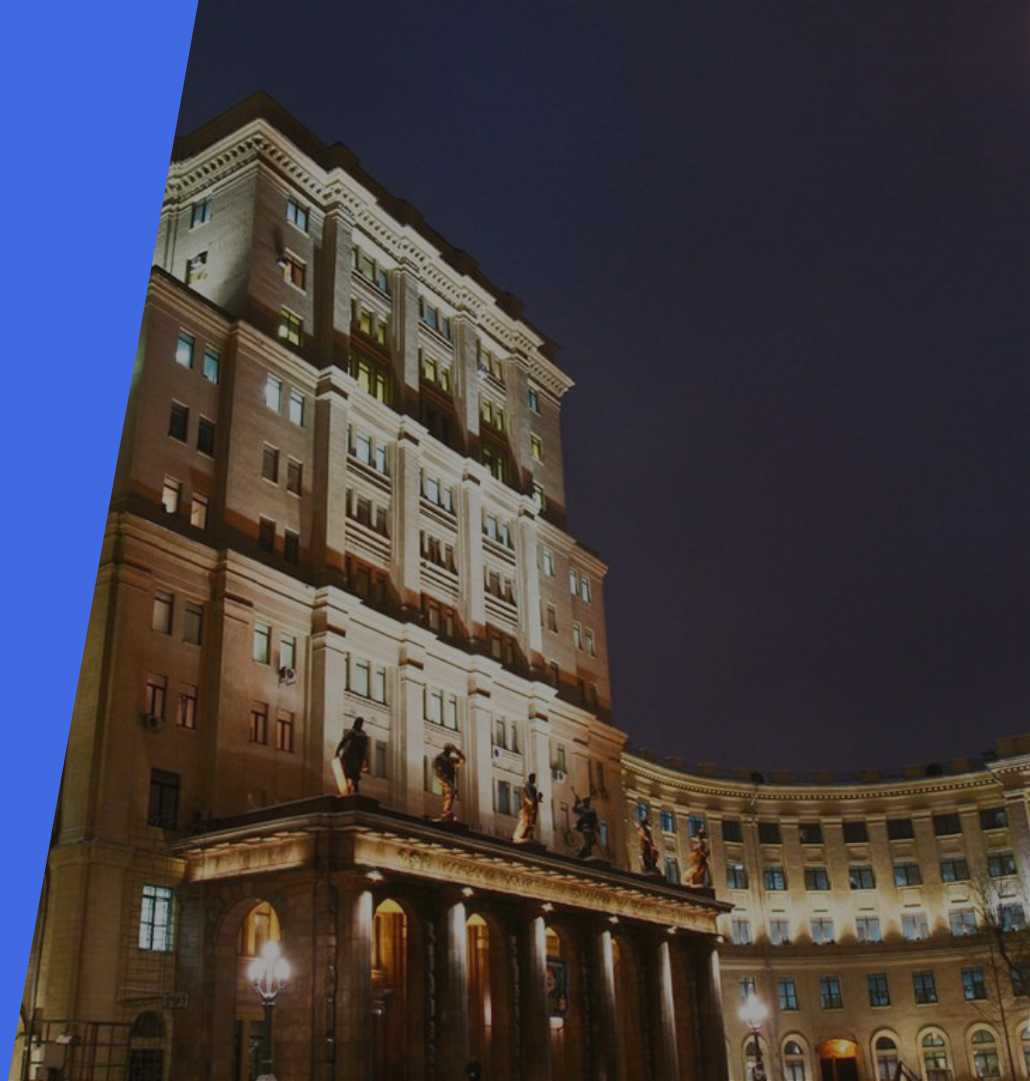
Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)

Архитектура распределенной вычислительной системы с аппаратной обработкой графовых структур и ассоциативной памятью

Алексей Юрьевич Попов,
д.т.н., профессор кафедры Компьютерных систем и сетей

Суперкомпьютерные дни в России, международная научная конференция,
29 - 30 сентября, Москва, сентябрь 2025 г.

01
Графы знаний



Перспективы использования графов знаний

Финансовый сектор

Борьба с мошенничеством
Управление рисками
Кредитный скоринг

Биология и медицина

Биологическая безопасность
Синтез новых лекарств
Персонализированная медицина
Интеграция геномных, протеомных, клинических данных
Диагностика редких заболеваний

Безопасность

Расследование кибер-инцидентов
Мониторинг критической инфраструктуры
Противодействие экстремизму и орг. преступности
Цифровая криминалистика

Искусственный интеллект

Объяснимый ИИ
Семантический поиск
Обучение с пополнением
Когнитивные архитектуры
Хранение контекста предметной области для LLM

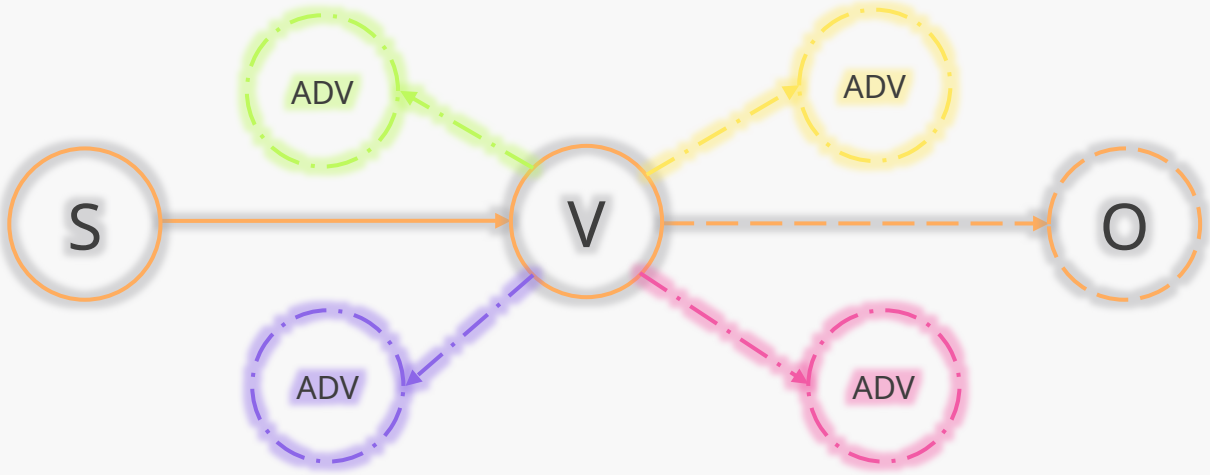
Графы знаний в биологии

- Интерактомика
- Анализ проблемы индивидуальной нормы
- Моделирование и визуализация процессов в биологических системах
- Моделирование и анализ популяций и сложных сообществ

Графы знаний в системах ИИ

- Обработка событий
- Построение алгоритмов
- Принятие решений
- Выдвижение гипотез
- Самообучение

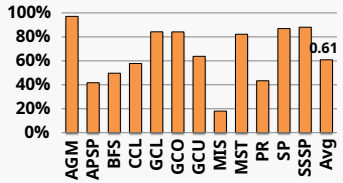
Образы действия, время, места, эмоциональный окрас, цели и причины непосредственно связаны с вершиной V



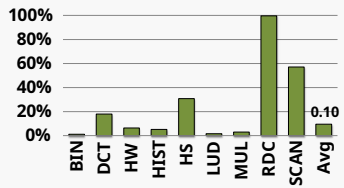
Проблемы обработки графов

Количество кэш-промахов*

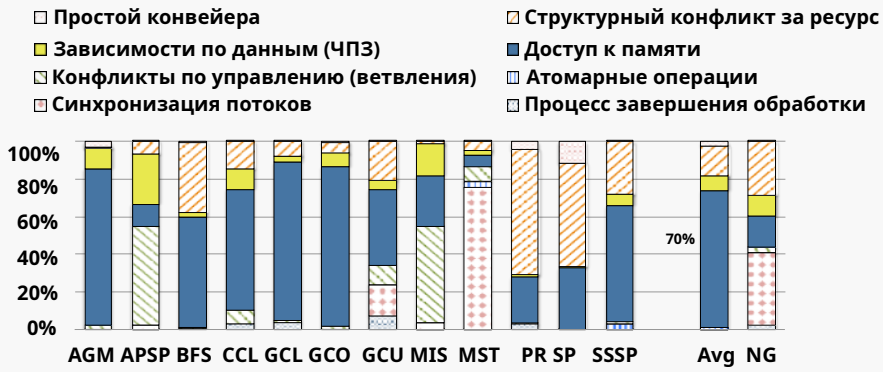
Графовые алгоритмы



Вычислительные алгоритмы



Распределение времени обработки*

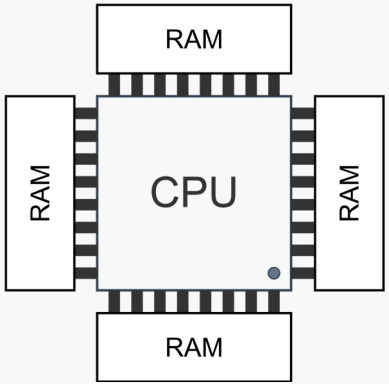


*Q. Xu, H. Jeon and M. Annavaram, "Graph processing on GPUs: Where are the bottlenecks?," 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, NC, USA, 2014, pp. 140-149, doi: 10.1109/IISWC.2014.6983053.

