DOI: http://dx.doi.org/10.15688/jvolsu1.2015.4.2

УДК 514.142.2+514.174.6
ББК 32.973.26-018.2

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ГЕОМЕТРИЧЕСКОГО ХЕШИРОВАНИЯ НА ОСНОВЕ ПАКЕТА NUMPY И ПУЛА ПРОЦЕССОВ[1]

Владимир Александрович Клячин

Доктор физико-математических наук,
заведующий кафедрой компьютерных наук и экспериментальной математики,
Волгоградский государственный университет
klchnv@mail.ru, kiem@volsu.ru
просп. Университетский, 100, 400062 г. Волгоград, Российская Федерация

Аннотация. В статье рассматривается задача многомерного геометрического хеширования. Предлагается способ построения соответствующей хеш-матрицы параллельным алгоритмом. В работе построен алгоритм параллельного геометрического хеширования с использованием шаблона «пул процессов». Реализация алгоритма выполнена с применением языка программирования Python и пакета NumPy для манипулирования многомерными данными. В качестве основы для пула процессов предложено использовать класс ProcessPoolExecutor модуля concurrent.futures, который входит в дистрибутив интерпретатора Python начиная с версии 3.2. Все решения представлены в статье соответствующими UML-диаграммами классов. Найденное обобщенное программное решение может быть использовано для реализации параллельных алгоритмов и других задач, которые могут быть описаны в терминах схемы пула процессов.

Ключевые слова: хеширование, пул процессов, пакет NumPy, вычислительная геометрия, параллельный алгоритм.

Введение

Геометрическое хеширование — это метод, позволяющий ускорить доступ к неупорядоченному массиву данных, основанный на их геометрическом представлении. Использование этого метода широко распространено в задачах вычислительной геометрии

[5; 12], алгоритмах компьютерной графики [13], геоинформационных системах [12]. Метод геометрического хеширования хорошо описан, в частности, в статьях [14; 15; 17], в которых указанный метод применяется в задачах распознавания объектов на изображениях. Мы приведем его описание на примере задачи локализации точки относительно триангуляции.

Пусть $p_1, ..., p_N \in [a, b] \times [c, d] \subset \mathbb{R}^2$ — конечный набор точек на плоскости и $T = \{T_1, ..., T_M\}$ — его триангуляция — такой набор треугольников, что:

1) каждая точка p_i заданного набора является вершиной одного из треугольников $T_k \in T$;

2) каждая вершина любого треугольника $T_k \in T$ является одной из точек $p_i, i = = 1, ..., N$;

3) внутренность пересечения любых двух треугольников пуста.

Задача локализации точки относительно триангуляции ставится как задача определения тех треугольников триангуляции, которым может принадлежать заданная точка $p \in \mathbb{R}^2$ [9]. Простейший алгоритм решения этой задачи сводится к перебору всех тругольников и определения принадлежности точки p каждому такому треугольнику. Сложность алгоритма — $O(M)$. Основная идея оптимизации прямого перебора сводится к отбрасыванию заведомо невозможных случаев: не рассматривать те треугольники, которым заданная точка заведомо не принадлежит. Покажем, как алгоритм оптимизируется с помощью специальной предобработки данных — геометрического хеширования.

