

Министерство общего и профессионального образования РФ
Московский Авиационный Институт
(Национальный Исследовательский Университет)



Факультет информационных технологий и прикладной математики
Кафедра математической кибернетики

Курсовой проект

**по курсу «Параллельное и распределённое
программирование»**

**на тему «Многопоточные вычисления на Python применительно
к задаче многомерной интерполяции радиальными базисными
функциями»**

II семестр

Студент: Кондаратцев В.Л.; Бобошина А.В.

Группа: 8О-105 М

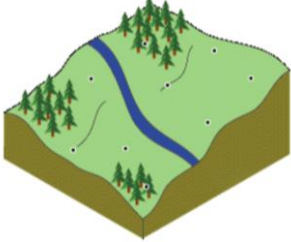
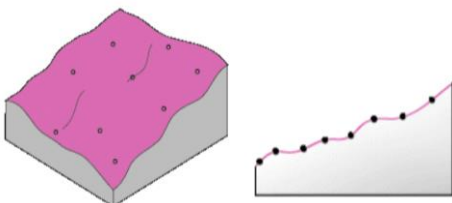


Преподаватель: Пановский В.Н.

Дата:

Оценка:

1. Практические истоки постановки задач

В самых различных задачах инженерии и анализа данных возникает необходимость восстанавливать функцию по известным точечным значениям (интерполяция, аппроксимация), например, для восстановления рельефа местности по точечным измерениям геодезистов возникает потребность в методе, который наиболее реалистично и менее ресурсозатратно выполнял бы эту задачу. Таким методом является метод интерполяции с помощью радиальных базисных функций (RBF-метод), [1]-[3]. Отметим также, что RBF-метод может быть применён не только к задачам геодезии, но активно использоваться в области анализа данных (RBF-функция в нейронных сетях [4]).

Данные, подающиеся на вход	Данные, полученные на выходе после RBF-интерполяции	Описание
		Задача восстановления рельефа местности по точечным геодезическим измерениям.
		Задача создания твердотельной модели объекта по его полигональной модели.

2. Математическая постановка задачи

Восстанавливать поверхность по точкам будем в виде функции

$$s(x) = \sum_{j=1}^N \lambda_j \varphi(|x - x_j|) + P(x), \quad x \in \mathbb{R}^d \quad (1),$$

где функция $\varphi : [\mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}]$ является RBF-функцией; λ_j – коэффициентами; $P(x)$ – глобальная полиномиальная функция, не превосходящая степень $K - 1$.

Коэффициенты $\{\lambda\}$ выбираются так, чтобы удовлетворять N подходящим условиям

$$s(x_j) = f(x_j), \quad j = 1, \dots, N,$$

а ограничения $\sum_{j=1}^N \lambda_j P(x_j) = 0$ для всех многочленов P , не превосходящие степень $K - 1$.

Тогда соответствующая система уравнений, которую нужно решить, чтобы найти коэффициенты λ и многочлен P задаётся в виде:

$$\begin{pmatrix} \Phi & P^T \\ P & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ b \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}, \quad (*)$$

где

$$\Phi_{ij} = \varphi(x_i - x_j), \quad i, j = 1, \dots, N;$$

$$P_{ij} = P_j(x_i), \quad i = 1, \dots, N; \quad j = N + 1, \dots, N + K.$$

Обычно степень K многочлена достаточно мала, и часто даже равняется 1. Тогда $P(x) = \text{const}$.

Популярными выборами являются следующие классы RBF-функций:

Выражение для $\varphi(x)$	Название	Замечания
r	Бигармоническая	\mathbb{R}^3
r^{2n+1}	Полигармоническая	\mathbb{R}^3
$(r^2 + c^2)^{1/2}$	Функция мультиквадриков	\mathbb{R}^d , наиболее часто используемая
$(r^2 + c^2)^{-1/2}$	Обратная функция мультиквадриков	В некоторых случаях более эффективная, чем функция мультиквадриков
$r^2 \log \frac{r}{r_0}$	Тонкий сплайн (для пластины)	Идеален для проблемы деформации пластины, в других случаях даёт нормальные результаты
$\exp\left(-\frac{1}{2} \frac{r^2}{r_0^2}\right)$	Гауссовский	Сложно, но реально получить высокую точность; быстро сходится к 0.

