25. PYTHON-БИБЛИОТЕКИ ДЛЯ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ПРИ РЕШЕНИИ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

Мацокин Н.П., магистрант гр. 376701

Мацокин М.П., магистрант гр. 376701

Белорусский государственный университет информатики и радиоэлектроники[1]
г. Минск, Республика Беларусь

Логинова И.П. – канд. тех. наук, доцент кафедры ЭИ

Аннотация. В статье представлен сравнительный анализ библиотек *Python* для параллельных матричных вычислений: *Numba*, *Dask*, *JAX* и *CuPy*. Рассматриваются их архитектурные особенности, поддержка *GPU* и *CPU*, а также производительность на различных задачах линейной алгебры. Исследование показывает, какие инструменты наиболее эффективны для конкретных вычислительных сценариев — от больших данных до ускоренных вычислений на *GPU*.

Ключевые слова. *Python*, параллельные вычисления, матричные операции, *Numba*, *Dask*, *JAX*, *CuPy*, *GPU*, *JIT*-компиляция, линейная алгебра.

Введение.

Современные вычислительные задачи всё чаще требуют эффективной обработки больших объёмов данных, что делает параллельные вычисления неотъемлемой частью научных исследований, машинного обучения и инженерных расчётов. В этом контексте *Python* остаётся одним из самых популярных языков благодаря богатой экосистеме библиотек, поддержке высокоуровневого программирования и интеграции с ускоренными вычислениями.

Работа с матрицами является ключевой операцией в таких областях, как численный анализ, обработка изображений, моделирование физических процессов и глубокое обучение. Однако стандартные инструменты *Python*, такие как *NumPy*, не всегда обеспечивают достаточную производительность, особенно при обработке больших данных. Для решения этой проблемы разработаны специализированные библиотеки, поддерживающие параллельные вычисления на *CPU* и *GPU*.

Основная часть.

В данной статье рассматриваются четыре популярных инструмента для параллельных матричных вычислений при решении линейных задач: *Numba*, *Dask*, *Jax* и *CuPy*. Эти библиотеки предоставляют разные механизмы ускорения, включая *JIT*-компиляцию, многопоточные и многопроцессорные вычисления, а также вычисления на *GPU*. Основной целью статьи является сравнительный анализ производительности этих инструментов и определение их сильных и слабых сторон в контексте различных вычислительных задач.

Numba – это *JIT*-компилятор, который позволяет ускорять *Python*-код с использованием технологии *LLVM*. Он предлагает [1]:

1 *JIT*-компиляция. С помощью декоратора *@jit* можно компилировать функции, что существенно повышает их производительность. *Numba* поддерживает как *CPU*, так и *GPU* через *CUDA.*

2 Многопоточность. *Numba* позволяет использовать параллельные вычисления на *CPU* с помощью декоратора *@prange* для обработки циклов с разделением работы между потоками.

3 Поддержка *CUDA*. *Numba* позволяет легко создавать и запускать код на *GPU* с использованием *@cuda.jit*, что даёт доступ к мощным возможностям параллельных вычислений.

Numba популярна среди разработчиков, которым нужно быстро ускорить код с минимальными усилиями и без изменения архитектуры программы.

Dask – это библиотека для параллельных и распределённых вычислений, которая позволяет работать с большими наборами данных и эффективно использовать многопроцессорные и многозадачные вычисления. Основные особенности [2]:

1 Масштабируемость. *Dask* позволяет обрабатывать данные, которые не помещаются в память, и эффективно использовать кластеры, поддерживая распределённые вычисления с использованием *dask.array*, *dask.dataframe* и других *API* обработкой данных в формате чанков.

2 Поддержка библиотек *NumPy* и *pandas*. *Dask* предоставляет интерфейсы, совместимые с *NumPy* и *pandas*, что упрощает работу с большими массивами данных, сохраняя при этом привычный *API*.

3 Параллельные вычисления. *Dask* может быть использован для параллельных вычислений на одном компьютере или распределённых вычислений на кластере. Он автоматически делит задачи на более мелкие части и распределяет их по вычислительным узламю

4 Гибкость. *Dask* подходит для работы с данными, которые не помещаются в оперативной памяти, и для сложных распределённых вычислений, таких как обучение моделей на больших данных.

Dask идеально подходит для обработки огромных массивов данных, которые выходят за пределы возможностей одного компьютера.

Jax – это библиотека для численных вычислений, предоставляющая поддержку автоматического дифференцирования и высокопроизводительных вычислений на *GPU* и *TPU*. *Jax* предоставляет такие возможности, как [3]:

1 *JIT*-компиляция. С помощью функции *jax.jit* код компилируется в машинный код, что значительно ускоряет выполнение. *JIT*-компиляция поддерживает как *CPU*, так и *GPU*.

2 Автоматическое дифференцирование. *Jax* интегрирует автоматическое дифференцирование с помощью *grad*, что полезно для оптимизации и обучения моделей.

3 Векторизация и параллелизация. Функции *vmap* и *pmap* позволяют легко векторизовать и параллелить операции для работы с большими наборами данных. *vmap* применяется для векторизации операций по батчам данных, а *pmap* — для распределённых вычислений на нескольких устройствах.

4 Поддержка *TPU*. *Jax* оптимизирован для работы с *TPU*, что делает его привлекательным для масштабируемых задач машинного обучения.

