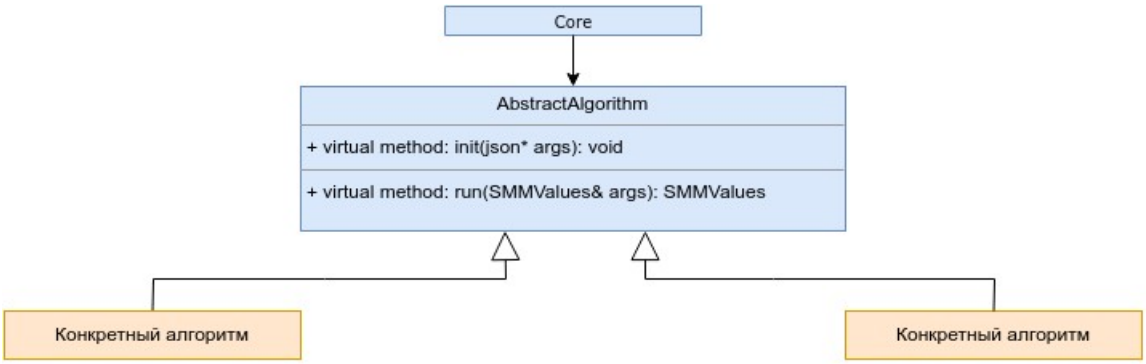

Рис. 1: Работа с плагинами в smm-2ng

3. Реализация

Ключевой целью является добавление новой функциональности на C++, которая бы позволила из C++ работать с Python алгоритмами, написанными пользователями.

В ходе обзора был выбран CPython, как инструмент, позволяющий работать из C++ с Python. Само же решение основывается на добавлении дополнительного плагина, позволяющего работать с Python алгоритмами. Как и другие плагины он перереализует основные методы AbstractAlgorithm, наиболее важными и сложными из которых являются init и run. Первый метод позволяет создать экземпляр Python класса для последующего использования. Второй же метод запускает одноименный метод, но уже из Python класса. Кроме того для этого метода есть дополнительные функции, которые необходимы для преобразования C++ типов в PyObject*, а также для обратного преобразования из PyObject* в C++ соответственно. Более подробно с кодом можно ознакомиться в репозитории[1].

Далее на рисунке 2 представлена архитектура проекта, где зелёном цветом выделена проделанная работа в ходе учебной практики, синим отмечено устройство ядра и оранжевым код, который реализует пользователь системы.