Задача состоит в том, чтобы восстановить коэффициент λ по системе (*).

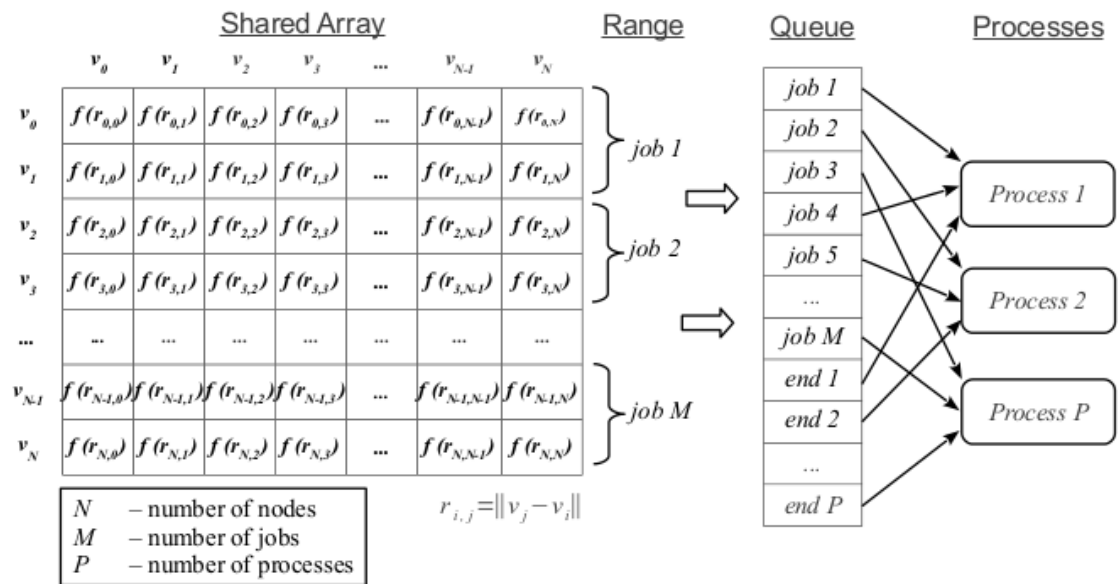
3. Программная реализация и изучение проблемы Python GIL

Для программного реализации метода был выбран язык программирования Python 3. Численные операции были выполнены с помощью библиотеки NumPy, а графики были построены с помощью Matplotlib. В качестве инструмента, позволяющего распараллелить проект, был выбран метод множества процессов использующих общую кучу памяти (использовалась библиотека multiprocessing).

Основной проблемой при реализации на Python программ использующих несколько потоков для распараллеливания процесса вычислений является Python Global Interpretation Lock (GIL). Если кратко, то GIL не позволяет в одном интерпретаторе Python эффективно использовать больше одного потока. Защитники GIL утверждают, что однопоточные программы при наличии GIL работают намного эффективнее. Но наличие GIL означает, что параллельные вычисления с использованием множества потоков и общей памяти невозможны. А это достаточно сильное ограничения в современном многоядерном мире.

Однако существуют способы преодоления ограничений GIL при помощи потоков. Выбранный способ основан на multiprocessing и shared array. Способ позволяет достаточно просто и эффективно

использовать процессы и разделяемую память для прозрачного параллельного программирования в стиле множества потоков и общей памяти.



В данной задаче каждая строка матрицы может вычисляться независимо. Из каждой нескольких строк матрицы сформируем независимые работы и поместим их в очередь заданий. Запустим несколько процессов. Каждый процесс будет брать из очереди следующее задание на выполнение, пока не встретит специальное задание с кодом «end». В этом случае процесс заканчивает свою работу.

В реализации на Python у нас будут два главных метода: `mpCalcDistance(nodes)` и `mpCalcDistance_Worker(nodes, queue, arrD)`.