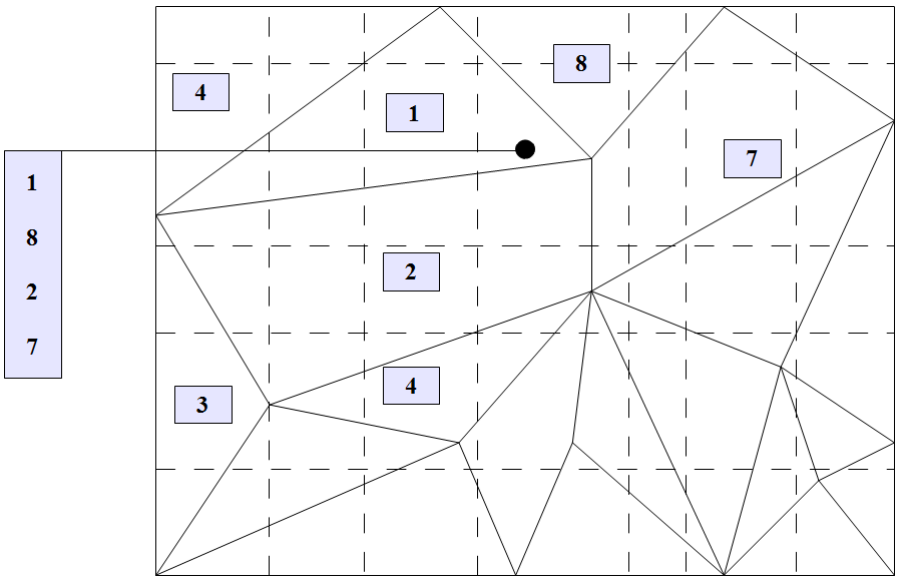

Рис. 1. Иллюстрация хеширования по двумерной сетке

Построим разбиения:

$$a = x_0 < x_1 < ... < x_m = b;$$

$$c = y_0 < y_1 < ... < y_m = d.$$

Далее каждой паре $(i, j), 1 \leq i \leq m, 1 \leq j \leq m$ сопоставим список номеров $h_{ij}^1, ..., h_{ij}^{n_{ij}}$ тех треугольников, которые пересекают прямоугольник $Q_{ij} = [x_{i-1}, x_i] \times [y_{j-1}, y_j]$

$$T_{h_{ij}^k} \cap Q_{ij} \neq \emptyset.$$

Для каждой пары (i, j) количество n_{ij} таких треугольников может быть различным. Теперь вернемся к задаче локализации точки. Новый алгоритм сводится теперь к следующим шагам:

1. Бинарным поиском определяем номера i, j такие, что $p \in Q_{ij}$.

2. За n_{ij} шагов находим треугольники из списка $\{T_{h_{ij}^k}\}, k = 1, ..., n_{ij}$, которые содержат точку p.

Сложность такого алгоритма порядка величины

$$C(\log_2 m + n_{ij}).$$

Оценим среднее значение n_{ij} длин списков номеров треугольников. Пусть

$$d = \min_{1 \le i,j \le m} \{x_i - x_{i-1}, y_j - y_{j-1}\}.$$

Обозначим через δ_k диаметр треугольника T_k. Тогда из простых геометрических соображений легко получить, что число m_k тех ячеек Q_{ij}, которые пересекает треугольник T_k не превосходит

$$m_k \le \left(\left[\frac{\delta_k}{d} \right] + 1 \right)^2.$$

Исходя из этого, получаем для среднего значения

$$\frac{1}{m^2} \sum_{ij} n_{ij} = \frac{1}{m^2} \sum_k m_k \le \frac{M}{m^2} \left(\left[\frac{\delta}{d} \right] + 1 \right)^2,$$

где

$$\delta = \max_k \delta_k.$$

Теперь сложность алгоритма локализации точки с использованием хеширования в среднем будет оцениваться величиной

$$C(\log_2 m + \frac{\Delta M}{m^2}),$$

где

$$\Delta = \left(\left[\frac{\delta}{d} \right] + 1 \right)^2.$$

Минимизируя это выражение относительно m, окончательно получим сложность алгоритма, в среднем равную $O(\log_2 M)$, при условии ограниченности величины Δ. Из приведенных рассуждений видно, что если применять равномерную сетку для хеширования, то оценка получается еще лучше: порядка $O(1)$. По сравнению с оценкой $O(M)$ это очень хорошо. Таким образом, геометрическое хеширование значительно ускоряет решение задачи локализации точки. Правда, при этом дополнительно затрачивается память порядка $O(M)$. Мы ставим задачу построения и реализации параллельного алгоритма геометрического хеширования.

1. Математическая модель геометрического хеширования

Пусть $X = \{x_1, ..., x_M\}, M \geq 1$ — произвольное конечное множество и K обозначает единичный интервал $K = [0, 1]$.

