

Основы параллельных вычислений

Бикулов Дмитрий, к.ф.-м.н., техлид в Яндексе

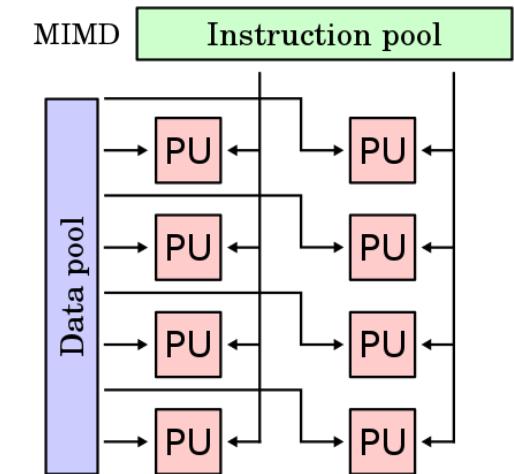
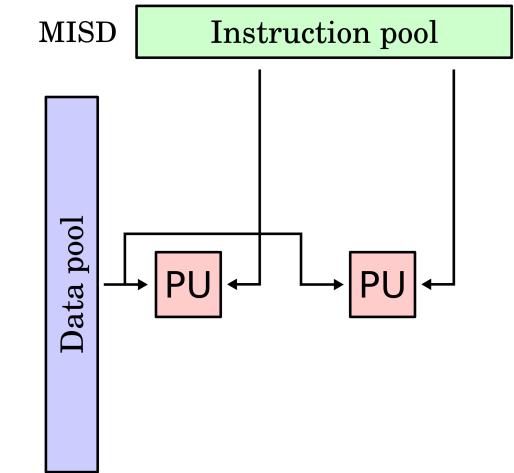
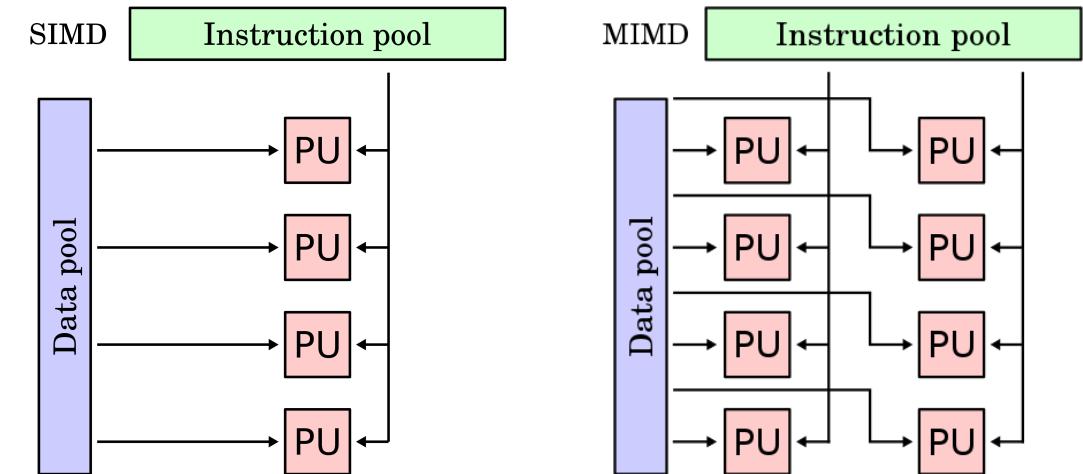
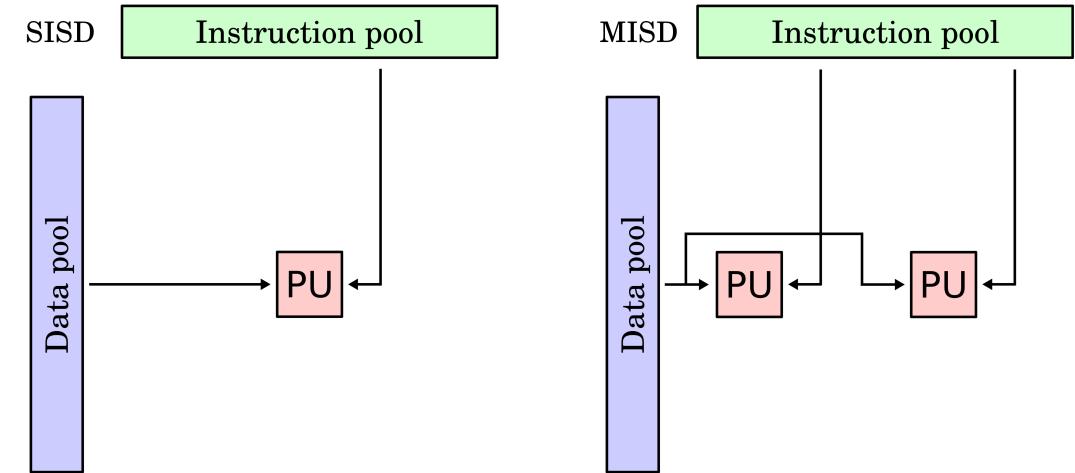
План