Метод `mpCalcDistance(nodes)` принимает на вход список узлов, создает область общей памяти, подготавливает очередь заданий и запускает процессы. Метод `mpCalcDistance_Worker(nodes, queue, arrD)` – это вычислительный метод, работающий в собственном потоке. Он принимает на вход список узлов, очередь заданий и область общей памяти.

4. Анализ результатов

При апробации программы на разных ЭВМ с разными операционными системами наблюдаются следующие эффекты:

- Количество узлов, для которого среднее время расчёта несколькими процессами становится меньше чем для одного процесса, для разных ЭВМ и операционных систем разное. Это объясняется разной частотой процессоров и разной архитектурой операционных систем.

- Для одного и того же числа узлов, время выполнения и многопроцессорно и однопроцессорно может отличаться от запуска к запуску, что на графиках отображается в виде случайных пиков. Это можно объяснить особенностью устройства библиотеки `multiprocessing` и

эффектами «перекеширования» кучи в оперативной памяти. Несмотря на этот недостаток, в среднем время выполнения задачи для множества процессоров асимптотически меньше чем для одного процессора.

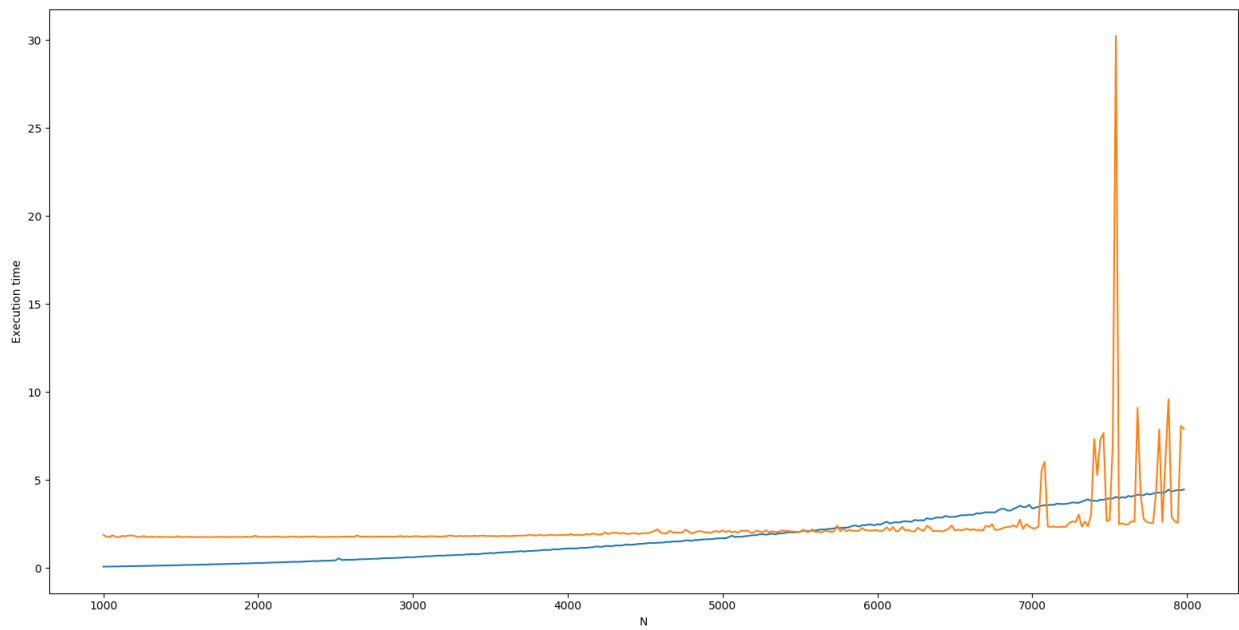


Рис.1

Зависимость времени расчёта для одного процессора (синий график) и для двух (жёлтый). Windows 7, 4CPU, 3GHz, 64bit.