[1]Ссылка на gitlab с кодом: https://gitlab.softcom.su/artem.baranov/smm-2ng-py

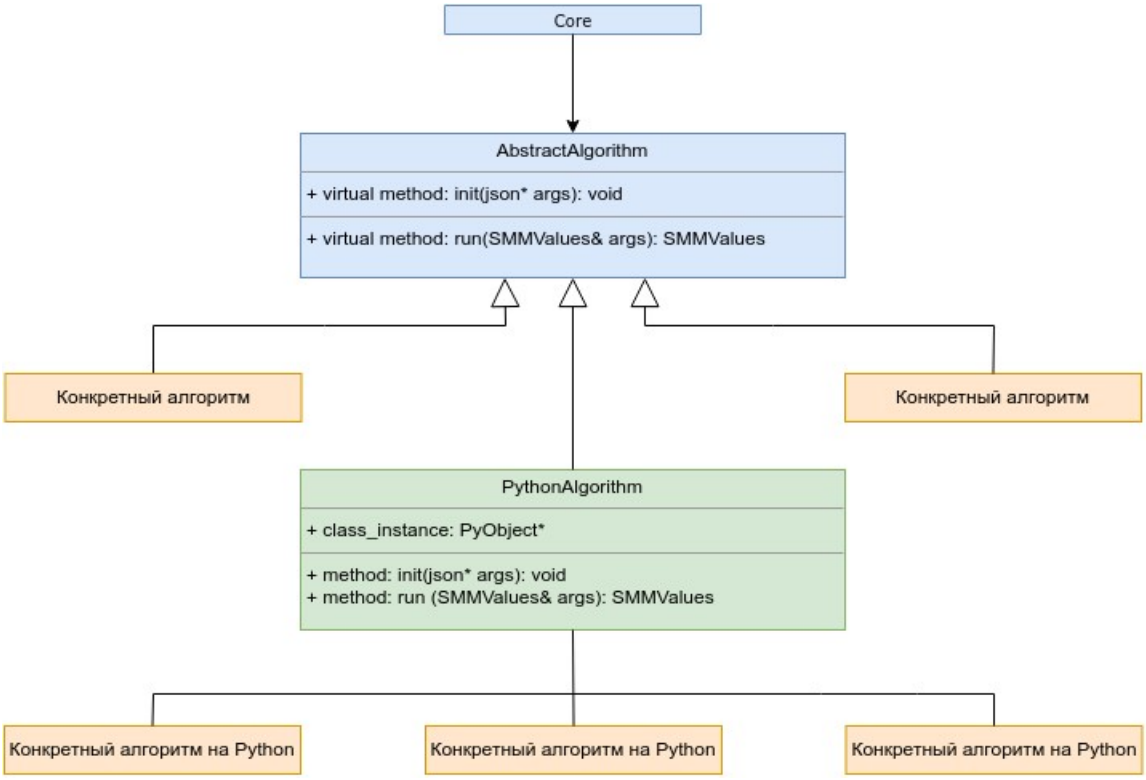

Рис. 2: Архитектура проекта

4. Эксперимент

В ходе учебной практики было проведено сравнение производительности ядра с использованием ряда плагинов:

- producer — плагин, который сохраняет полученные данные в ходе инициализации и возвращает их при работе;
- combiner — плагин, который объединяет несколько блоков данных в один;
- consumer — плагин, который принимает данные.

Далее идут схемы, которые показывают связь плагинов. На первой схеме 3 представлены плагины, отмеченные синим цветом и написанные на C++, разработанные компанией “ООО Софтком” ещё до учебной практики. На второй схеме 4 представлен переработанный плагин Producer, отмеченный зелёным цветом и переписанный в ходе учебной практики. Он имеет ту же функциональность, что и изначальный плагин, написанный ещё до учебной практики.

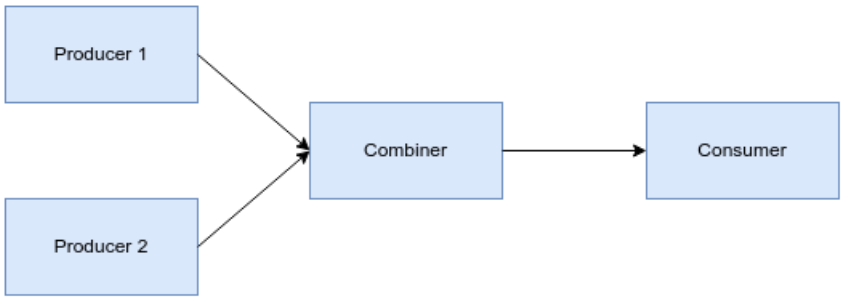

Рис. 3: Исходная схема тестового примера

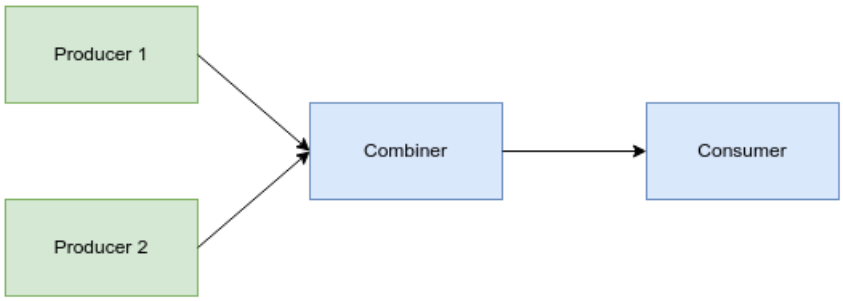

Рис. 4: Доработанная схема с плагинами на Python, они отмечены зелёным цветом

В ходе сравнения эффективности был построен график, его можно увидеть на рисунке 5. Сравнение проходило с помощью команды “time” в Debian, которая используется для измерения времени, затраченного на выполнение программы. Рассматривалась строка “real”, в которой прописано общее время, затраченное на выполнение программы. На каждую точку приходится 10 замеров. Для каждой серии экспериментов среднеквадратичное отклонение не превышает 2%.