Пусть дано некоторое отображение

$$s : X \to 2^{K^n}, \quad n \geq 1.$$

Так что каждой точке $x \in X$ ставится в соответствие некоторое подмножество n-мерного единичного куба $s(x) \subset K^n$. Непосредственно в задаче локализации точки относительно триангуляции в качестве X рассматривается множество всех треугольников триангуляции T, расположенных, например, в единичном квадрате $[0, 1] \times [0, 1]$. Для случая $n = 1$ отображением s может служить отображение, которое каждому треугольнику ставит в соответствие отрезок проекции этого треугольника на ось Ox. При $n = 2$ мы можем взять в качестве s отображение, сопоставляющее треугольнику прямоугольник, являющийся прямым произведением двух отрезков проекций треугольника — один на ось Ox, другой — на ось Oy.

Помимо отображения s, предположим задана матрица разбиений $B = ||t_{ij}||$ размерности $n \times (m + 1), m \geq 1$. При этом предполагаем выполнение условий

$$0 = t_{i0} < t_{i1} < t_{i2} < ... < t_{im} = 1, \quad \forall i = 1, ..., n.$$

Элементы этой матрицы задают сетку, по которой будет вестись хеширование. В простейшем случае можно использовать равномерную сетку, для которой $t_{ik} - t_{ik-1} \equiv \text{const}$.

Через I обозначим мультииндекс $(i_1, i_2, ..., i_n), 1 \geq i_k \geq m, k = 1, ..., n$. Каждому мультииндексу сопоставим ячейку Q_I построенной выше сетки хеширования

$$Q_I = [t_{1i_1-1}, t_{1i_1}] \times [t_{2i_2-1}, t_{2i_2}] \times ... \times [t_{ni_n-1}, t_{ni_n}].$$

Хеширование заключается в построении многомерной матрицы H размерности $m \times ... \times m$ (здесь n множителей), элементами которой являются наборы натуральных чисел $h_I = (h_I^1, ..., h_I^{n_I})$, $1 \geq h_I^k \leq M$ такие, что

$$s(x_{h_I^k}) \cap Q_I \neq \emptyset.$$

Другими словами, эти натуральные числа задают номера элементов множества X, чьи образы $s(x)$ пересекают ячейку Q_I.

2. Алгоритм геометрического хеширования

Изложим для начала последовательный алгоритм хеширования. На входе алгоритм получает список объектов $x_1, ..., x_M$ элементов множества X. На выходе алгоритм должен сформировать многомерную матрицу H, содержащую списки требуемых номеров объектов $x_i, i = 1, ..., M$. Основной цикл перебирает весь список $x_1, ..., x_M$ и на каждом шаге осуществляет следующую последовательность действий:

- Вычисляет значение отображения $s(x_i)$.

- Находит ячейки Q_I, которые пересекают множество $s(x_i)$. Результатом работы алгоритма на этом этапе является набор мультииндексов $(I_1^i, ..., I_{n_i}^i)$ таких, что

$$Q_{I_{n_i}^i} \cap s(x_i) \neq \emptyset.$$

- В хеш-матрицу H добавляется номер i в списки $H[I]$ для каждого мультииндекса $I = I_1^i, ..., I_{n_i}^i$.

В применении к задаче локализации точки в качестве X выбирается множество треугольников, а $s(T_k)$ можно выбирать либо сам треугольник $T_k \in T$, либо ограничивающий его прямоугольник.

3. Параллельный алгоритм и его реализация

Распараллеливание будем осуществлять по входному множеству X, то есть каждому параллельно работающему исполнителю будет назначаться свое подмножество $X_j \subset X, j = 1, ..., q, X = \cup_j X_j, X_i \cap X_j = \emptyset, i \neq j$. Основной исполнитель будет формировать матрицу H, получая от вспомогательных исполнителей соответствующие списки наборов мультииндексов. Обозначим через p число исполнителей. В общем случае $p \neq q$.