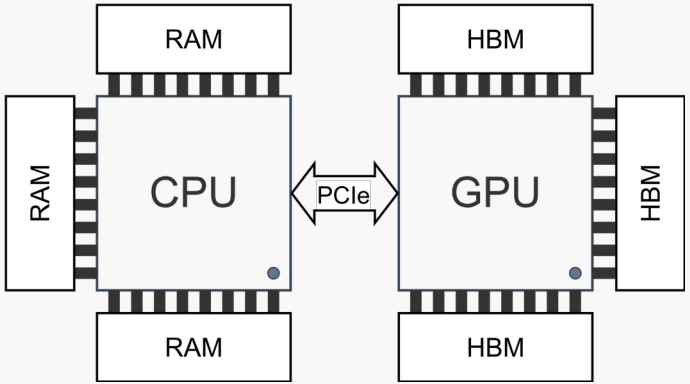
- Алгоритмы с зависимостями по данным
- Нерегулярная структура графа
- Доступ к памяти по случайным адресам
- Большое количество кэш-промахов
- Время доступа к данным больше времени обработки (может использоваться кроткий реплей)
- MISD вместо SIMD/SIMT
- Распределенная обработка затруднительна
- Алгоритмы обработки графов знаний должны использовать значения атрибутов вершин и связей

Проблема ограничения пропускной способности памяти

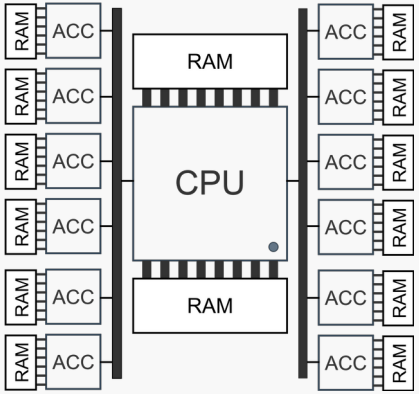
CPU



CPU+GPU



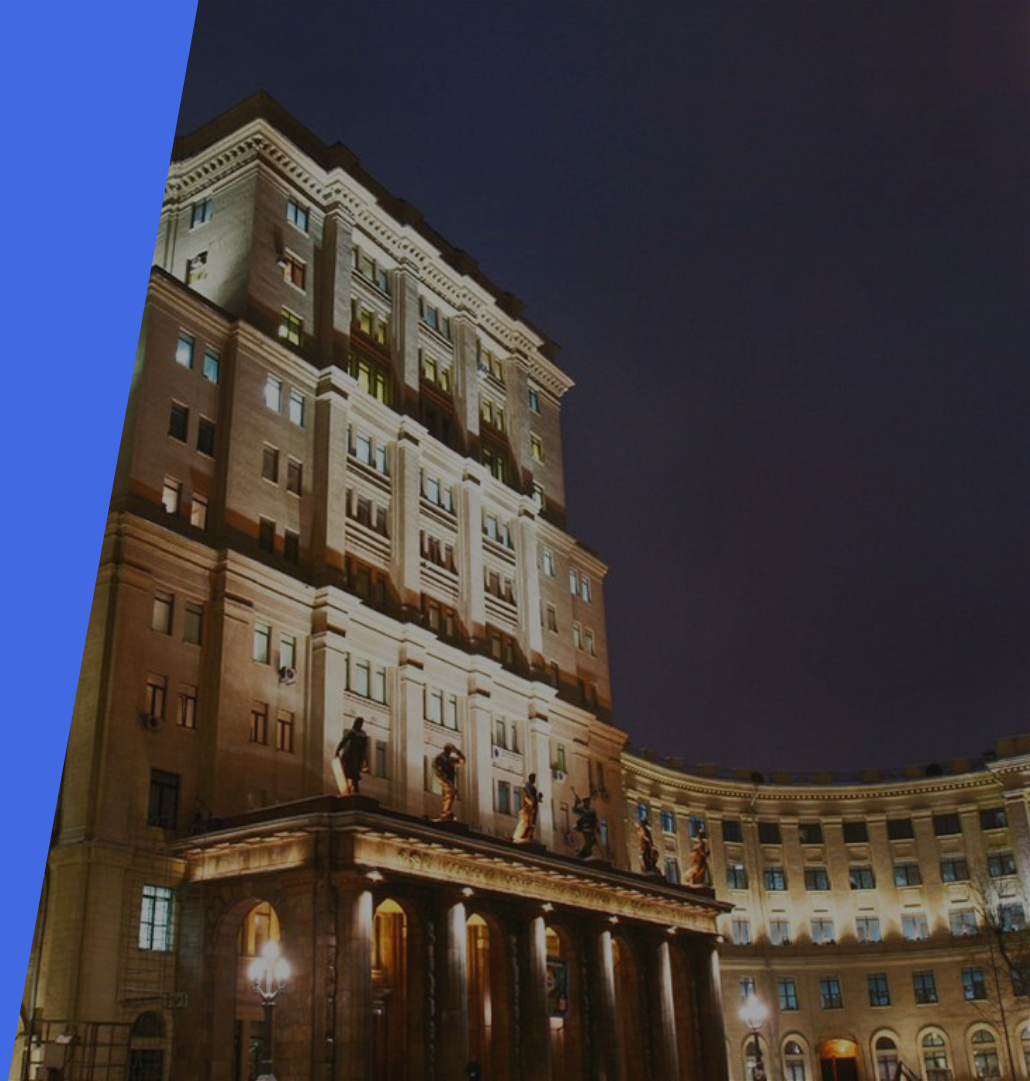
CPU+n*ACC



- **Фон-Неймановской архитектуры** - основана на применении высокопроизводительных CPU. Рост производительности упираются в ограниченное число каналов памяти на чип. Множество потоков создают конкуренцию за доступ, в результате чего рост производительности становится невозможным.
- **Гетерогенная архитектура на основе CPU и GPU**. Используемая совместно с GPU память HBM обеспечивает более высокую пропускную способность для регулярных операций. Стоимость HBM высокая, однако нерегулярный шаблон доступа снижает ее эффективность.
- **Массивно-параллельная гетерогенная архитектура** - множество небольших CPU и аппаратных ускорителей вычислений, связанных с центральным процессорным устройством (хост-микропроцессором) позволяют распределить вычислительную нагрузку на большое количество каналов памяти. Каждый ускоритель обладает собственным каналом памяти, а их совокупная пропускная способность превосходит другие системы.

02

Вычислительный комплекс «Тераграф»



Операции, кванторы и функции

Теоретико-множественные операции, кванторы, функции	Описание	Инструкции набора команд дискретной математики (DISC)
$A = (A_1, ..., A_n)$	Функция хранения n множеств как кортежа A	Insert
$R(A_i, x, y), x \in A_i, y \in A_i$	Отношение между x и y в множестве A_i	Next/Previous/Neighbors
$\|A_i\|, i = 1,n$	Мощность множества A_i	Cardinality
$x \in A_i, x \notin A_i, i = 1,n$	Проверка принадлежности x множеству A_i	Search
$A_i \cup x, i = 1,n$	Добавление элемента x в множество A_i	Insert
$A_i \setminus x, i = 1,n$	Удаление элемента x из множества A_i	Delete, Delete structure
$A \setminus A_i$	Удаление множества A_i из кортежа A	Delete structure
$A_i \subset A_j$	Отношение включения множества A_i в A_j	Slices
$A_i \equiv A_j$	Операция отношения эквивалентности	Slices
$A_i \cup A_j$	Операция объединения двух множеств	OR
$A_i \cap A_j$	Операция пересечения двух множеств	AND
$A_i \setminus A_j$	Операция разности множеств	NOT
$A_i \triangle A_j$	Симметрическая разность	-
$\bar{A}_i$	Дополнение множества A_i	NOT
$A_i \times A_j$	Операция декартова произведения	-
2^{A_i}	Булеан (множество всех подмножеств)	-

Набор команд дискретной математики DISC

Операции, основанные на поиске

- Поиск по ключу *SRCH*
- Поиск минимального *MIN*
- Поиск максимального *MAX*
- Поиск следующего *NEXT*
- Поиск предыдущего *PREV*
- Ближайший больший *NGR*
- Ближайший меньший *NSM*

Операции добавления/удаления

- Вставка *INS*
- Удаление *DEL*
- Удаление множества *DELS*

Операции И-ИЛИ-НЕ

- Объединение множеств *OR*
- Пересечение множеств *AND*
- Дополнение множеств *NOT*

