ГИАБ. Горный информационно-аналитический бюллетень /
MIAB. Mining Informational and Analytical Bulletin, 2026;(9):150-161
ОРИГИНАЛЬНАЯ СТАТЬЯ / ORIGINAL PAPER

УДК 622.75 + 004.932.2 + 004.021 DOI: 10.25018/0236_1493_2026_9_0_150

ВНЕДРЕНИЕ КОМПИЛИРУЕМОГО МОДУЛЯ НА C++ ДЛЯ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ РАСПОЗНАВАНИЯ ПАРАМЕТРОВ ПЕННОЙ ФЛОТАЦИИ

А.В. Затонский[1,2], С.В. Кузнецов[2], П.В. Плехов[1]

[1] Пермский национальный исследовательский политехнический университет, Березниковский филиал, Березники, Россия, e-mail: zxenon@narod.ru
[2] Пермский национальный исследовательский политехнический университет, Пермь, Россия

Аннотация: Процесс пенной флотации калийной руды является основным методом разделения продукта и примесей. Управление флотацией производится вручную оператором, обслуживающим целое отделение флотации. Это приводит к запаздыванию в принятии решений и ошибкам в изменении подачи флотационных реагентов и воздуха. Одним из способов исключения недостатков, связанных с человеческим фактором, является применение систем технического зрения. Однако подстройка существующего программного обеспечения под конкретные параметры съемки занимает длительное время, что снижает точность распознавания параметров пенного слоя. Предлагается ускорить распознавание за счет использования многопоточности для определения оптимальных порогов бинаризации бликов, антибликов и коэффициента повышения контрастности изображения. Поскольку в языке программирования Python использовать многопоточность невозможно, написана специальная динамическая библиотека на C++, реализующая многопоточный поиск параметров. Достигнуто уменьшение времени поиска параметров на 87,4%, что позволяет определять их каждый полуоборот пеногона, а не один раз для длительной серии кадров. За счет этого на 8,97% увеличено количество правильно распознаваемых пузырьков, то есть качество определения параметров пенного слоя калийной флотомашины. При использовании библиотеки в системах контроля технологического процесса это приведет к уменьшению числа ложных срабатываний сигнализации или к более точной дозировке реагентов, подаваемых в флотомашину.

Ключевые слова: калийная флотация, компьютерное зрение, управление, техническое зрение, многопоточность.

Для цитирования: *Затонский А. В., Кузнецов С. В., Плехов П. В.* Внедрение компилируемого модуля на C++ для повышения эффективности распознавания параметров пенной флотации // Горный информационно-аналитический бюллетень. – 2026. – № 9. – С. 150–161. DOI: 10.25018/0236_1493_2026_9_0_150.

Introduction of compilation module in C++ to enhance efficiency of identification of froth flotation parameters

A.V. Zatonskiy[1,2], S.V. Kuznetsov[2], P.V. Plekhov[1]

[1] Perm National Research Polytechnic University, Berezniki Branch, Berezniki, Russia, e-mail: zxenon@narod.ru
[2] Perm National Research Polytechnic University, Perm, Russia

Abstract: Froth flotation of potash is a main method of separation of the product and impurities. The process is controlled manually by an operator who serves the whole flotation floor. This can lead to retarded decision-making and to inaccurate adjustment of feed of flotation reagents and air. One of the ways of avoiding shortages related with human factor is employment of computer vision systems. However, software tailoring to fit specific workflow parameter takes a long time, which impairs accuracy of identification of froth bed parameters. It is proposed to accelerate the identification process through multithreading in determination of the optimal thresholds for glare and anti-glare binarization and the image contrast improvement factor. Since multithreading is impossible in Python, a dedicated dynamic library is created in C++ to implement the multithreaded search of parameters. The time of the search is shortened by 87.4%, which allows identification of parameters in each half-turn of the frother rather than once in a long series of images. Thank to this, the number of the correctly identified bubbles is increased by 8.97%, i.e. the quality of identification of the froth bed parameters in a potash flotation machine is improved. The use of the library in the process flowsheet control can reduce the number of nuisance alarms or enhance accuracy of metering feed of reagents to flotation machine.

Key words: potash flotation, computer vision, control, machine vision, multithreading.

For citation: Zatonskiy A. V., Kuznetsov S. V., Plekhov P. V. Introduction of compilation module in C++ to enhance efficiency of identification of froth flotation parameters. *MIAB. Mining Inf. Anal. Bull.* 2026;(9):150-161. [In Russ]. DOI: 10.25018/0236_1493_2026_9_0_150.

Введение

Калийные удобрения являются важным средством повышения урожайности сельскохозяйственных продуктов. Основным регионом их производства в РФ является Верхнекамское месторождение, расположенное в Пермском крае. Добываемая руда содержит полезный продукт (сильвинит, KCl) и примеси (галит NaCl, нерастворимый остаток и другие).