Для реализации параллельных вычислений матрицы H мы воспользуемся шаблоном «пул потоков/процессов» (Thread/Process Pool) проектирования параллельных программ [1; 10; 16]. Схематично шаблон представлен на рисунке 2 на примере пула потоков. На этой диаграмме класс Task содержит порцию данных (объект класса Data) в виде отдельного задания. Все задания собраны в классе TaskQueue в виде очереди, защищенной от совместного доступа мютексом — объектом класса Mutex. Класс ThreadPool содержит список потоков threads объектов класса Thread и ссылку на очередь заданий task_queue. Метод ProcessTasks этого класса запускает на выполнение все потоки пула threads, передавая им ссылку tasks_queue. Потоки, выполняя метод run(), выбирают из очереди заданий еще не выполненное задание и обрабатывают соответствующие ему данные. Пул потоков работает до тех пор, пока остаются невыполненные задания, то есть пока очередь tasks_queue не станет пустой (см. рис. 3).

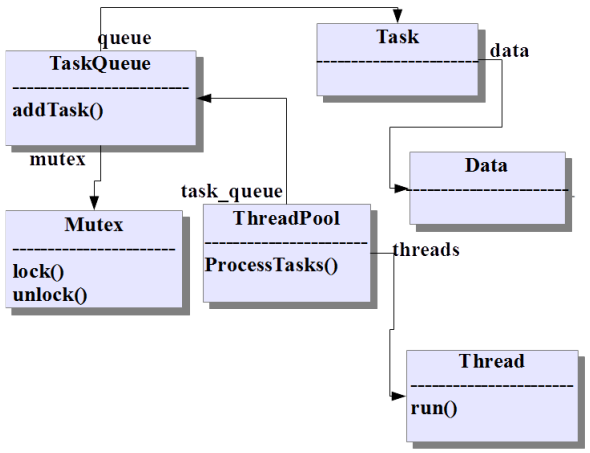

Рис. 2. UML-диаграмма пула потоков

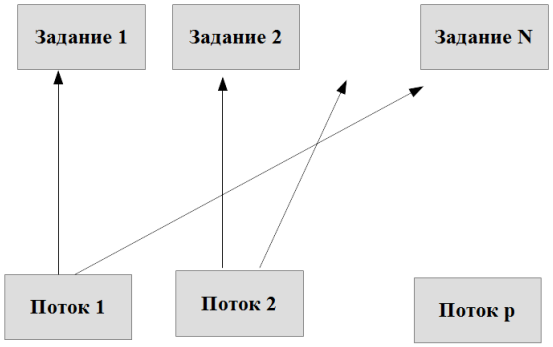

Рис. 3. Распределение заданий по потокам

Для реализации алгоритма геометрического хеширования нами использовался язык программирования Python версии 3.4. Этот выбор можно объяснить следующими обстоятельствами:

- Python широко используется для решения вычислительных задач [5–7].
- Для языка Python имеется пакет NumPy, который реализует итераторы многомерных массивов с помощью соответствующих мультииндексов [8].
- В этом же пакете реализованы Си-подобные массивы вместе с различными операциями линейной алгебры [2].
- В версии Python 3.2 имеется модуль concurrent.futures, в котором есть два класса ThreadPoolExecutor и ProcessPoolExecutor — необходимые нам пулы потоков и процессов [11].

С другой стороны, реализация схемы на рисунке 2 при использовании языка программирования Python сопряжено с определенными трудностями. Дело в том, что интерпретатор Python (точнее его Си-реализация — CPython [11]) является однопоточным и для защиты собственных данных использует глобальную блокировку GIL (Global Interpreter Lock). GIL представляет собой объект синхронизации, который захватывается потоками (потоки в Python — это обычные POSIX потоки) в соответствии с собственной политикой интерпретатора. Такая однопоточность интерпретатора приводит к тому, что потоки параллельно не могут выполняться на разных процессорах или ядрах процессора. А, значит, использование пула потоков становится бесполезным. Один из способов обойти это ограничение в контексте схемы (рис. 2) — это вместо пула потоков использовать пул процессов. Это, конечно, более накладно, но позволит выполнять разные процессы на нескольких процессорах.