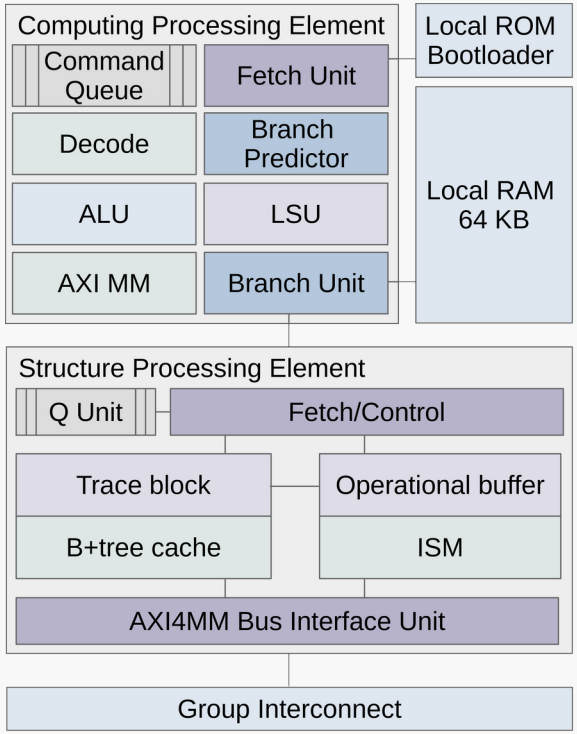
Операции среза

- Срез больше *GR*
- Срез больше или равно *GREQ*
- Срез меньше *LS*
- Срез меньше или равно *LSEQ*
- Срез меньше/больше *GRLS*

Свойства множеств

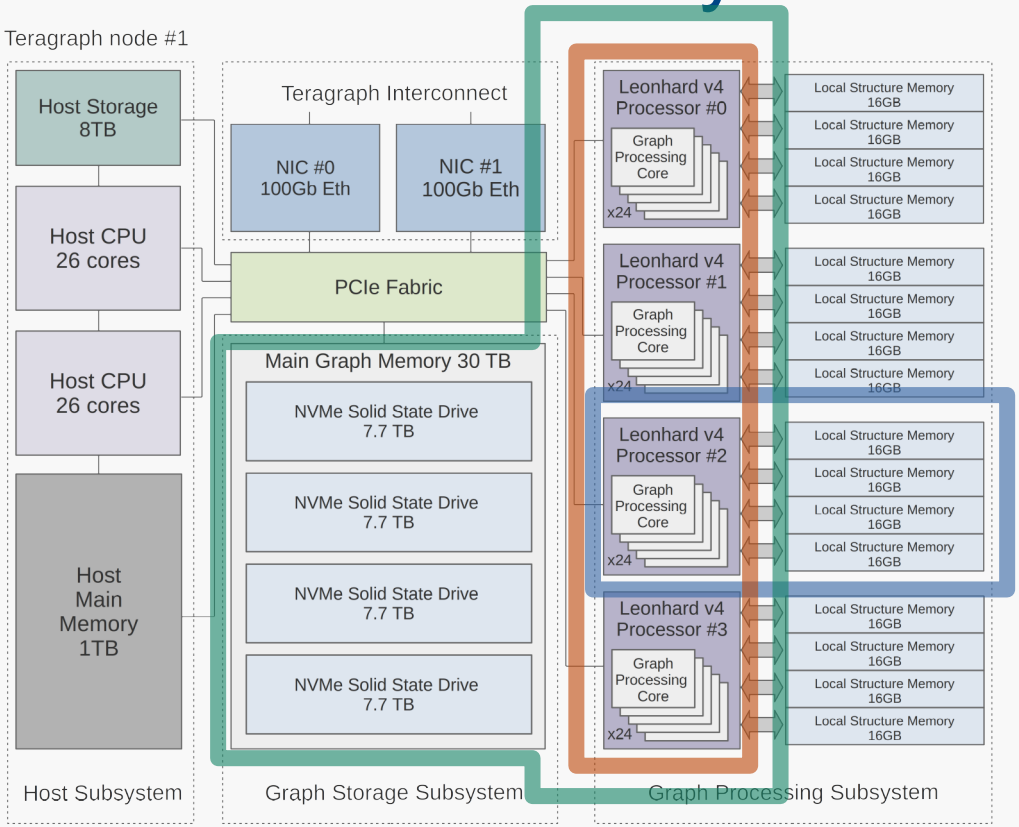
- Мощность множества *CNT*

Гетерогенное ядро обработки графов



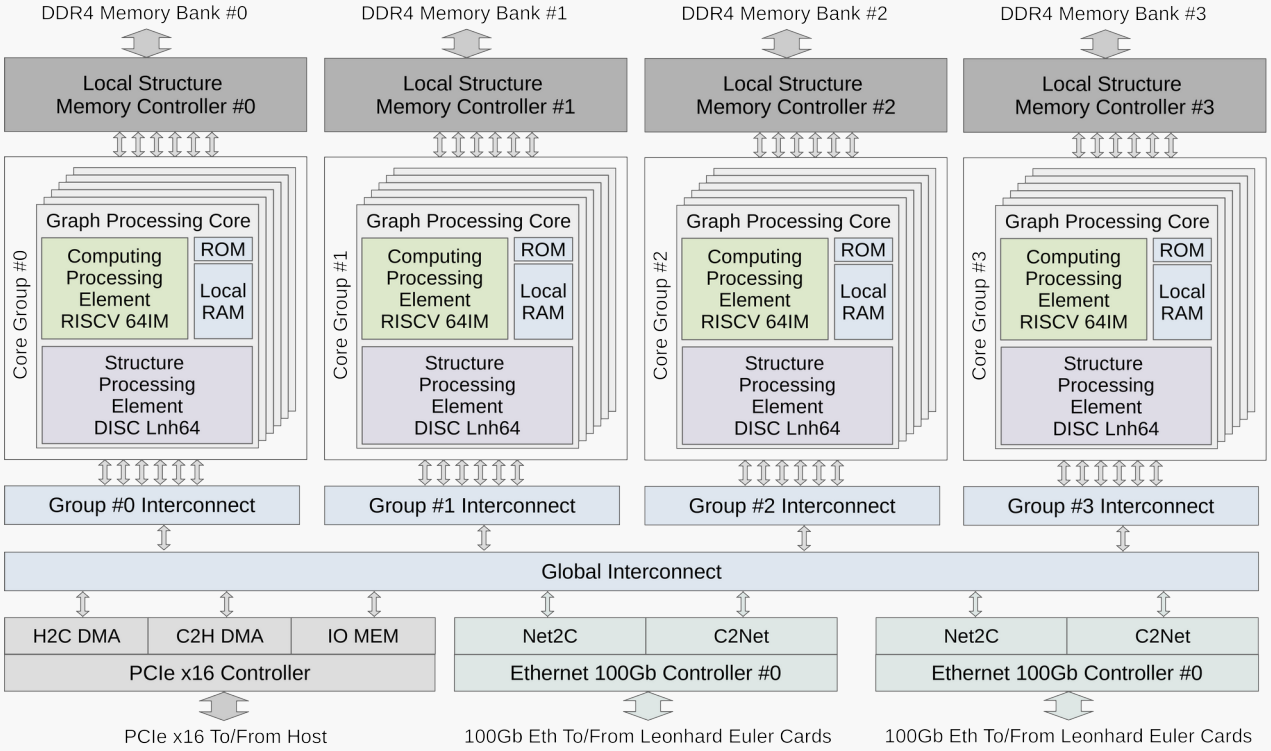
- GPC состит из двух тесно связанных микропроцессоров: Computing Processor Element (CPE) и Structure Processing Element (SPE).
- CPE реализован на базе микропроцессора с набором команд riscv64im.
- SPE представляет собой микропроцессор с набром команд дискретной математики DISC
- SPE подключен, как ускорительное ядро к шине памяти CPE.
- Производительность GPC сопоставима с производительностью одного ядра Intel Xeon Platinum v8 при 10x меньшей частоте (~200 МГц) и 40x меньшем количестве вентилей (2.5 млн.)

Вычислительный узел комплекса Тераграф



- Предусмотрено длительное размещение графов в оперативном доступе.
- Используется ассоциативная память большого объема (2.5ГБ на одно ядро Graph Processing Core, GPC)
- Ядра GPC являются весокоэффективными гетерогенными системами, взаимодействующими через единой адресное пространство PCIe
- GPC самостоятельно обращается в локальное графовое хранилище 30ТБ и графовые хранилища других узлов
- Хост система выполняют второстепенные функции (инициализация,распределение и т.д.)