Современные технологии не позволяют осуществлять селективную экстракцию хлорида калия, поэтому добытая руда отправляется на последующую переработку с обогащением [1]. Основным методом обогащения калийных руд является пенная флотация. Она основана на избирательном разделении частиц за счет их различной смачиваемости, так как гидрофобные минералы прилипают к пузырькам воздуха и поднимаются в пенный слой [2]. Эффективность процесса определяется состоянием пены, в частности, формой и размером пузырьков, а также цветом пены. Форма и размер пузырьков влияют на стабильность пены и эффективность захвата частиц, а изменение цвета сигнализирует о попадании нежелательных минералов [3]. Эти визуальные параметры предоставляют оператору важную информацию для управления процессом и обеспечения качества конечной продукции.

Ручной мониторинг флотационной пены неэффективен из-за человеческого фактора: оператор флотационной машины не может одновременно отслеживать пены во всех машинах отделения. Внедрение систем машинного зрения позволяет оперативно привлекать оператора к принятию решений только при возникновении особых ситуаций, что обеспечивает повышение качества продукции и производительности [4].

Для распознавания параметров калийной флотационной пены ранее было разработано специализированное десктопное приложение на языке програм-

мирования Python. Оно обрабатывает видеопоток с камеры, установленной над флотационной машиной, и в режиме реального времени выводит ключевые параметры пенного слоя, а именно количество распознанных пузырьков в кадре, количество красной компоненты, а также среднее медианное и среднее арифметическое расстояние между центрами пузырьков [5].

Алгоритм подсчета пузырьков основан на распознавании не контуров пузырьков, а бликов, формируемых от точечного источника освещения. Реализация алгоритма включает в себя предварительную бинаризацию обрабатываемой области с последующим подсчетом бликов. В работе [6] для повышения полноты распознавания было внедрено распознавание «антибликов» — недосвеченных пузырьков, которые не бликуют из-за неперпендикулярной ориентации к точечному источнику освещения. Улучшенный алгоритм позволил учитывать такие пузырьки за счет применения инверсной бинаризации. Дальнейшее развитие в работе [7] добавило этап предварительной коррекции контраста, что повысило качество распознавания параметров пенного слоя. Однако такие «доработки» привели к более, чем двухкратному увеличению времени на подбор коэффициентов бинаризации, контраста и антиблика, при которых распознается наибольшее количество пузырьков.

Исходя из специфики задачи обработки кадров в режиме реального времени и продолжительности процесса, подбор «оптимальных» коэффициентов производился периодически по одному разу, после чего к каждому кадру применялись ранее сохраненные настройки. Они не могли определяться многократно из-за нарастания задержки обработанного видеопотока, это привело бы к снижению качества обработки изображения, так как освещенность от кадра к кадру может колебаться, что, хоть и незначительно, снижает количество распознанных пузырьков.

Целью данной работы является решение вышеописанной проблемы за счет модификации алгоритма таким образом, чтобы он успевал рассчитывать оптимальные коэффициенты бинаризации каждого кадра. Это приведет к улучшению системы автоматизированного регулирования в целом.

Достижение цели возможно за счет использования многопоточности. Имеется в виду такой подход к параллельным вычислениям, при котором один процесс может выполняться в нескольких параллельных потоках. Каждый поток является независимым потоком выполнения инструкций, и может быть назначен для решения отдельной подзадачи, например, для вычисления такого порога бинаризации, при котором распознается наибольшее количество пузырьков. Все потоки выполняются параллельно в общем адресном пространстве процесса, что значительно сокращает время выполнения вычислительной задачи за счет распределения нагрузки между несколькими ядрами процессора [8 — 10].

Однако применение многопоточности на Python затруднено из-за GIL (Global Interpreter Lock), который, по сути, запрещает выполнение нескольких потоков одновременно. Это вызывает трудности для реализации параллельных вычислений на Python. Хоть последние версии Python и поддерживают экспериментальную возможность отключения GIL, ее практическое применение невозможно, так как сторонние библиотеки, которые используются в существующем ПО, не поддерживают отключенный GIL [11 — 13].

Поэтому предлагается использовать динамически подключаемые библиоте-

ки (DLL), которые можно собрать на компилируемом языке программирования, не подверженном GIL. Таким образом, задача может быть решена путем выноса части программы со сложными вычислениями в отдельный модуль, поддерживающий многопоточность. Такой подход позволит увеличить количество распознаваемых параметров пены по сравнению с предыдущим алгоритмом за счет подбора оптимальных коэффициентов бинаризации не на период, а при обработке каждого кадра.

Языки программирования

Прежний алгоритм перебирал коэффициенты и преобразовывал кадры по очереди. Это ограничение связано с тем, как устроен Python: в нем тяжело писать вычислительные циклы без потери скорости. Библиотеку для Python нельзя сделать интерпретируемой — ее нужно заранее скомпилировать в машинный код. Тогда система загрузит готовый модуль, и программа будет работать с ним напрямую.