На рисунке 4 приведена объектно-ориентированная модель решения задачи на основе класса ProcessPoolExecutor. Все решение помещено в пакет GeomHashing. В классе Job метод work_do() является шаблонным. Его реализация абстрагируется от реализации методов get_job() и alone_job(). Эти методы переопределяются в подклассах в зависимости от конкретной задачи распараллеливания. В нашем случае мы реализуем

подкласс GeomHash. Метод get_job() должен вернуть информацию об очередном задании для процесса, а метод alone_job() обрабатывает порцию данных этого задания в контексте соответствующего процесса из пула.

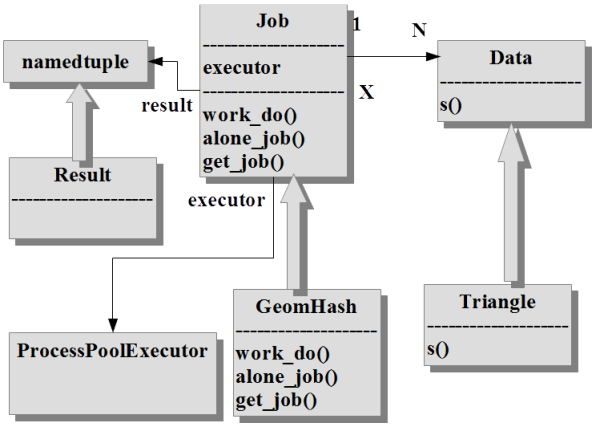

Рис. 4. UML-диаграмма классов пакета GeomHashing

Использование пула процессов сопряжено еще с одной трудностью, которую мы оценим количественно. Трудность эта связана с необходимостью осуществлять передачу данных между процессами. Для потоков данной проблемы не возникает из-за использования общего адресного пространства. Пусть имеется $mp, m \geq 1, p \geq 1$ единиц данных, требующих определенной обработки пулом из p процессов, которые будут выполняться на p процессорах. Обозначим через t_1 время передачи единицы данных между процессами, а через t_2 — время обработки единицы данных одним процессором. Для последовательной обработки данных потребуется время

$$T_{\text{посл}} = mpt_2.$$

Для параллельной обработки, с использованием схемы пула процессов, понадобится время

$$T_{\text{парал}} = mpt_1 + mt_2.$$

Таким образом, выигрыш во времени $T_{\text{парал}} < T_{\text{посл}}$ будет иметь место, если

$$t_2 > \frac{p}{p-1} t_1.$$

Другими словами, параллельная реализация на основе пула процессов имеет смысл, если время обработки единицы данных в $p/(p-1)$ раз больше времени пересылки единицы данных при межпроцессном взаимодействии. При этом ускорение получается равным

$$\frac{T_{\text{посл}}}{T_{\text{парал}}} = \frac{p\tau}{p+\tau},$$

где $\tau = t_2/t_1$. Эту формулу можно интерпретировать как своеобразный аналог закона Амдала [3], учитывающий не соотношение между последовательной и параллельной частями программы, а учитывающий время на пересылку данных.

В соответствии со сделанными замечаниями внесем изменения в нашу схему. Для этого заменим отношение композиции «один ко многим» на отношение инстанцирования между классами Job и Data (пунктирная линия на рисунке 5). Это означает, что вместо передачи процессам самих данных будет передаваться необходимая информация о них. Добавленный в класс Job «фабричный» метод make_data() по переданной информации создает данные для обработки.