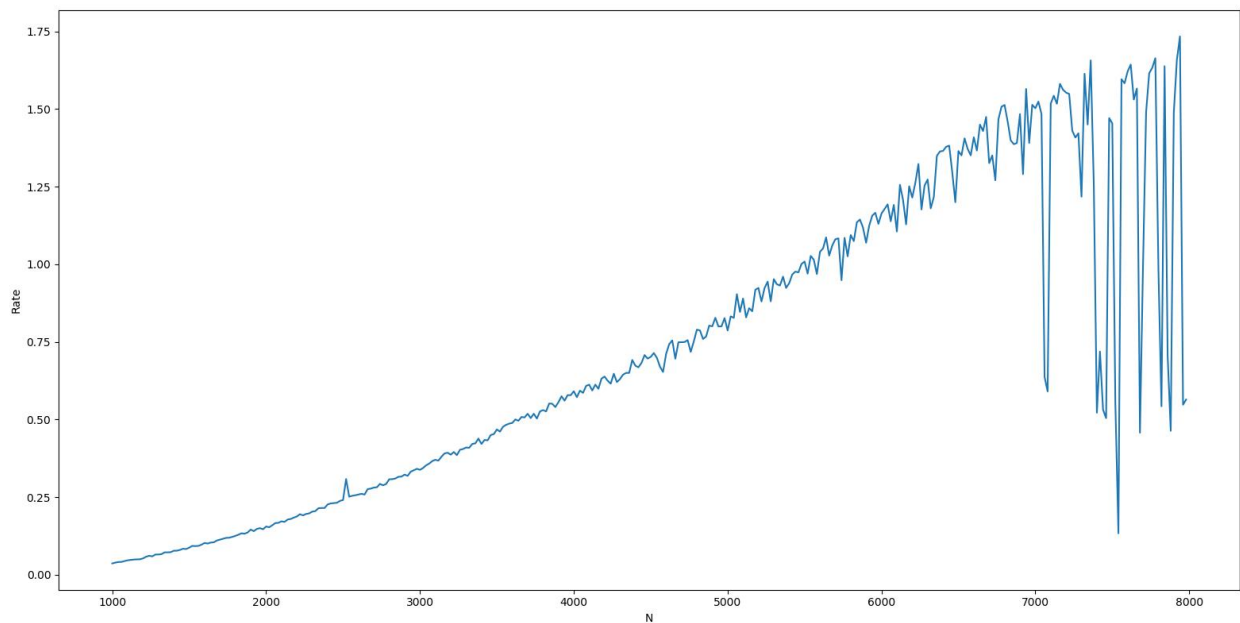


Рис.2

Зависимость отношения времени расчёта двух процессоров к одному. Windows 7, 4CPU, 3GHz, 64bit.