Интерпретируемые языки, например, Python, для решения задачи не подходят. Их исходный код не компилируется в машинный до старта, а выполняется построчно через интерпретатор. Процессор такой код напрямую не видит, а прослойка в виде интерпретатора неизбежно снижает скорость [14]. Таким образом, для решения задачи требуется использовать компилируемый язык.

Существующее приложение уже использует OpenCV. Согласно документации, OpenCV напрямую доступна для языков программирования C++, Java и JavaScript. JavaScript — интерпретируемый язык, поэтому он не подходит. Остаются C++ и Java.

Языки являются компилируемыми, и дают выигрыш в скорости перед Python, но между ними есть разница. C++ транслируется в нативный код без прослоек. Для обработки видео в реальном времени, где важна каждая миллисекунда, это дает максимальную производительность. Кроме того, OpenCV изначально написана под C++, поэтому на нем доступны все ее возможности. Java работает иначе. Код превращается в байт-код для виртуальной машины JVM (Java Virtual Machine). У JVM есть Just-In-Time-компилятор: он переводит требовательные участки в машинный код уже по ходу работы. Но даже при этом Java уступает C++ в производительности [15, 16]. Поэтому для сборки библиотеки выбран C++.

Модификация алгоритма

Исходный алгоритм работал путем полного перебора в заданных пределах двух коэффициентов — порога бинаризации для блика и антиблика. С его помощью подбирались такие значения, при которых распознавалось наибольшее количество пузырьков.

Сначала выполнялся поиск оптимального порога бинаризации. Для каждого значения порога программа обрабатывала изображение и подсчитывала число распознанных пузырьков. Все результаты сохранялись в словарь, где ключом служил порог бинаризации, а значением — количество пузырьков. Затем выбирался порог, дававший максимальное значение.

Далее проводилась коррекция контрастности с помощью алгоритма CLAHE. После этого порог бинаризации уточняли, поскольку в работе [7] было показано, что изменение контраста смещает оптимальный порог. Затем производился поиск порога бинаризация для нахождения антибликов. Порог бинаризации перебирался от 80 до 244, контраст — от 1 до 9, антиблик — от 0 до 119. Эти пределы установлены на основе экспериментов и имитационного моделирования. Вне этих диапазонов на

изображениях с предприятий ПАО «Уралкалий» и ООО «Еврохим–Усольский калийный комбинат» пузырьки вообще не распознавались. К тому же с учетом скорости старого алгоритма такое ограничение сокращало время расчетов в Python без потери качества распознавания.

Новый алгоритм сохраняет логику поиска, но меняет способ перебора. Вместо последовательной проверки значений диапазон делится на примерно равные части. Количество частей равно числу возможных потоков [17]. Каждый поток обрабатывает свою часть так же, как в исходном алгоритме, и записывает результаты в общий словарь (коэффициент и значения). За счет параллельной работы поиск ускоряется.

При многопоточности возникает одна проблема — доступ к общим ресурсам. Если несколько потоков одновременно пишут результаты в один словарь, может случиться состояние гонки. Из-за этого данные портятся или результаты становятся неверными. Поэтому в алгоритм добавили мьютексы — механизмы взаимного исключения. Работают они следующим образом: поток блокирует файл записи для других, пока сам пишет данные [18]. После окончания записи поток разблокирует файл, а следующий поток получает доступ. Тогда данные остаются целыми, и алгоритм устойчиво работает в параллельном режиме.

Еще была учтена разница между Python и C++. В Python память чистит сборщик мусора, тогда как в C++ в программном коде должно быть явное освобождение памяти. Поэтому были написаны механизмы ручного освобождения памяти, чтобы не было утечек при долгой работе системы [19]. Для промышленной флотации это важно, потому что обработка идет непрерывно, и очень длительное время (недели, месяцы). Без контроля ресурсов программа со временем начнет работать нестабильно.

Интеграция и сборка библиотеки

Есть несколько способов подключать C++ код к Python. Python через библиотеки ctypes и CFFI умеет напрямую загружать и компилировать C код. Но для C++ кода это потребовало бы писать дополнительную обертку для работы со структурами C. Такое решение усложняет реализацию и будущие доработки модуля. Однако фреймворк pybind11 сам генерирует привязки между языками и не усложняет существующий код, поэтому он был выбран для работы. Кроме того, он проще в использовании, чем другие аналогичные фреймворки [20].

Для сборки библиотеки рассматривались различные инструменты. Сначала предполагалось использовать g++, но в Windows он требует дополнительной настройки — WSL или MinGW. Для поставленной задачи это избыточно [21]. Основной проект на Python писали и запускали только в Windows, а переносить всю сборку и тестирование на Linux нецелесообразно. Рассматривался генератор от Microsoft Visual Studio. Он полностью работает в Windows, но требует ручной настройки параметров сборки, путей зависимостей и компоновки. Также рассматривалась CMake — кроссплатформенная система автоматизации сборки. CMake использует единый файл конфигурации, с которым сборка проходит проще, и ее легче сопровождать. Система сама находит зависимости и определяет пути к библиотекам. Ручная настройка и связанные с ней ошибки исключаются довольно часто [22]. В итоге был выбран CMake из-за простоты сборки.