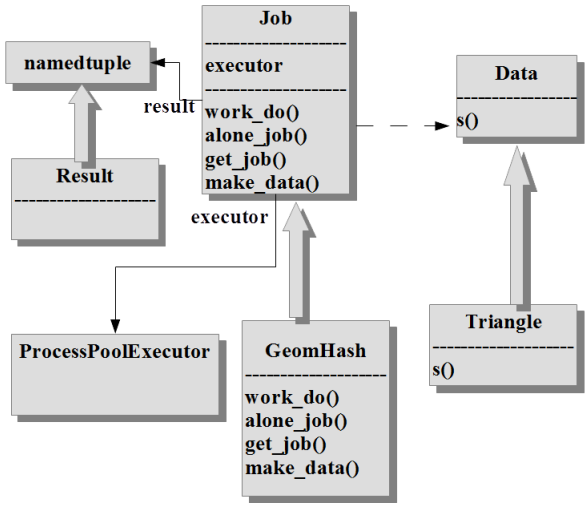

Рис. 5. Окончательная UML-диаграмма классов пакета GeomHashing

На рисунке 6 представлены результаты тестирования последовательной и параллельной программы, решающих задачу геометрического хеширования на равномерной сетке 10×10. По горизонтальной оси отложены значения мощности входного множества X, а по вертикальной оси — указано время в секундах. Тестирование проводилось на машине с процессором Intel Pentium с 4-мя ядрами по 2,16 ГГц.

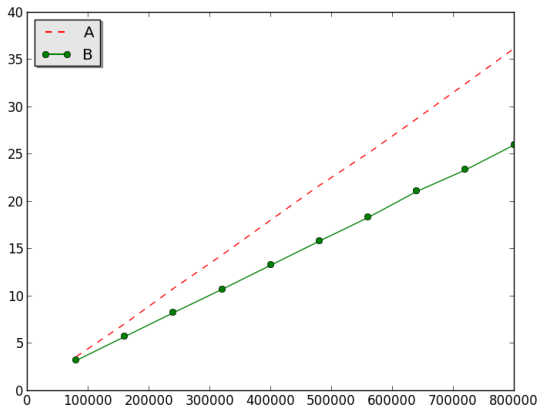

Рис. 6. Время хеширования: A — последовательный алгоритм; B — параллельный алгоритм

ПРИМЕЧАНИЕ

[1] Работа выполнена при финансовой поддержке РФФИ (проект № 15-41-02517 р_поволжье_а).

СПИСОК ЛИТЕРАТУРЫ

1. Академия Microsoft: Параллельные вычисления и многопоточное программирование. Лекция 7: Пул потоков и библиотека параллельных задач. — Электрон. текстовые дан. — Режим доступа: http://www.intuit.ru/studies/courses/10554/1092/lecture/21509. — Загл. с экрана.

2. Бабищевич, П. Н. Численные методы: Вычислительный практикум / П. Н. Бабищевич. — М. : ЛЕНАНД, 2015. — 320 с.

3. Гергель, В. П. Теория и практика параллельных вычислений / В. П. Гергель. — М. : Интернет-Университет Информационных технологий ; БИНОМ. Лаборатория знаний, 2007. — 423 с.

4. Гриценко, Ю. Б. Решение проблемы обновления пространственных данных в среде Oracle Spatial / Ю. Б. Гриценко, В. Ю. Вишняков, С. С. Ощепков // Докл. ТУСУРа. Управление, вычислительная техника и информатика. — 2008. — № 2 (18). — С. 76–79.

5. Клячин, В. А. Оптимизация построения расчетной сетки для решения задачи локального криовоздействия с использованием многомерного геометрического хеширования на основе пакета NumPy / В. А. Клячин // Изв. Сарат. ун-та. Новая серия. Серия Математика. Механика. Информатика. — 2014. — Т. 14, № 3. — С. 355–362.

6. Математический Python. — Электрон. текстовые дан. — Режим доступа: http://jenyay.net/Programming/PyMath. — Загл. с экрана.

7. Никольский, Д. Н. Разработка программного обеспечения для численного решения задач эволюции границы раздела различных жидкостей в пористых средах сложной геологической структуры с использованием пакета NumPy / Д. Н. Никольский // Ученые записки Орловского государственного университета. Серия: Естественные, технические и медицинские науки. — 2012. — № 6-1. — С. 42–47.