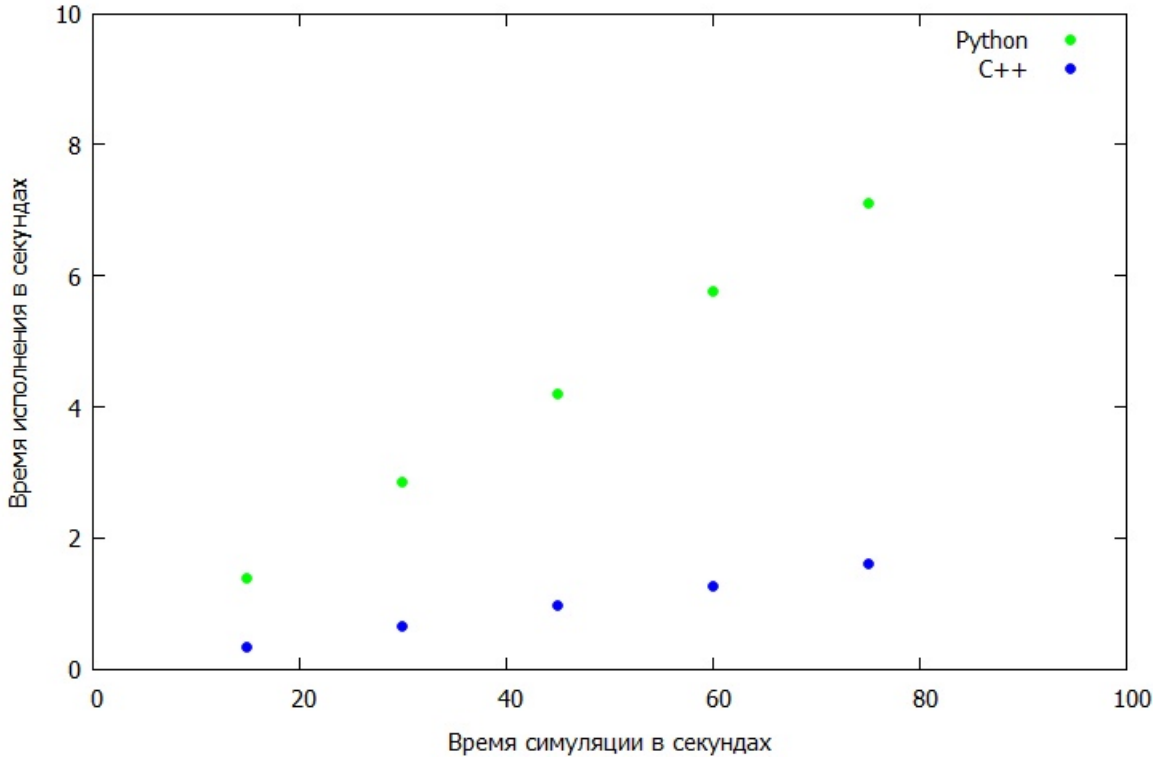

Рис. 5: График, составленный в ходе эксперимента

В ходе эксперимента было задействовано следующее оборудование:

- процессор — AMD Ryzen 5 5625U with Radeon Graphics 2.30 GHz;
- операционная система — Debian GNU/Linux 12 (bookworm);
- оперативная память — 8,00 ГБ.

В результате эксперимента выяснилось, что данная схема с использованием языка Python медленнее оригинальной схемы примерно в 4 раза. Несмотря на то что алгоритмы на Python показывают более продолжительное время работы по сравнению с аналогичными на C++,

заказчика устраивает данная производительность для прототипирования моделей.