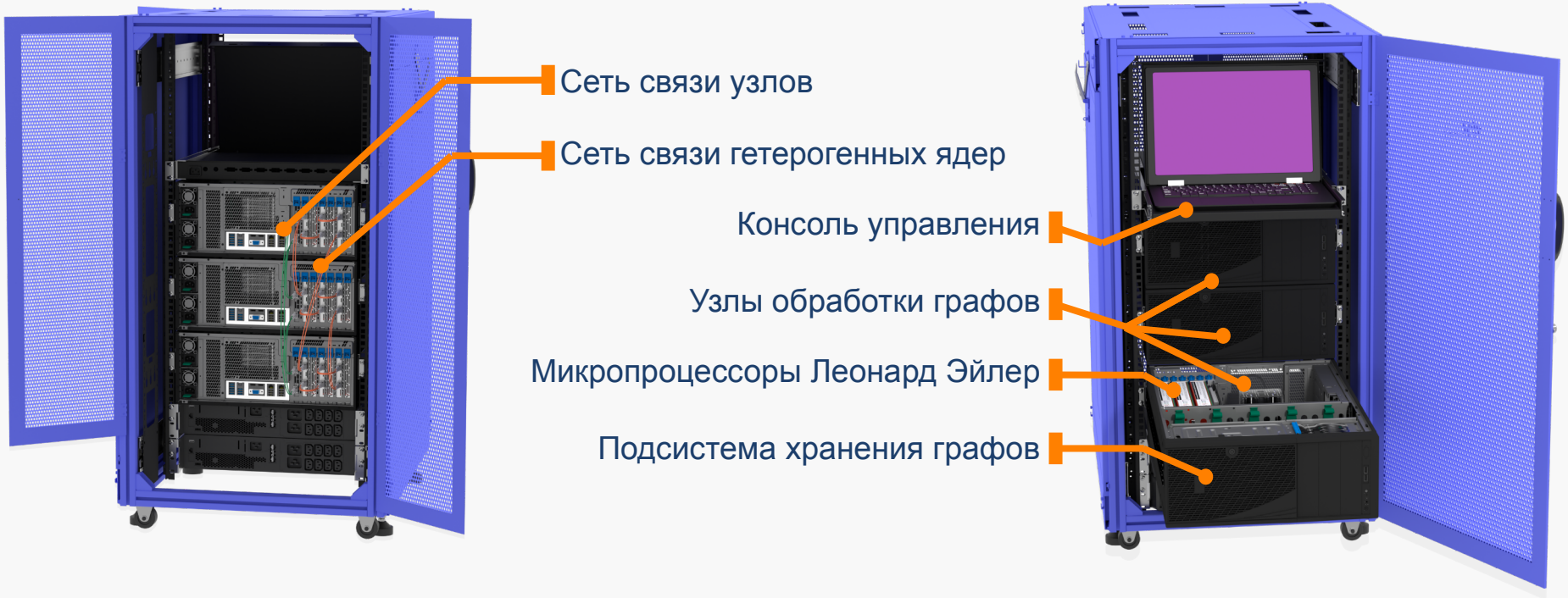
Микропроцессор Леонард Эйлер



- Ядра GPC объединяются в группы ядер (до 6 ядер в группе)
- В каждой группе предусмотрена глобальная память 128КБ для обмена данными между хост-подсистемой и CPE.
- В каждом ядре CPE предусмотрены аппаратные очереди сообщений Host2GPC и GPC2Host на 512 записей по 64 бит каждая.
- Все GPC в одной группе подключены к одной шине памяти DDR4 (16ГБ).
- Хост-подсистема может независимо управлять каждым GPC в отдельности.
- Основным программным компонентов программного ядра является обработчик (подобно шейдеру), который написан на языке C/C++ и загружается по запросу хост-системы.
- На хост подсистеме может быть использованы би блиотеки на языке C/C++ или Python.

Примеры, исходные коды библиотек: https://alexbmstu.github.io/2025

Вычислительный комплекс Тераграф



Производительность микропроцессора Леонард Эйлер

Характеристика	lnh64 v4	Intel Xeon Platinum 8168, 2.7МГц		
Тип структуры данных	B+ tree	Red-Black tree	B+ tree	ART
Реализация набора команд DISC	аппаратная	программная		
Количество ядер	1	1	1	1
Тактовая частота (MHz)	267	2700	2700	2700
Количество вентилей (млн.)	2.5	83	83	83
Скорость вставки последовательности ключей (операций/секунду)	1179495	719948	2463484	7356619
Скорость вставки случайных ключей (операций/секунду)	200607	667519	1358625	5278771
Скорость поиска по последовательным ключам (операций/секунду)	7570905	1168116	6354895	15860176
Скорость поиска по случайным ключам (операций/секунду)	329867	614191	1341165	4812921
Скорость последовательного удаления (операций/секунду)	1649342	788151	2689176	11228889
Скорость удаления случайных ключей (операций/секунду)	313392	491050	941129	3903154

Характеристики микропроцессора lnh64:

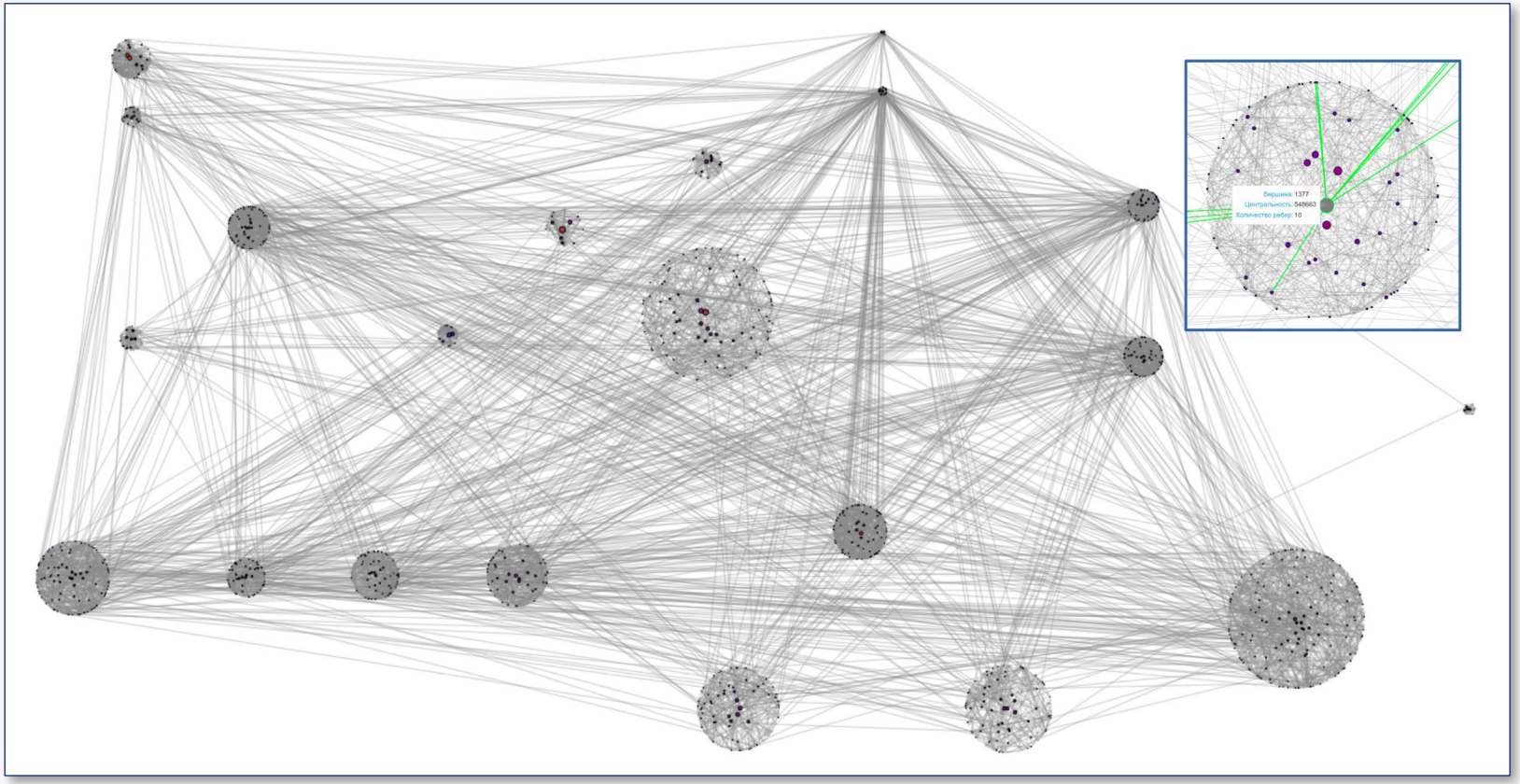
- 1-24 ядра обработки графов GPC64/100
- 64 ГБ память структур
- До 2,8 миллиарда ключей и значений
- Программный API интерфейс C/C++, Python
- Тактовая частота: 267 МГц

Код тестов и результаты доступны по адресу: https://latex.bmstu.ru/gitlab/hackathon/lnh64_speedtest