Результаты

Для проверки работоспособности модифицированного алгоритма был про-

веден сравнительный анализ с предыдущим алгоритмом. Для тестирования было взято изображение с пенной флотацией, представленное на рис. 1.

На данном изображении была протестирована скорость автоматического подбора оптимальных коэффициентов бинаризации, контраста и антиблика предыдущего алгоритма, реализованного на Python, с модифицированным алгоритмом, написанным на C++ и применяющим многопоточность. Результаты сравнения представлены в табл. 1.

Для более наглядного представления зависимости времени выполнения от количества потоков был построен график (рис. 2).

Вариант решения на C++ демонстрирует более высокую производительность по сравнению с Python-версией даже в однопоточном режиме, автоподбор выполняется быстрее на 41,4%. Использование многопоточности позволяет достичь дополнительного ускорения.

Стоит отметить, что наибольший прирост производительности (примерно в пять раз по сравнению с исходной реализацией на Python) достигается при увеличении числа потоков с 1 до 4 — в этом случае ускорение составляет 79,1%. При дальнейшем увеличении числа количества потоков прирост становится значительно менее выраженным: для 8 потоков ускорение составляет 85,3%, а для 16 потоков — 86,9%

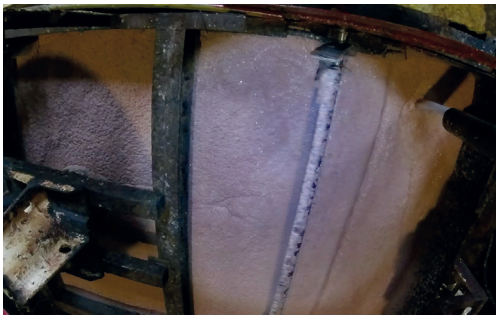

Рис. 1. Изображение флотомашины, взятое для тестирования модифицированного алгоритма
Fig. 1. An image of a flotation machine taken to test the modified algorithm

Таблица 1

Результаты сравнения скорости выполнения автоподбора
Results of autosearch speed compare

Реализация	Количество потоков	Затраченное время, с	Ускорение по сравнению с исходным алгоритмом, %
Python	1	1,91	—
C++	1	1,12	41,4
	2	0,67	64,9
	4	0,4	79,1
	8	0,28	85,3
	16	0,25	86,9

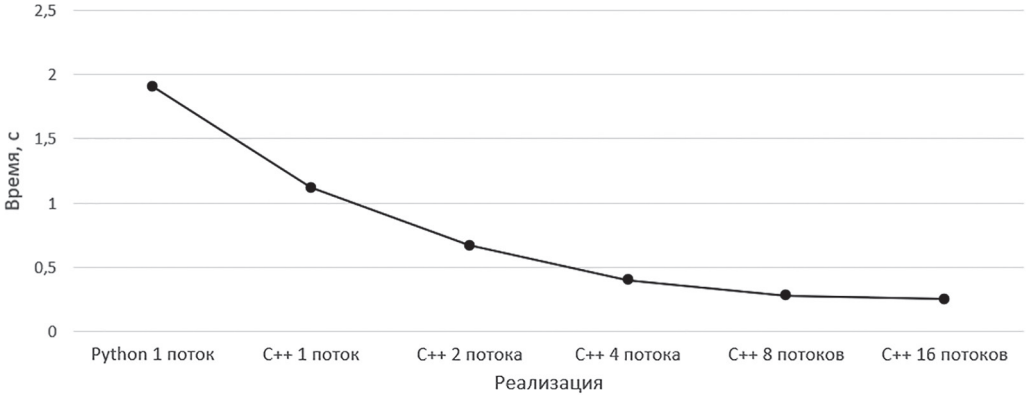

Рис. 2. График сравнения скорости выполнения автоматического подбора в зависимости от реализации
Fig. 2. A graph comparing the speed of automatic selection, depending on the implementation

(в восемь раз). Таким образом, при текущей задаче дальнейшее увеличение числа потоков практически не влияет на итоговую производительность.

Сокращение времени обработки с 1,91 до 0,25 с меняет работу алгоритма. Если раньше подбор выполнялся один раз из-за «долгой» обработки, то теперь становится возможным выполнять его для каждого кадра видеопотока. Это позволяет перейти от использования фиксированных параметров к их динамическому определению, что повышает точность оценки характеристик пенного слоя.

Дополнительная проверка была проведена на видеопотоках, полученных с различных флотационных машин предприятий ПАО «Уралкалий» и ООО «Еврохим — Усольский калийный комбинат» (г. Березники Пермского края). В 7 экспериментах сравнивались время автоподбора коэффициентов для исходного алгоритма и модифицированной версии, реализованной на C++ с использованием 16 потоков. Результаты представлены в табл. 2.