5. Заключение

По итогам учебной практики были выполнены следующие задачи:

- проведён обзор библиотек, позволяющих использовать Python из C++;
- выбран наиболее подходящий инструмент;
- добавлена возможность работы с Python алгоритмами;
- измерена производительность добавленной функциональности.

Более подробно с кодом можно ознакомиться в репозитории[2].

[2]Ссылка на gitlab с кодом: https://gitlab.softcom.su/artem.baranov/smm-2ng-py

Список литературы

[1] Grosse-Kunstleve Ralf W. why choose python. — URL: https://scfbm.biomedcentral.com/articles/10.1186/1751-0473-7-5 (дата обращения: 12 2023 г.).

[2] Lyons Damian M. Pybind11 a light weight library. — URL: https://arxiv.org/ftp/arxiv/papers/1906/1906.08351.pdf (дата обращения: 1 января 2024 г.).

[3] Pecevski Dejan. Boost.Python have big functional. — URL: https://www.frontiersin.org/articles/10.3389/neuro.11.011.2009/full (дата обращения: 1 января 2024 г.).

[4] Pike Thomas. Python libraries. — URL: https://arxiv.org/ftp/arxiv/papers/2201/2201.06040.pdf (дата обращения: 1 января 2024 г.).

[5] Rubinsteyn Alex. Python prototype. — URL: https://www.usenix.org/system/files/conference/hotpar12/hotpar12-final37.pdf (дата обращения: 1 января 2024 г.).

[6] Shen Guobao. Boost.Python should be installed. — URL: https://accelconf.web.cern.ch/icalepcs2011/papers/wepks021.pdf (дата обращения: 1 января 2024 г.).

[7] Sifakis Joseph. Modeling system in real time. — URL: https://www.researchgate.net/publication/4117414_Modeling_Real-Time_Systems (дата обращения: 11 2023 г.).

[8] Simonov Michael. Real-time modeling system in medicine. — URL: https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1002861 (дата обращения: 12 2023 г.).

[9] Srinath K.R. Slow Python. — URL: https://www.irjet.net/archives/V4/i12/IRJET-V4I1266.pdf (дата обращения: 12 2023 г.).

Приложение

Листинг 1: Пример кода с использованием Boost.Python

```
#include <boost/python.hpp>
#include <iostream>

using namespace boost::python;
using namespace std;

int main() {
    Py_Initialize();
    try {
        object module = import("module");
        object multiplyNumbersFunc = module.
               attr("multiply_numbers");
        object result = multiplyNumbersFunc(2, 3);
        int intResult = extract<int>(result);
        cout << "Multiply: " << intResult << endl;
    }
    catch (error_already_set) {
        PyErr_Print();
    }
    Py_Finalize();
    return 0;
}
```

Листинг 2: Пример кода с использованием Pybind11

```
#include <iostream>
#include <pybind11/embed.h>

using namespace std;

int main() {
```

```
    pybind11::scoped_interpreter guard{};
    pybind11::module pModule = pybind11::module::import("module");
    pybind11::object result = pModule.
                    attr("multiply_numbers")(4, 4);
    int value = result.cast<int>();
    cout << value << endl;
    return 0;
}
```

Листинг 3: Пример кода с использованием CPython

```
#include <Python.h>

using namespace std;

int main() {
    PyObject *pModule, *pFunction, *pArgs, *pResult;
    Py_Initialize();
    pModule = PyImport_Import(PyUnicode_FromString("module"));
    pFunction = PyObject_GetAttrString(pModule,
                "multiply_numbers");
    pArgs = Py_BuildValue("(ii)", 2, 3);
    pResult = PyObject_CallObject(pFunction, pArgs);
    long value = PyLong_AsLong(pResult);
    Py_DECREF(pModule);
    Py_DECREF(pFunction);
    Py_Decref(pArgs);
    Py_Decref(pResult);

    Py_Finalize();

    printf("%ld\n", value);

    return 0;
```

```
}
```