Превосходство по тактам

Превосходство по тактам и времени

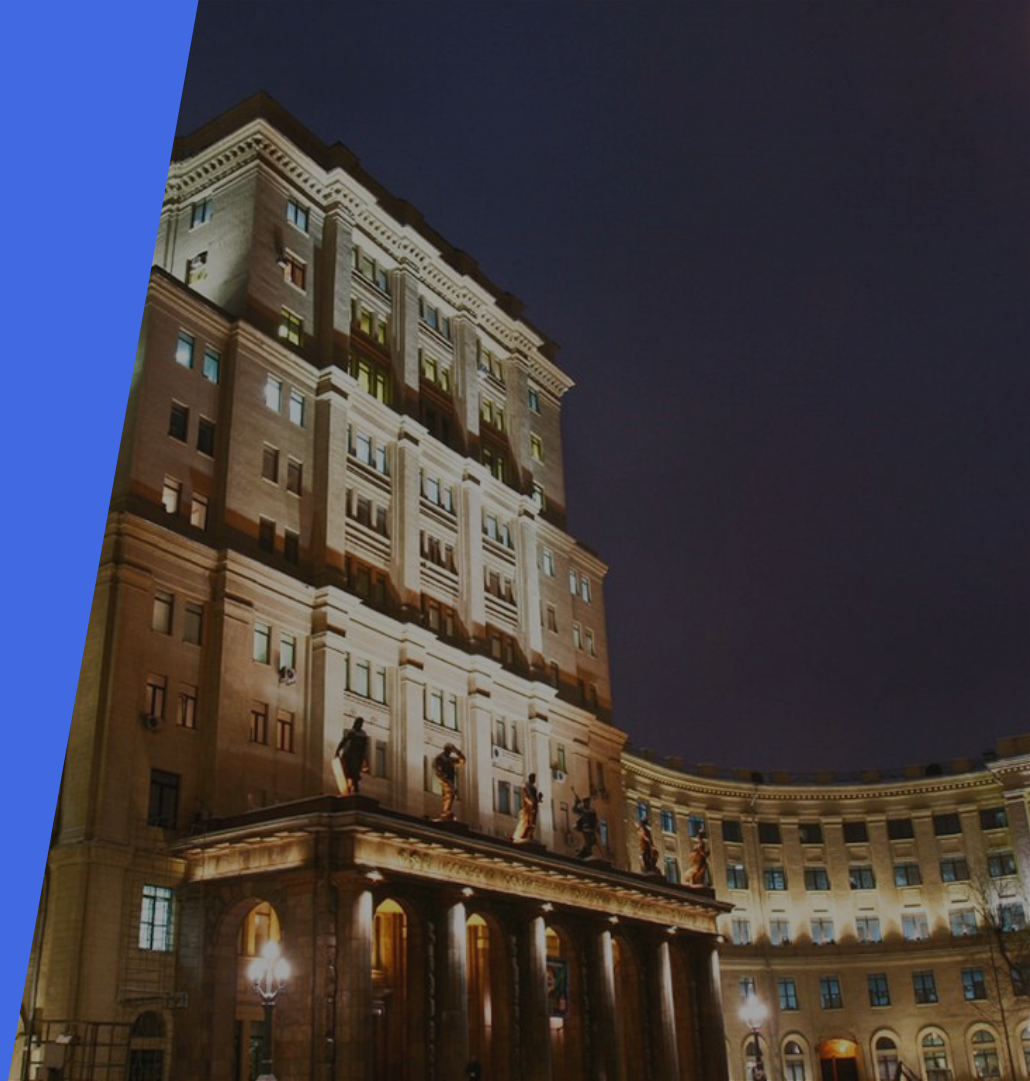
Пример работы одного гетерогенного ядра Тераграф



Определение сообществ и центральности
~4K вершин, 16 млн кратчайших путей

03

Внедрение ВК Тераграф



Открытая документация

2.5. Библиотека lnh64 L0

Библиотека lnh64 L0 (библиотека уровня "ноль") представляет собой API системного уровня, реализующего базовые функции взаимодействия с драйвером ускорительной карты микропроцессора Леонард Эйлер. Библиотека позволяет:

- Создание файлового дескриптора для эксклюзивного доступа к символьному устройству GPC `/dev/gpc*`.
- Инициализация вычислительного элемента CPE.
- Запуск обработчика вычислительного элемента CPE.
- Прием и передачи сообщений для взаимодействия Host и CPE.
- Взаимодействие CPE и SPE (микропроцессором lnh64 DISC) .

Библиотека разделена на две части, представленные в таблице 2.

Таблица 2 - Описание частей библиотеки lnh64 L0

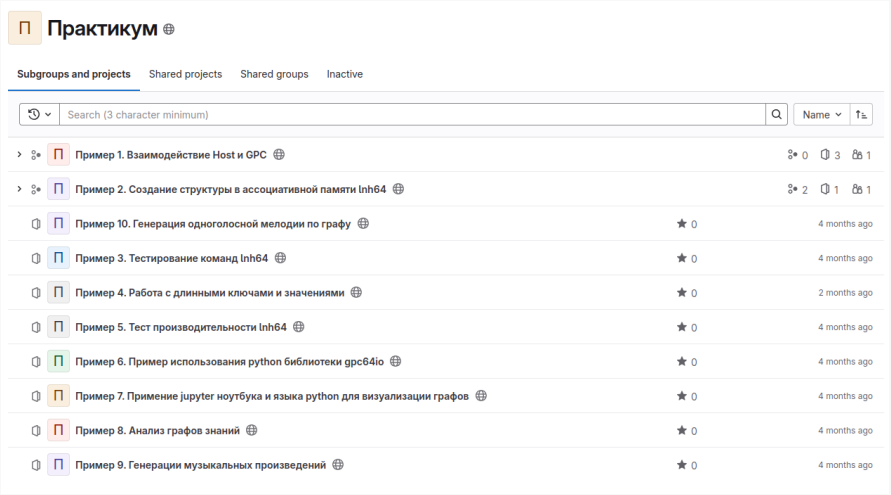
Раздел библиотеки	Описание	Язык программирования	Архитектура, Компилятор	Способ отладки
Host Lib	Управления хост-подсистемой и взаимодействие с ядрами GPC	C/C++, объектная модель	x86, g++	Jupyter,VSCode,gdb
SW Kernel Lib	Взаимодействие с микропроцессором lnh64	C/C++, процедурная модель	riscv64, g++	Вывод сообщений

Функциональные возможности Host Lib:

- Получение информации от дайвера о количестве и состоянии гетерогенных ядер gpc
- Открытие указанного пользователем или любого свободного gpc в эксклюзивным доступ пользователя.

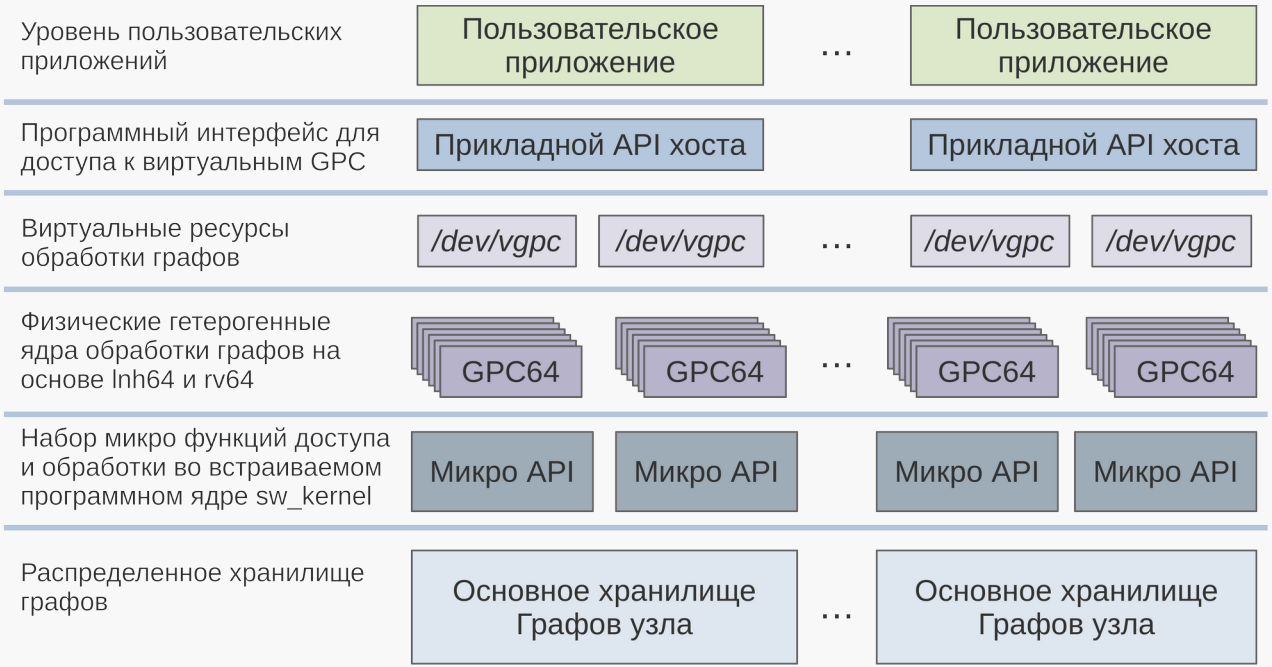
Ссылка: https://alexbmstu.github.io/2025/