CuPy – это библиотека, предназначенная для выполнения операций, совместимых с *NumPy*, на *GPU* с использованием *CUDA*. Она предлагает [4]:

1 Совместимость с *NumPy*. *CuPy* предоставляет *API*, полностью совместимое с *NumPy*, что позволяет легко переносить код на *GPU* с минимальными изменениями.

2 *GPU*-ускоренные вычисления. *CuPy* выполняет операции матричного умножения, трансформации Фурье и другие вычисления на *GPU*, что значительно ускоряет выполнение задач по сравнению с *CPU*.

3 Поддержка *CUDA*. *CuPy* использует *CUDA* для работы с графическими процессорами *NVIDIA*, что даёт доступ к мощным вычислительным возможностям.

CuPy идеально подходит для задач, где требуется высокая производительность при обработке больших объёмов данных, и когда имеется доступ к графическому процессору.

Numba способствует оптимизации вычислений, но библиотеки *Jax*, *Dask* и *Cupy*, конкретно ускоряет математические операции с матрицами, реализованные модулем *NumPy*.

Для проверки прироста производительности при использовании этих библиотек для распределённых вычислений при решении задач линейной алгебры был использован ПК с конкретными характеристиками (таблица 1).

Таблица 1 – Технические характеристики ПК

Операционная система	*Windows 10*
Версия *Python*	3.12.7
Размер дискового пространства	512 ГБ
Размер ОЗУ	16 ГБ
CPU	*12th Gen Intel(R) Core(TM) i7-12700H 2.70 GHz*
GPU	*NVIDIA GeForce RTX ы3060 Laptop GPU*

Технологии *Dask*, *Jax*, *CuPy* вместо с исходной библиотекой *NumPy* были протестированы по трём математическим операциям:

- матричное перемножение квадратичных матриц размеров N;
- решение линейной системы $Ax = b$ при размере квадратичной матрицы A равным N и длине вектора B равной N;
- сингулярное разложение квадратичной матрицы N.

В итоге проведения расчётов получились характерные результаты (таблица 2).

Таблица 2 – Результаты тестирования библиотек

Операция	Значение *N*	Время операции, с.			
		NumPy	*Dask*	*Jax (CPU)*	*CuPy*
1	2	3	4	5	6
Матричное умножение	100	0.001	0.148	0.007	0.098
	1000	0.061	0.063	0.006	0.001
	5000	5.231	5.911	0.008	0.001
	10000	38.853	48.469	0.285	0.089

Продолжение таблицы 2

1	2	3	4	5	6
Решение линейного уравнения	100	0.002	0.279	0.106	0.979
	1000	0.082	0.204	0.059	0.055
	5000	10.341	5.209	0.048	0.617
	10000	34.078	36.983	0.052	10.934
Сингулярное разложение	100	0.025	0.303	0.023	0.292
	1000	1.394	0.303	0.018	2.204
	5000	188.301	2.526	0.02	160.452

Заключение.

Производительность зависит от типа вычислений. *Numba* отлично показывает себя на простых операциях (арифметика, матричные произведения), благодаря *JIT*-компиляции, но не поддерживает сложные линейные операции. *JAX* даёт огромное ускорение на *CPU* и *GPU*, но требует прогрева. *Dask* подходит для распределённых вычислений и работы с большими данными, но некоторые функции требуют специальных форматов чанков. *CuPy* максимально эффективен на больших плотных матрицах, так как использует *GPU*-ускорение.

Можно сказать, что для работы с *GPU*, в основном, понадобятся *JAX* или *CuPy*. Для больших данных и с взаимодействием распределёнными датафреймами, например в Pandas, хорошим вариантом будет *Dask*. Для высокопроизводительных *CPU*-вычислений пригодится *Numba*, но только простые операции. Для общего использования можно использовать связку *JAX*, *NumPy* и *Numba*, что даёт достаточный выигрыш в производительности при параллельных вычислениях, и не только при решении линейной алгебры.

Список использованных источников:

1. Лавринов М. И. ОБЗОР JIT-КОМПИЛЯТОРА NUMBA КАК ИНСТРУМЕНТА ДЛЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА GPU // Современные наукоемкие технологии. – 2022. – № 2. – С. 67-71..

2. Guiding Tech Media [Электронный ресурс]. – An Introduction to Dask: The Python Data Scientist's Power Tool. – Режим доступа: https://www.kdnuggets.com/introduction-dask-python-data-scientist-power-tool. Дата доступа 07.03.2025.

3. Jaxoplanet [Электронный ресурс]. – A brief introduction to JAX. – Режим доступа: https://jax.exoplanet.codes/en/latest/tutorials/introduction-to-jax/. Дата доступа 18.03.2025.

4. Okuta R. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations / R. Okuta, Y. Unno, D. Nishino // ML Systems Workshop in NIPS. – 2017.

PYTHON LIBRARIES FOR PARALLEL COMPUTING IN SOLVING LINEAR ALGEBRA PROBLEMS

Matsokin N.P.

Matsokin M.P

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Loginova I.P. – PhD in Technical Science, associate professor of the department of Economic Informatics

Annotation. This paper presents a comparative analysis of Python libraries for parallel matrix computations: Numba, Dask, JAX, and CuPy. It examines architectural features, support for GPU/CPU, and performance in various linear algebra tasks. The study highlights which tools perform best in specific scenarios, from large-scale data processing to GPU-accelerated operations.

Keywords. Python, parallel computing, matrix operations, Numba, Dask, JAX, CuPy, GPU, JIT compilation, linear algebra.