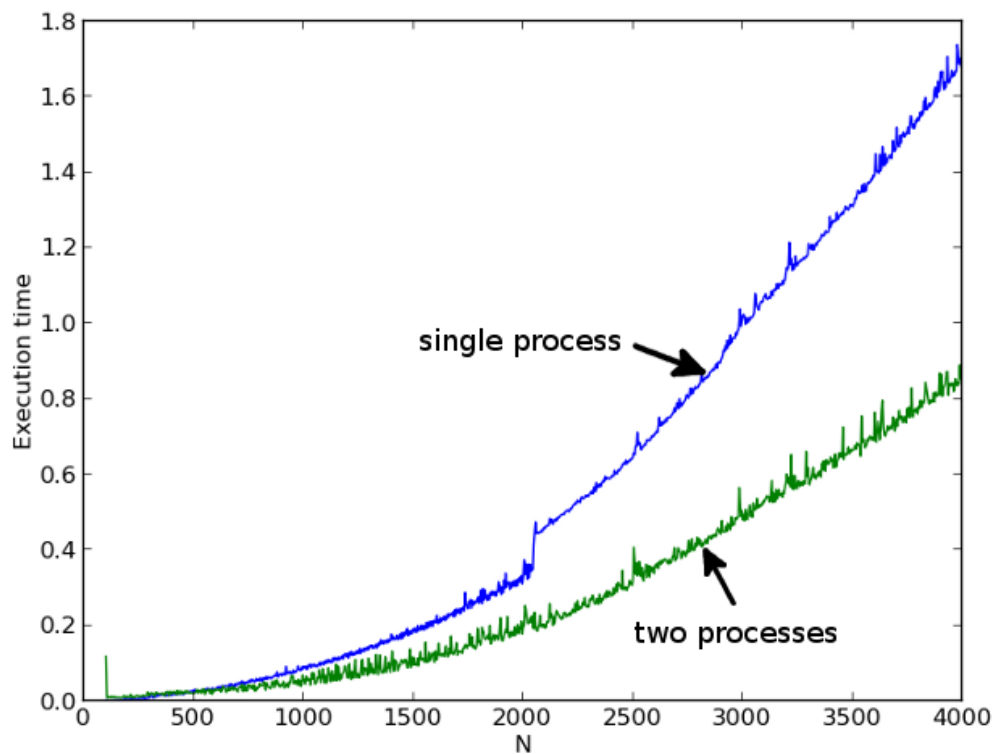


Рис.3

Зависимость времени расчёта для одного процессора (синий график) и для двух (жёлтый). Ubuntu 12.04, 64bit, 4CPU, 2.4GHz.

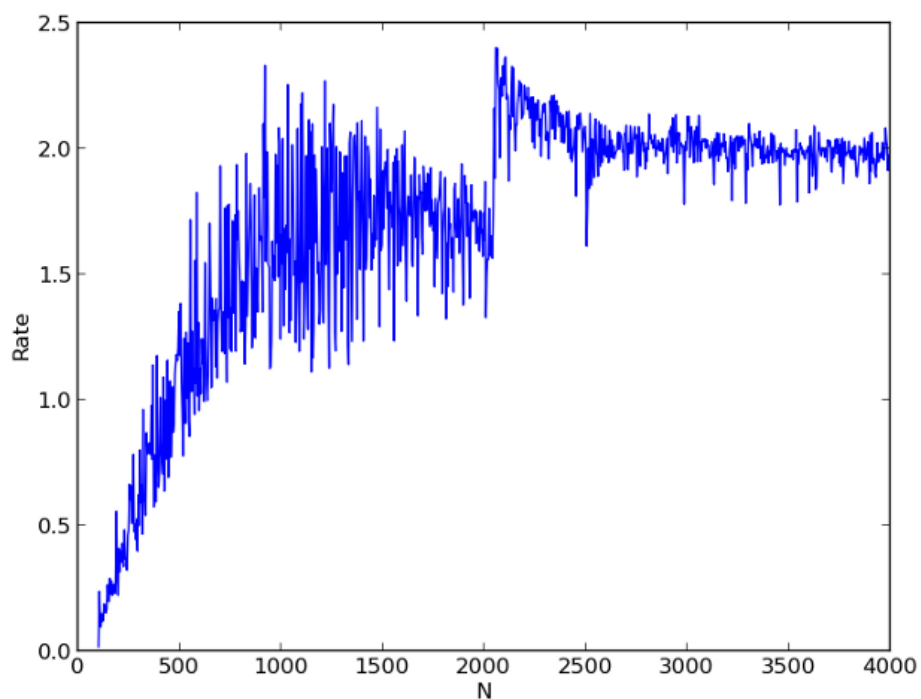


Рис.4

Зависимость отношения времени расчёта двух процессоров к одному. Ubuntu 12.04, 64bit, 4CPU, 2.4GHz.

5. Список использованных источников

- [1] Buhmann, Martin D. (2003), Radial Basis Functions: Theory and Implementations, [*Cambridge University Press*](#), [*ISBN 978-0-521-63338-3*](#).
- [2] Fast radial basis function interpolation via preconditioned Krylov iteration. Nail A. Gumerov, Ramani Duraiswami. University of Maryland, College park
- [3] Презентация «Radial basis function interpolation», Kim Day, Jessie Twigger
- [4] - https://en.wikipedia.org/wiki/Radial_basis_function