Как видно из таблицы, использование нового алгоритма позволило увеличить скорость подбора оптимальных коэффициентов в среднем на 87,4%, среднее время на обработку составило 0,13 с. Полученная скорость обработки является эффективной для стандартных условий промышленного видеомониторинга с ПАО «Уралкалий» и ООО «Еврохим — Усольский калийный комбинат», где применяются камеры с разрешением 720 линий и частотой 30 кадров в секунду. При времени обработки 0,13 с система способна стабильно обрабатывать каждый 4-й кадр видеопотока (30 FPS / (1/0,13 с) ≈ 3,8 FPS), что достаточно для решения задачи распознавания параметров пены, где обрабатывается в среднем каждый 20-й кадр. Таким образом, алгоритм не только демонстрирует рекордное ускорение, но и обеспечивает производительность, которая с запасом покрывает требования программного обеспечения.

После проверки алгоритма на эффективность и адекватность был проведен эксперимент для оценки изменчивости оптимальных коэффициентов при динамическом отборе для каждого 20 «пригодного» кадра видеопотока. То есть кадра, в обрабатываемую область которого не попадает пеногон, чтобы исключить кадры с искаженным распознаванием пузырьков.

В качестве эксперимента использовалась съемка флотомашины, представленной на рис. 1. Получившиеся графи-

Таблица 2

Результаты сравнительного анализа старого и нового алгоритмов
Results of a comparative analysis of the old and new algorithms

№ изображения	Время на обработку старым алгоритмом, с	Время на обработку модифицированным алгоритмом, с	Разница между старым и новым алгоритмами, %
1	0,97	0,13	86,6
2	0,71	0,08	88,7
3	1,01	0,13	87,1
4	1,17	0,16	86,3
5	0,9	0,11	87,7
6	1,07	0,12	88,8
7	1,51	0,2	86,8
среднее	1,04	0,13	87,4

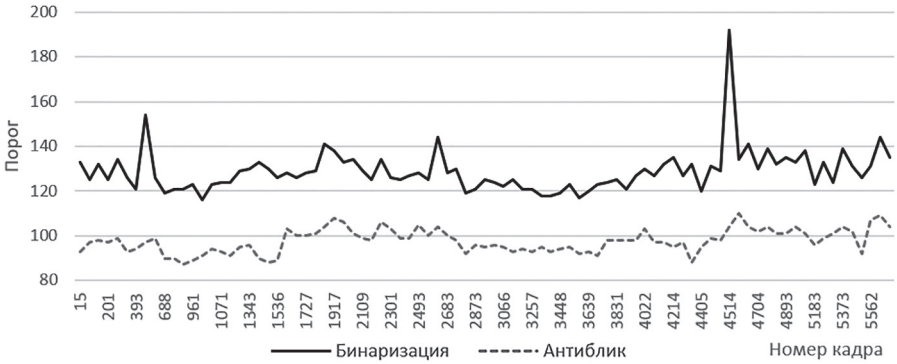

Рис. 3. График изменения порогов бинаризации и антиблика в зависимости от кадра
Fig. 3. Graph of changes in binarization and anti-glare thresholds depending on the frame

ки коэффициентов показаны на рис. 3 и рис. 4.

Анализ графиков подтверждает первоначальное предположение о том, что изменение условий освещенности приводит к колебанию коэффициентов бинаризации, контраста и антиблика. Разброс значений порога бинаризации достигает 29,5% относительно среднего значения, антиблика — 11,7%. Наибольшая изменчивость наблюдалась для коэффициента контраста (диапазон изменений — 114%), что свидетельствует о его сильной изменчивости при меняющихся условиях освещенности. Полученные результаты подтверждают существенное влияние внешних факторов на процесс флотации и обосновывают необходимость применения динамического подбора параметров.

Далее было проведено сравнение количества распознанных пузырьков с использованием динамического и статического подборов. График зависимости количества распознанных пузырьков от номера кадра представлен на рис. 5.

График позволяет наглядно представить преимущества адаптивного выбора параметров в режиме реального времени по сравнению со статической обработкой. Исходя из полученных результатов, количество распознанных пузырьков при использовании нового метода выросло в среднем на 8,97%.

Рис. 4. График изменения коэффициента контраста в зависимости от кадра
Fig. 4. Graph of contrast ratio changes depending on the frame

Рис. 5. График зависимости количества распознанных пузырьков от кадра
Fig. 5. Graph of the dependence of the number of detected bubbles on the frame

Заключение

Таким образом, была произведена модификация существующего алгоритма: ключевые функции были перенесены с Python на динамически подключаемую библиотеку, написанную на C++ и использующую многопоточность при вычислении оптимальных порогов бинаризации, контраста и антиблика. В результате время поиска коэффициентов сократилось в среднем на 87,4%. Это позволяет подбирать пороги бинаризации при обработке каждого кадра, а не серии кадров, то есть улучшить качество распознавания параметров пенного слоя калийной флотомашины.