Открытый репозиторий библиотек и примеров



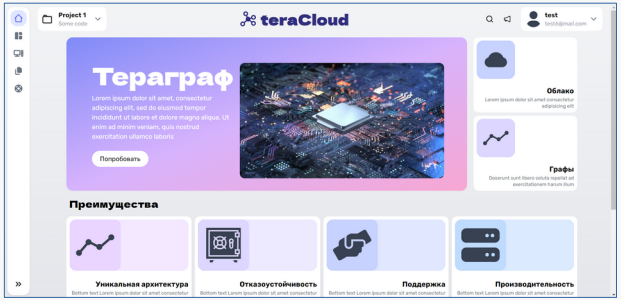
Ссылка: https://latex.bmstu.ru/gitlab/hackathon

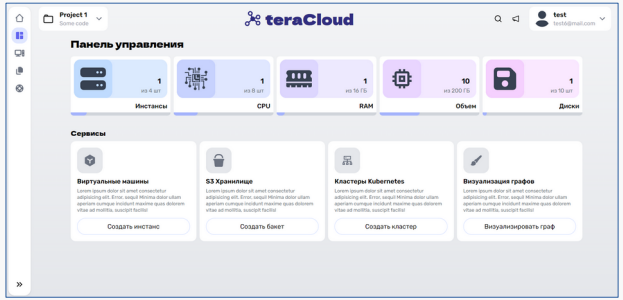
Облачная платформа Терраграф Cloud

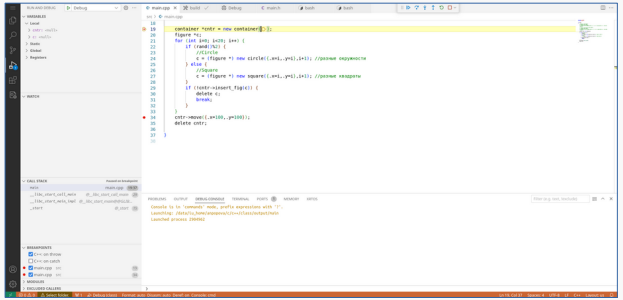


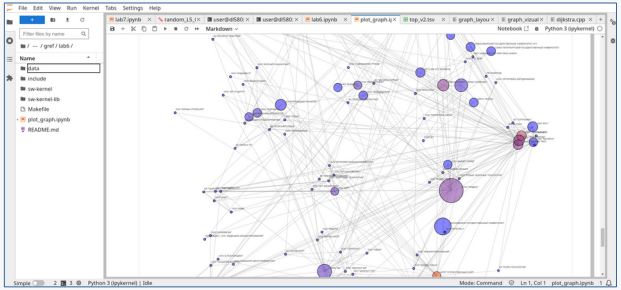
- Современные средства разработки программ для хост-подсистем и программных ядер.
- Мониторинг состояния гетерогенных ядер.
- Изоляция ресурсов пользователей.
- Отладка программ хост-подсистемы с использованием открытых отладочных средств.
- Готовые библиотеки с открытым исходным кодом для языков C/C++ и Python.

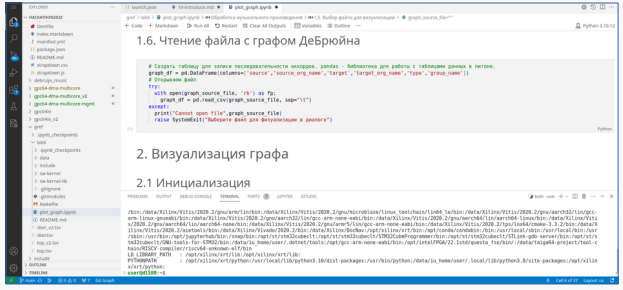
Облачная платформа Тераграф Cloud

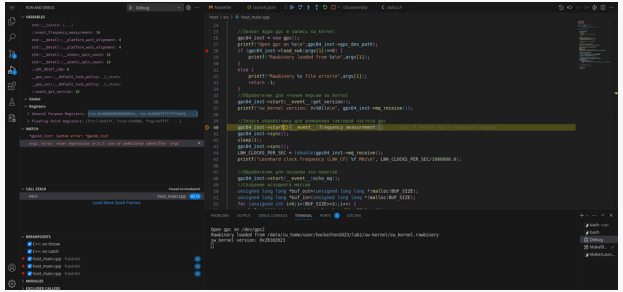












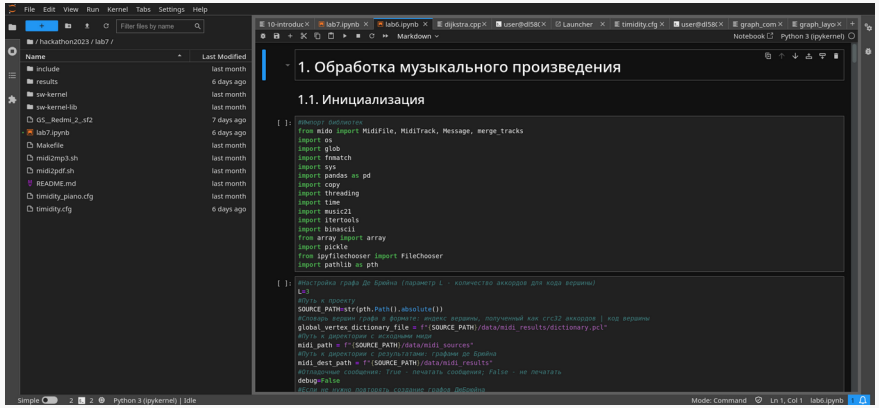
Выделенные и временные ресурсы. Облачная платформа обеспечивает выделение пользователю на длительное время ресурсов: ассоциативное хранилище, файловой хранилище, логические группы GPC, GPU, CPU. Ресурсы могут быть изъяты у пользователя в соответствии с политикой управления.

Программная модель системы. Для реализации пользовательских программ разработаны системные библиотеки, а также библиотеки для управления GPC с помощью языков C/C++ и Python.

Анализ и визуализация графов. В составе библиотек реализованы средства анализа и их визуализация.

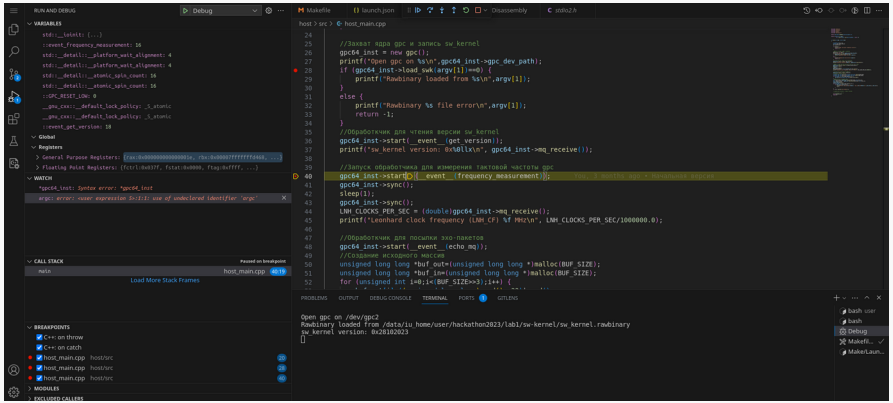
Среды разработки

Среда разработки и отладки программ JupyterHub



- Использование наиболее востребованного продукта Jupyter Notebook.
- Библиотка lnh64io для языка Python с открытым исходным кодом (https://latex.bmstu.ru/gitlab/hackathon2023)
- Утилита мониторинга lnh_nfo.

Среда разработки и отладки VSCode



- Возможность отладки кода на C/C++, Python и других популярных языков программирования.
- Поддержка Jupyter Notebook.
- Средства отладки
- Большое количество плагинов расширения.