8. Олифант, Т. Многомерные итераторы NumPy / Т. Олифант // Идеальный код. — СПб. : Питер, 2011. — С. 341–358.

9. Препарата, Ф. Вычислительная геометрия / Ф. Препарата, М. Шеймос. — М. : Наука, 1989. — 478 с.

10. Пул управляемых потоков. MSDN — Microsoft. — Электрон. текстовые дан. — Режим доступа: https://msdn.microsoft.com/ru-ru/library/0ka9477y(v=vs.110).aspx. — Загл. с экрана.

11. Саммерфилд, М. Python на практике / М. Саммерфилд. — М. : ДМК Пресс, 2014. — 338 с.

12. Скворцов, А. В. Алгоритмы построения и анализа триангуляции / А. В. Скворцов, Н. С. Мирза. — Томск : Изд-во Том. ун-та, 2006. — 168 с.

13. Шикин, Е. В. Компьютерная графика. Полигональные модели / Е. В. Шикин, А. В. Боресков. — М. : ДИАЛОГ-МИФИ, 2001. — 464 с.

14. An Improved Method of Geometric Hashing Pattern Recognition / M. Ling, L. Yumin, J. Huiqin, W. Zhongyong, Z. Haofei // I. J. Modern Education and Computer Science. — 2011. — Vol. 3. — P. 1–7.

15. Mian, A. S. Three-dimensional model-based object recognition and segmentation in cluttered scenes / A. S. Mian, M. Bennamoun, R. Owens // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2006. — Vol. 28. — P. 1584–1601.

16. Thread pool pattern. — Electronic text data. — Mode of access: https://en.wikipedia.org/wiki/Thread_pool_pattern. — Title from screen.

17. Wolfson, H. J. Geometric Hashing: An Overview / H. J. Wolfson, I. Rigoutsos // IEEE Computational Science and Engineering. — 1997. — Vol. 4, № 4. — P. 10–21.

REFERENCES

1. *Akademiya Microsoft: Parallelnye vychisleniya i mnogopotochnoe programmirovanie. Lektsiya 7: Pul potokov i biblioteka parallelnykh zadach* [Microsoft Academy: Parallel computing and multi-thread programming. Lecture 7: Thread Pool and Parallel Tasks Library]. Available at: http://www.intuit.ru/studies/courses/10554/1092/lecture/21509.

2. Babishchevich P.N. *Chislennye metody: Vychislitelnyy praktikum* [Computational methods in practice]. Moscow, LENAND Publ., 2015. 320 p.

3. Gergel V.P. *Teoriya i praktika parallelnykh vychisleniy* [Theory and practice of parallel calculations]. Moscow, Internet-Universitet Informatsionnykh tekhnologiy ; BINOM. Laboratoriya znaniy Publ., 2007. 423 p.

4. Gritsenko Yu.B., Vishnyakov V.Yu., Oshchepkov S.S. Reshenie problemy obnovleniya prostranstvennykh dannykh v srede Oracle Spatial [Addressing update spatial data in Oracle Spatial]. *Dokl. TUSURa. Upravlenie, vychislitelnaya tekhnika i informatika*, 2008, no. 2 (18), pp. 76-79.

5. Klyachin V.A. Optimizatsiya postroeniya raschetnoy setki dlya resheniya zadachi lokalnogo kriovozdeystviya s ispolzovaniem mnogomernogo geometricheskogo kheshirovaniya na osnove paketa NumPy [Optimization of Calculus Mesh for Cryobiology Problem Based on Multidimensional Hashing Using NumPy]. *Izv. Sarat. un-ta. Novaya seriya. Seriya Matematika. Mekhanika. Informatika*, 2014, vol. 14, no. 3, pp. 355-362.

6. *Matematicheskiy Python* [Mathematical Python]. Available at: http://jenyay.net/Programming/PyMath.