Эксперимент, проведенный в ходе модификации существующего алгоритма, показал, что использование компилируемого языка в сочетании с многопоточностью позволяет динамически подбирать коэффициенты для каждого кадра и обеспечивает увеличение количества распознаваемых пузырьков в среднем на 8,97%. Следовательно, повысилось качество распознавания пенного слоя флотационной машины, что при практическом применении метода позволит либо избежать ложных срабатываний системы сигнализации состояния машины, либо более точно дозировать подачу реагентов.

Окончательным результатом будут увеличение извлечения из руды хлорида калия и уменьшение его количества в «хвостах» — отходах, отправляемых на открытое складирование или захоронение. Таким образом, разработанная библиотека способствует повышению экономических показателей производства и снижению нагрузки на окружающую среду.

СПИСОК ЛИТЕРАТУРЫ

1. *Земсков А. Н.*, *Лискова М. Ю.*, *Заалишвили В. Б.*, *Шамрин М. Ю.* Современные технологические и технические решения при ведении горных работ на калийных рудниках // Известия Тульского государственного университета. Науки о земле. — 2022. — № 2. — С. 284—296. DOI: 10.46689/2218-5194-2022-2-1-284-296.

2. *Юркевич Н. В.*, *Грошева Т. В.*, *Еделев А. В.*, *Гуреев В. Н.*, *Мазов Н. А.* Современные подходы к обогащению баритовых руд // Записки Горного института. — 2024. — Т. 270. — С. 977—993.

3. *Коннова Н. И.*, *Чебокчинов И. П.* Исследование процесса обогащения пробы руды Буреломного рудопроявления методом флотации // Международный научно-исследовательский журнал. — 2023. — № 12 (138). — С. 74. DOI: 10.23670/IRJ.2023.138.165.

4. *Силаков А. В.*, *Варламова С. А.*, *Котков П. В.* Программное распознавание дефектов изображений регулярных текстур в текстильной промышленности // Известия высших учебных заведений. Технология текстильной промышленности. — 2022. — № 2 (398). — С. 266—272. DOI: 10.47367/0021-3497_2022_2_266.

5. *Варламова С. А.*, *Затонский А. В.*, *Федосеева К. А.* Исследование чувствительности к освещению метода бликового распознавания пен калийных флотационных машин // Обогащение руд. — 2021. — № 6. — С. 29—33. DOI: 10.17580/or.2021.06.05.

6. *Затонский А. В.*, *Варламова С. А.*, *Федосеева К. А.* Улучшение компьютерного распознавания параметров пены калийных флотомашин за счет учета антибликов // Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника. — 2022. — Т. 22. — № 3. — С. 57—67. DOI: 10.14529/ctcr220306.

7. *Затонский А. В.*, *Федосеева К. А.*, *Кузнецов С. В.* Улучшение качества распознавания окатышей полиметаллических руд на изображении за счет адаптивной коррекции контраста // Инновационное приборостроение. — 2024. — Т. 3. — № 3. — С. 48—52. DOI: 10.31799/2949-0693-2024-3-48-52.

8. *Nemirovsky M., Tullsen D.* Multithreading architecture. Springer Nature, 2022. 112 p.

9. *Lee S. H.* Real-time edge computing on multi-processes and multi-threading architectures for deep learning applications // Microprocessors and Microsystems. 2022, vol. 92, article 104554. DOI: 10.1016/j.micpro.2022.104554.

10. *Wei X., Ma L., Zhang H., Liu Y.* Multi-core-, multi-thread-based optimization algorithm for large-scale traveling salesman problem // Alexandria Engineering Journal. 2021, vol. 60, no. 1, pp. 189—197. DOI: 10.1016/j.aej.2020.06.055.

11. *Фешина Е. В.*, *Омельченко Д. А.*, *Гонатаев Р. Г.* Многопоточность и асинхронность в языке программирования Python // Инновации. Наука. Образование. — 2021. — № 28. — С. 988—992.

12. *Arjona A., Finol G., López P. G.* Transparent serverless execution of Python multiprocessing applications // Future Generation Computer Systems. 2023, vol. 140, pp. 436—449. DOI: 10.1016/j.future.2022.10.038.

13. *Ruys W., Lee H., You B., Talati S., Park J., Almgren-Bell J., Yan Y., Fernando M., Erez M., Gligoric M., Burtscher M., Rossbach C. J., Pingali K., Biros G.* Performance characterization of python runtimes for multi-device task parallel programming // International Journal of Parallel Programming. 2025, vol. 53, no. 2, pp. 1—24. DOI: 10.1007/s10766-025-00788-1.