Облачная платформа Тераграф Cloud

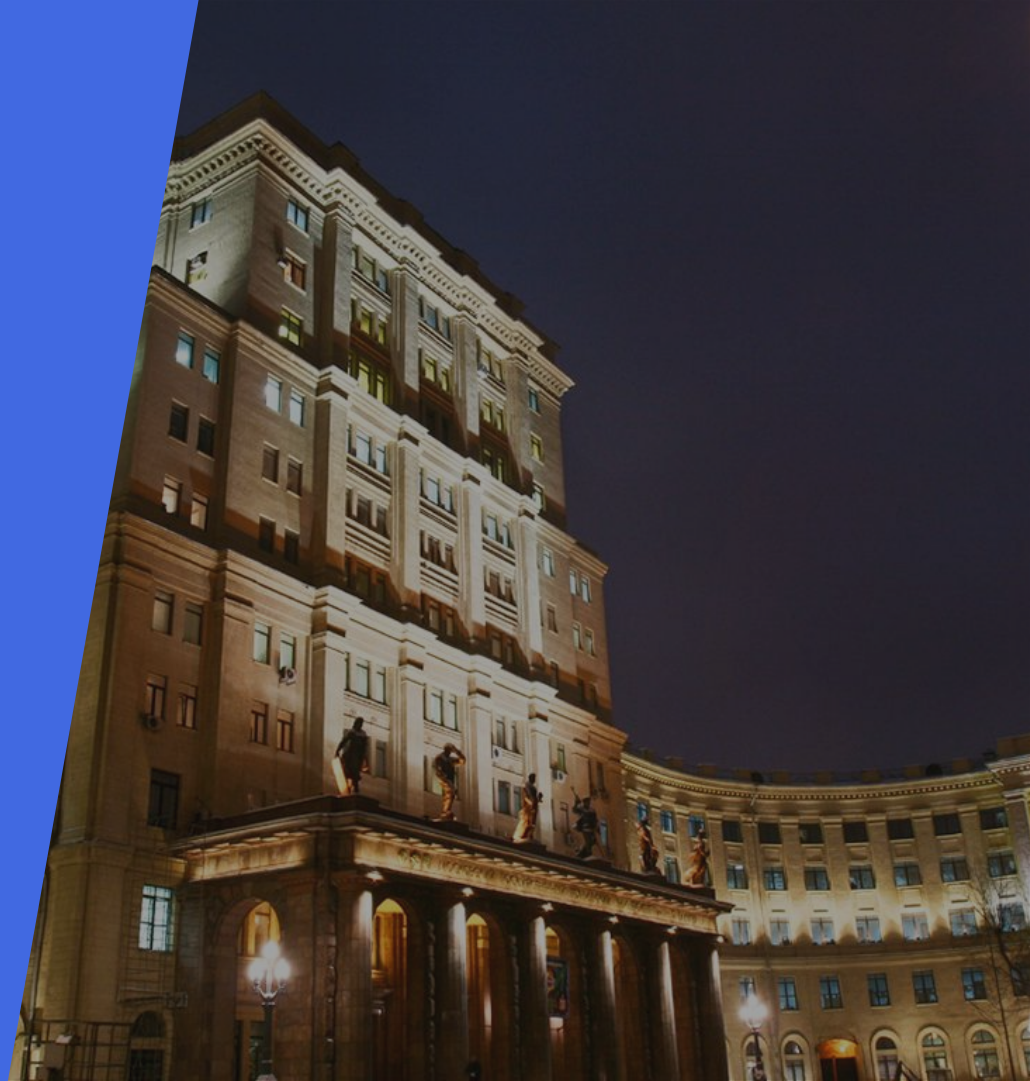

```
Утилита для мониторинга состояния гетерогенных ядер обработки графов.
Использование:  lnh_nfo -t/j [-l0,6,..]
-t - вывод в виде таблицы (по умолчанию); -j - вывод в виде json; -l - список ядер gpc (номера через запятую)
Условные обозначения:  <ключи> - количество ключей в структурах 1..7;  <атрибуты> - состояние b+поддеревьев 0..7;
<O> - флаг переполнения поддерева; <E> - флаг пустого поддерева; <S> - номер структуры, занимающей поддерево;
<busy> - ядро выполняет обработчик; <rdy> - ядро ожидает запуска обработчика; * - символическое устройство открыто
```

ядро#	параметр	#0	#1	#2	#3	#4	#5	#6	#7
* 0 busy	ключи	-	256	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=1	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
* 1 busy	ключи	-	256	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=1	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
2 busy	ключи	-	5000	10000	10000	0	0	0	0
	атрибуты	O=0 E=1 S=1	O=0 E=1 S=2	O=0 E=1 S=3	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
3 busy	ключи	-	0	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
6 busy	ключи	-	10457	2135	2035	50	0	0	0
	атрибуты	O=0 E=1 S=1	O=0 E=1 S=2	O=0 E=1 S=3	O=0 E=1 S=4	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
12 ready	ключи	-	0	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
18 ready	ключи	-	0	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
19 busy	ключи	-	0	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
20 ready	ключи	-	100	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=1	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0
21 ready	ключи	-	100	0	0	0	0	0	0
	атрибуты	O=0 E=1 S=1	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0	O=0 E=1 S=0

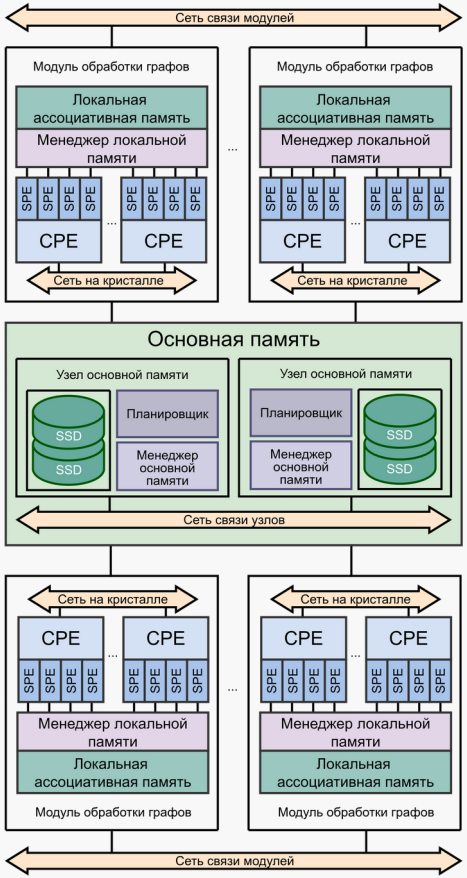
- Современные средства разработки программ для хост-подсистем и программных ядер.
- Мониторинг состояния гетерогенных ядер.
- Изоляция ресурсов пользователей.
- Отладка программ хост-подсистемы с использованием открытых отладочных средств.
- Готовые библиотеки с открытым исходным кодом для языков C/C++ и Python.

04

Архитектура перспективной вычислительной системы



Структурная схема системы



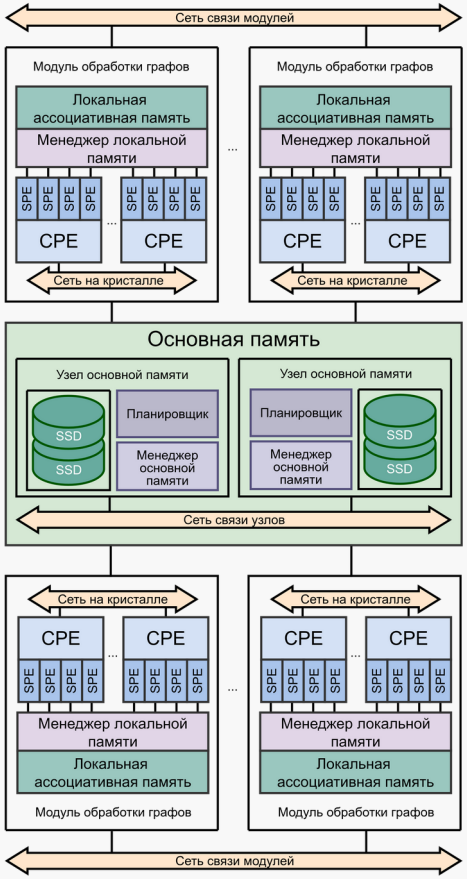
Основная память представляет собой центральное хранилище данных, построенное на основе высокопроизводительных блочных накопителей (SSD), распределенное между несколькими узлами вычислительного комплекса.

Физическое размещение данных на носителях остается распределенным и не гарантирует пространственной локальности для произвольных графовых запросов. Это приводит к необходимости множественных обращений к различным областям хранения при обработке связанных данных, что создает узкое место в производительности системы.

Модуль обработки графов — это гетерогенная вычислительная структура, спроектированная для эффективной работы с графовыми данными. Его архитектура сочетает специализированные процессорные элементы разных типов, оптимизированные для конкретных задач: обхода префиксных деревьев, обработки композитных ключей и работы со значениями переменной структуры.

Модуль функционирует по принципу "near-memory computing", загружая из основной памяти необходимые для текущей задачи подграфы и организуя их в локальном пространстве для минимизации задержек доступа. Каждый модуль способен независимо выполнять полный цикл обработки — от загрузки индексного дерева и значений из основной памяти до выполнения сложных графовых операций.