7. Nikolskiy D.N. Razrabotka programmnogo obespecheniya dlya chislennogo resheniya zadach evolyutsii granitsy razdela razlichnykh zhidkostey v poristykh sredakh slozhnoy geologicheskoy struktury s ispolzovaniem paketa NumPy [Development of software for the numerical solution of the evolution of the interface between different fluids in porous media complex geological structure using the package NumPy]. *Uchenye zapiski Orlovskogo gosudarstvennogo universiteta. Seriya: Estestvennye, tekhnicheskie i meditsinskie nauki* [Scientific notes of Oryol State University. Series: natural, technical and medical sciences], 2012, no. 6-1, pp. 42-47.

8. Olifant T. Mnogomernye iteratory NumPy [Multidimensional iterators NumPy]. *Idealnyy kod* [Beautiful Code]. Petersburg, Piter Publ., 2011, pp. 341-358.

9. Preparata F., Sheymos M. *Vychislitelnaya geometriya* [Computational geometry]. Moscow, Nauka Publ., 1989. 478 p.

10. *Pul upravlyaemykh potokov. MSDN — Microsoft* [Pool of managed threads. MSDN — Microsoft]. Available at: https://msdn.microsoft.com/ru-ru/library/0ka9477y(v=vs.110).aspx.

11. Sammerfild M. *Python na praktike* [Python in practice]. Moscow, DMK Press Publ., 2014. 338 p.

12. Skvortsov A.V., Mirza N.S. *Algoritmy postroeniya i analiza triangulyatsii* [Analisys and algorithms of triangulations]. Tomsk, Izd-vo Tom. un-ta Publ., 2006. 168 p.

13. Shikin E.V., Boreskov A.V. *Kompyuternaya grafika. Poligonalnye modeli* [Computrer graphics. Polygon models]. Moscow, DIALOG-MIFI Publ., 2001. 464 p.

14. Ling M., Yumin L., Huiqin J., Zhongyong W., Haofei Z. An Improved Method of Geometric Hashing Pattern Recognition. *I. J. Modern Education and Computer Science*, 2011, vol. 3, pp. 1-7.

15. Mian A.S., Bennamoun M., Owens R. Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006, vol. 28, pp. 1584-1601.

16. *Thread pool pattern*. Available at: https://en.wikipedia.org/wiki/Thread_pool_pattern.

17. Wolfson H.J., Rigoutsos I. Geometric Hashing: An Overview. *IEEE Computational Science and Engineering*, 1997, vol. 4, no. 4, pp. 10-21.

PARALLEL ALGORITHM OF GEOMETRICAL HASHING BASED ON NUMPY PACKAGE AND PROCESSES POOL

Vladimir Aleksandrovich Klyachin

Doctor of Physical and Mathematical Sciences,
Head of Department of Computer Science and Experimental Mathematics,
Volgograd State University
klchnv@mail.ru, kiem@volsu.ru
Prosp. Universitetsky, 100, 400062 Volgograd, Russian Federation

Abstract. The article considers the problem of multi-dimensional geometric hashing. The paper describes a mathematical model of geometric hashing and considers an example of its use in localization problems for the point. A method of constructing the corresponding hash matrix by parallel algorithm is considered. In this paper an algorithm of parallel geometric hashing using a development pattern «pool processes» is proposed. The implementation of the algorithm is executed using the Python programming language and NumPy package for manipulating multidimensional data. To implement the process pool it is proposed to use a class Process Pool Executor imported from module concurrent.futures, which is included in the distribution of the interpreter Python since version 3.2. All the solutions are presented in the paper by corresponding UML class diagrams. Designed GeomNash package includes classes Data, Result, GeomHash, Job.

The results of the developed program presents the corresponding graphs. Also, the article presents the theoretical justification for the application process pool for the implementation of parallel algorithms. It is obtained condition

$$t_2 > \frac{p}{p-1} t_1$$

of the appropriateness of process pool. Here t_1 — the time of transmission unit of data between processes, and t_2 — the time of processing unit data by one processor.

Key words: hashing, process pool, package NumPy, computational geometry, parallel algorithm.