14. *Сычёв М. Д.*, *Трунова К. Д.* Сравнение производительности при выполнении сортировки методом пузырька на языках программирования C++ и Python // Вестник Пензенского государственного университета. — 2024. — № 4 (48). — С. 131—133.

15. *Kaur A., Nayyar R.* A comparative study of static code analysis tools for vulnerability detection in C/C++ and Java source code // Procedia Computer Science. 2020, vol. 171, pp. 2023—2029. DOI: 10.1016/j.procs.2020.04.217.

16. *Naveed M. S.* Comparison of C++ and Java in implementing introductory programming algorithms // Quaid-E-Awam University Research Journal of Engineering, Science & Technology. 2021, vol. 19, no. 1, pp. 95—103.

17. *Oliveira C.* Multithreading / Practical C++20 Financial Programming. Apress, Berkeley, CA. 2021, pp. 449—474. DOI: 10.1007/978-1-4842-6834-6_17.

18. *Valero A., Gran-Tejero R., Suárez-Gracia D., Georgescu E. A., Ezpeleta J., Álvarez P., Muñoz A., Ramos L. M., Ibáñez P.* A learning experience toward the understanding of abstraction-level interactions in parallel applications // Journal of Parallel and Distributed Computing. 2021, vol. 156, pp. 38—52. ISSN 0743-7315. DOI: 10.1016/j.jpdc.2021.05.008.

19. *Monat R., Ouadjaout A., Miné A.* A multilanguage static analysis of Python programs with native C extensions // International Static Analysis Symposium. Cham: Springer International Publishing, 2021, pp. 323—345. DOI: 10.1007/978-3-030-88806-0_16.

20. *Gladstone A.* Module development with boost. Python and PyBind / C++ Software Interoperability for Windows Programmers: Connecting to C#, R, and Python Clients. Berkeley, CA: Apress, 2022, pp. 173—209. DOI: 10.1007/978-1-4842-7966-3_8.

21. *Belinassi G., Biener R., Hubichka J., Cordiero D., Goldman A.* Compiling files in parallel: A study with GCC / 2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW). IEEE, 2022, pp. 1—8.

22. *Hulton-Harrop T.* Minimal CMake: Learn the best bits of CMake to create and share your own libraries and applications. Packt Publishing Ltd. 2025, 266 p. ГИАБ

REFERENCES

1. Zemskov A. N., Liskova M. Yu., Zaalishvili V. B., Shamrin M. Yu. Modern technological and technical solutions for mining operations potash mines. *News of the Tula state university. Sciences of Earth*. 2022, no. 2, pp. 284—296. [In Russ]. DOI: 10.46689/2218-5194-2022-2-1-284-296.

2. Yurkevich N. V., Grosheva T. V., Edelev A. V., Gureev V. N., Mazov N. A. Modern approaches to the enrichment of barite ores. *Journal of Mining Institute*. 2024, vol. 270, pp. 977—993. [In Russ].

3. Konnova N. I., Chebokchinov I. P. A study of flotation ore sample treatment from a breakage ore occurence. *International Research Journal*. 2023, no. 12 (138), pp. 74. [In Russ]. DOI: 10.23670/IRJ.2023.138.165.

4. Silakov A. V., Varlamova S. A., Kotkov P. V. Software recognition of image defects of regular textures in the textile industry. *Proceedings of higher education institutions. Textile industry technology*. 2022, no. 2 (398), pp. 266—272. [In Russ]. DOI: 10.47367/0021-3497_2022_2_266.

5. Varlamova S. A., Zatonskiy A. V., Fedoseeva K. A. Illumination sensitivity analysis for the speck-based froth detection method using potash flotation machines. *Obogashchenie Rud*. 2021, no. 6, pp. 29—33. [In Russ]. DOI: 10.17580/or.2021.06.05.

6. Zatonskiy A. V., Varlamova S. A., Fedoseeva K. A. Improvement of computer recognition parameters in potash flotation machines by anti-glares from bubbles. *Bulletin of the South Ural state university. Computer technologies, automatic control, radio electronics*. 2022, vol. 22, no. 3, pp. 57—67. [In Russ]. DOI: 10.14529/ctcr220306.

7. Zatonskiy A. V., Fedoseeva K. A., Kuznetsov S. V. Improving the quality of recognition of polymetallic ore pellets in an image by adaptive contrast correction. *Innovative instrumentation*. 2024, vol. 3, no. 3, pp. 48—52. [In Russ]. DOI: 10.31799/2949-0693-2024-3-48-52.

8. Nemirovsky M., Tullsen D. *Multithreading architecture*. Springer Nature, 2022. 112 p.

9. Lee S. H. Real-time edge computing on multi-processes and multi-threading architectures for deep learning applications. *Microprocessors and Microsystems*. 2022, vol. 92, article 104554. DOI: 10.1016/j.micpro.2022.104554.