Структурная схема системы (продолжение)



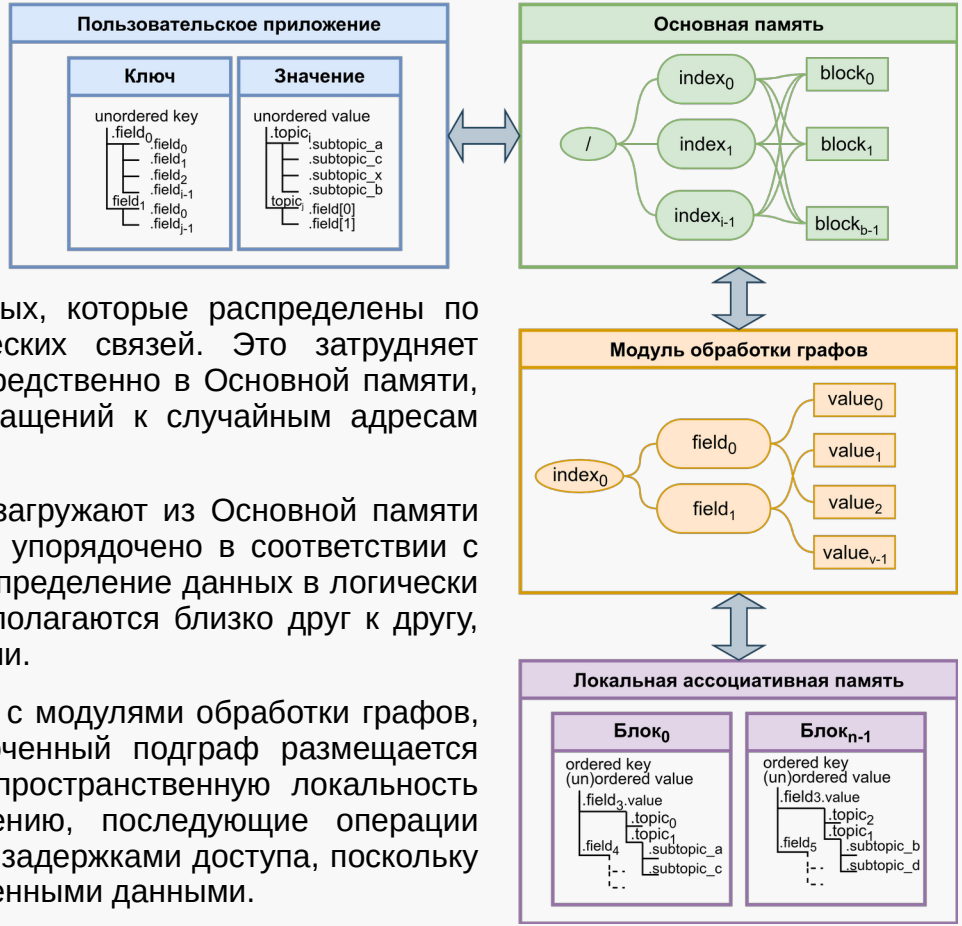
CPE (Computing Processing Element) и SPE (Structure Processing Element) представляют собой два взаимодополняющих типа многонитевых обрабатывающих элементов.

CPE ориентированы на общую вычислительную нагрузку и управляющую логику — они координируют выполнение операций, управляют потоком данных и обрабатывают стандартные вычислительные задачи. В отличие от них, SPE специализируются на эффективной обработке специфических структур данных — прежде всего, префиксных деревьев и сложных композитных ключей. Такое разделение обязанностей позволяет достичь высокой эффективности: CPE обеспечивают гибкость управления, а SPE обеспечивает высокую производительность при работе с графовыми структурами.

Менеджер локальной памяти — это специализированное аппаратное устройство, отвечающее за динамическое управление блоками локальной ассоциативной памяти внутри Модуля обработки графов. Его ключевая функция — выделение и освобождение блоков памяти для размещения фрагментов префиксных деревьев, обеспечивающее максимальное использование доступного пространства. Менеджер оптимизирует размещение данных с учетом паттернов доступа, размещая связанные узлы графа в соседних блоках памяти для минимизации задержек при обходе. Это аппаратное решение исключает необходимость программного управления памятью.

Пространственная локализация при обработке графов

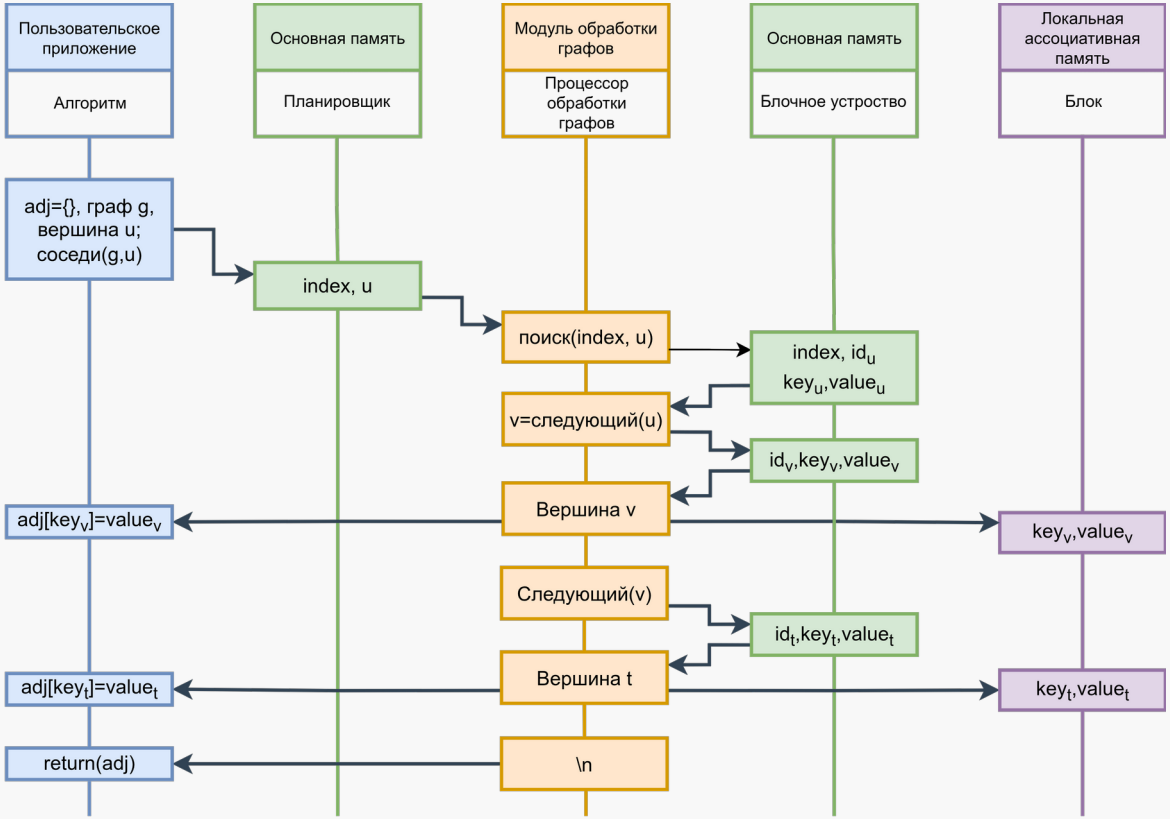
- **Пользовательское приложение** формулирует запросы на обработку графовых данных, такие как обход: определённого подграфа или поиск связанных сущностей. Пользователь работает с логической структурой графа, задавая атрибуты вершин и рёбер в виде иерархических структур.



- **Основная память** хранит весь объем графовых данных, которые распределены по различным областям хранения без учёта семантических связей. Это затрудняет обработку обработки связанных элементов графа непосредственно в Основной памяти, так как системе приходится выполнять множество обращений к случайным адресам памяти.
- **Модули обработки графов** независимо друг от друга загружают из Основной памяти необходимые вершины и ребра графа, и размещает их упорядочено в соответствии с используемым индексом. Это преобразует случайное распределение данных в логически организованную структуру, где связанные элементы располагаются близко друг к другу, что принципиально меняет характер доступа к информации.
- **Локальная ассоциативная память** расположена рядом с модулями обработки графов, что снижает латентность доступа к ней. Переупорядоченный подграф размещается компактно в виде цепочек блоков, что обеспечивает пространственную локальность данных. Благодаря такому организованному размещению, последующие операции обхода или модификации выполняются с минимальными задержками доступа, поскольку система работает с локальными, предсказуемо расположенными данными.

Модель выполнения запросов в Тераграф

- Алгоритм формирует запрос на обход графа (например, поиск соседей вершины u). Запрос содержит начальные параметры: целевой граф g и вершину u.
- Система обращается к основной памяти для получения индексной информации по вершине u. Основная память хранит распределенные индексы, позволяющие найти связанные данные.
- Модуль обработки графов выполняет итеративный обход:
  - использует номер блока id для нахождения следующей вершины (v = следующий(u));
  - последовательно обрабатывает вершины графа (v→ … →t);
  - для каждой вершины извлекает ассоциированные ключи и значения.
- Накопленные данные о смежных вершинах (adj) возвращаются пользовательскому приложению в виде готового результата.



Спасибо!

alexpopov@bmstu.ru

Москва, сентябрь 2025 г.