10. Wei X., Ma L., Zhang H., Liu Y. Multi-core-, multi-thread-based optimization algorithm for large-scale traveling salesman problem. *Alexandria Engineering Journal*. 2021, vol. 60, no. 1, pp. 189—197. DOI: 10.1016/j.aej.2020.06.055.

11. Feshina E.V., Omelchenko D.A., Gonataev R.G. Multithreading and asynchrony in the Python programming language. *Innovation. Science. Education*. 2021, no. 28, pp. 988—992. [In Russ].

12. Arjona A., Finol G., López P. G. Transparent serverless execution of Python multiprocessing applications. *Future Generation Computer Systems*. 2023, vol. 140, pp. 436—449. DOI: 10.1016/j.future.2022.10.038.

13. Ruys W., Lee H., You B., Talati S., Park J., Almgren-Bell J., Yan Y., Fernando M., Erez M., Gligoric M., Burtscher M., Rossbach C. J., Pingali K., Biros G. Performance characterization of python runtimes for multi-device task parallel programming. *International Journal of Parallel Programming*. 2025, vol. 53, no. 2, pp. 1—24. DOI: 10.1007/s10766-025-00788-1.

14. Sychev M.D., Trunova K.D. Performance comparison when performing bubble sorting in the C++ and Python programming languages. *Bulletin of Penza State University*. 2024, no. 4 (48), pp. 131—133. [In Russ].

15. Kaur A., Nayyar R. A comparative study of static code analysis tools for vulnerability detection in C/C++ and Java source code. *Procedia Computer Science*. 2020, vol. 171, pp. 2023—2029. DOI: 10.1016/j.procs.2020.04.217.

16. Naveed M. S. Comparison of C++ and Java in implementing introductory programming algorithms. *Quaid-E-Awam University Research Journal of Engineering, Science & Technology*. 2021, vol. 19, no. 1, pp. 95—103.

17. Oliveira C. Multithreading. *Practical C++20 Financial Programming*. Apress, Berkeley, CA. 2021, pp. 449—474. DOI: 10.1007/978-1-4842-6834-6_17.

18. Valero A., Gran-Tejero R., Suárez-Gracia D., Georgescu E. A., Ezpeleta J., Álvarez P., Muñoz A., Ramos L. M., Ibáñez P. A learning experience toward the understanding of abstraction-level

interactions in parallel applications. *Journal of Parallel and Distributed Computing*. 2021, vol. 156, pp. 38 — 52. ISSN 0743-7315. DOI: 10.1016/j.jpdc.2021.05.008.

19. Monat R., Ouadjaout A., Miné A. A multilanguage static analysis of Python programs with native C extensions. *International Static Analysis Symposium*. Cham: Springer International Publishing, 2021, pp. 323 — 345. DOI: 10.1007/978-3-030-88806-0_16.

20. Gladstone A. Module development with boost. Python and PyBind. *C++ Software Interoperability for Windows Programmers: Connecting to C#, R, and Python Clients*. Berkeley, CA: Apress, 2022, pp. 173 — 209. DOI: 10.1007/978-1-4842-7966-3_8.

21. Belinassi G., Biener R., Hubichka J., Cordiero D., Goldman A. Compiling files in parallel: A study with GCC. *2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*. IEEE, 2022, pp. 1 — 8.

22. Hulton-Harrop T. *Minimal CMake: Learn the best bits of CMake to create and share your own libraries and applications*. Packt Publishing Ltd. 2025, 266 p.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Затонский Андрей Владимирович[1] — д-р техн. наук,
профессор, зав. кафедрой, e-mail: zxenon@narod.ru,
ORCID ID: 0000-0003-1863-2535,
Кузнецов Станислав Викторович — магистрант,
Пермский национальный исследовательский политехнический
университет, e-mail: kuznetsovstas04@yandex.ru,
Плехов Павел Владимирович[1] — канд. техн. наук,
доцент, e-mail: onim@rambler.ru,
[1] Пермский национальный исследовательский политехнический университет,
Березниковский филиал.
Для контактов: Затонский А.В., e-mail: zxenon@narod.ru.

INFORMATION ABOUT THE AUTHORS

A.V. Zatonskiy[1], Dr. Sci. (Eng.), Professor,
Head of Chair, e-mail: zxenon@narod.ru,
ORCID ID: 0000-0003-1863-2535,
S.V. Kuznetsov, Master's Degree Student,
Perm National Research Polytechnic University,
Perm, Russia, e-mail: kuznetsovstas04@yandex.ru,
P.V. Plekhov[1], Cand. Sci. (Eng.), Assistant Professor,
e-mail: onim@rambler.ru,
[1] Perm National Research Polytechnic University, Berezniki Branch, Berezniki, Russia.
Corresponding author: A.V. Zatonskiy, e-mail: zxenon@narod.ru.

Получена редакцией 14.04.2026; получена после рецензии 26.05.2026; принята к печати 10.08.2026.
Received by the editors 14.04.2026; received after the review 26.05.2026; accepted for printing 10.08.2026.