1. Архитектуры
2. Программирование
3. Примеры вокруг

Основные архитектуры

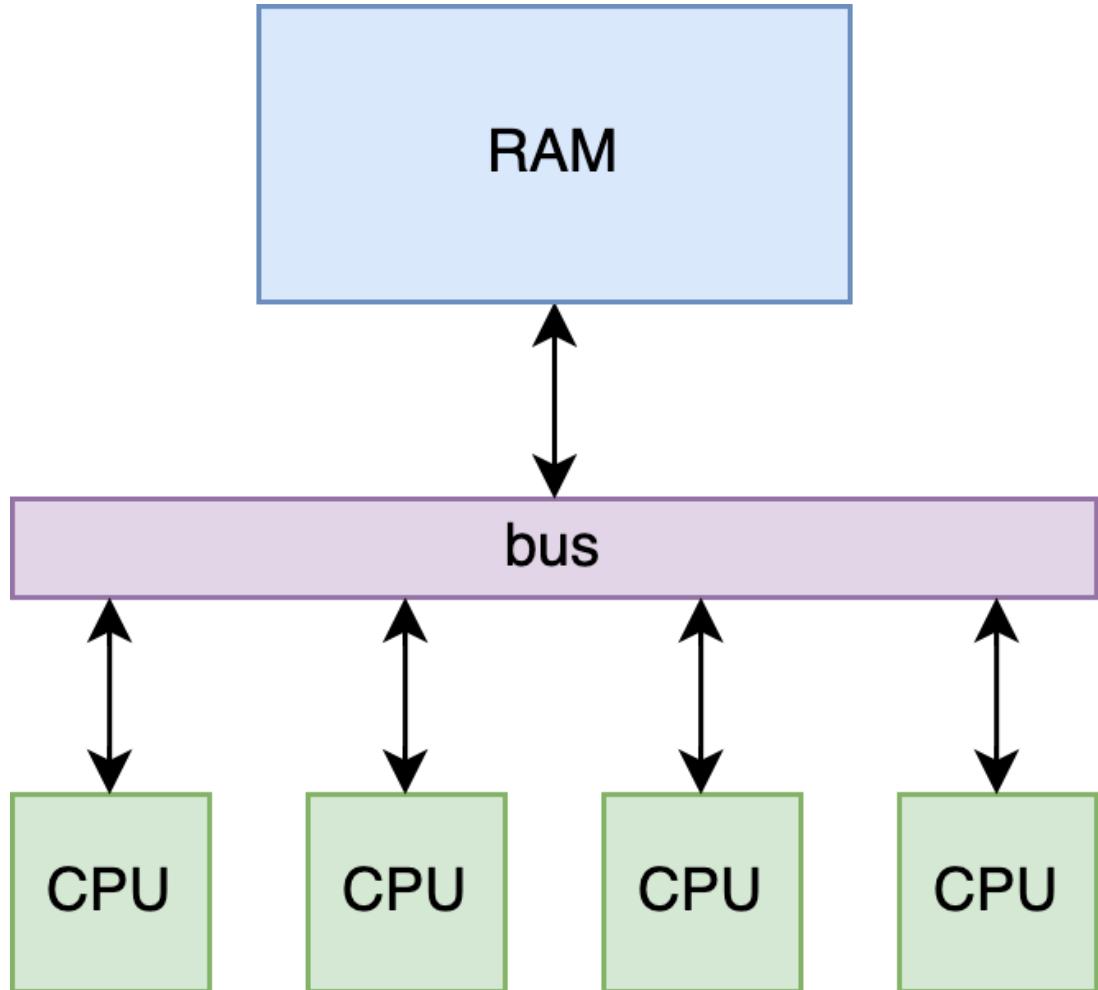
Классификация Флинна

1. Single Instruction, Single Data
2. Multiple Instruction, Single Data
3. Single Instruction, Multiple Data
4. Multiple Instruction, Multiple Data

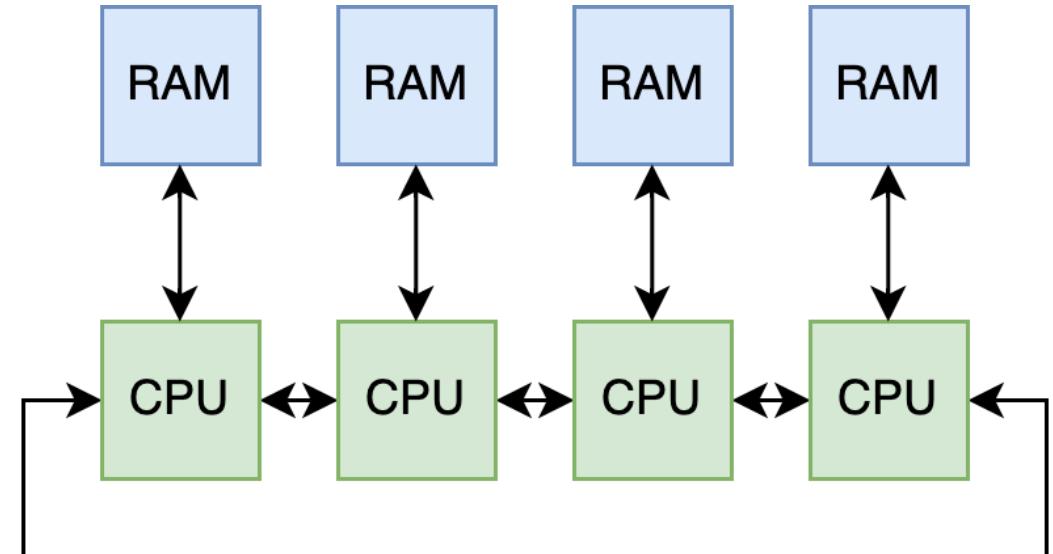


По доступу к памяти

Shared



Distributed



Другие классификации

- Классификация Фенга
- Классификация Шнайдера
- Классификация Джонсона
- [https://parallel.ru/search/node/
классификация](https://parallel.ru/search/node/классификация)



Фактически доступно

- собственный ноутбук
- видеокарта
- один или несколько серверов в облаке
- *суперкомпьютер*

Собственный ноутбук

Ограничения:

- мало ресурсов
- не масштабируется
- обжигает коленки

Плюсы:

- привычно
- быстрый запуск и отладка
- можно сразу добавить любую визуализацию

Видеокарты

Ограничения:

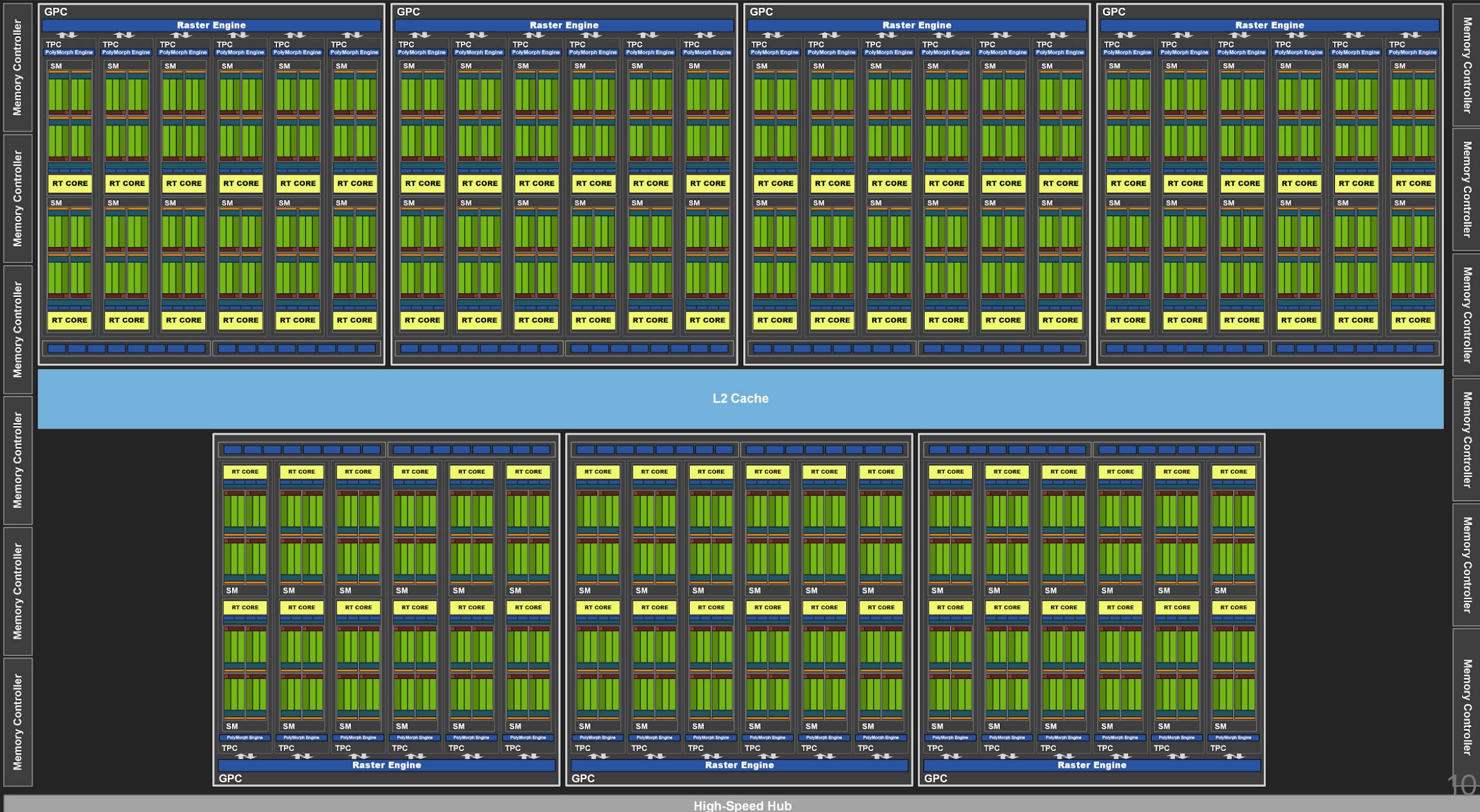
- хороши только для некоторого класса задач
- нужно писать код специально под них

Плюсы:

- можно получить x10 производительности
- энергоэффективны

[Ampere GA102 GPU Architecture whitepaper](#)

GigaThread Engine



Облака

Ограничения:

- дорого
- непривычно
- консолька

Плюсы:

- масштабируется
- есть видеокарты

[Экскурсия по ДЦ Яндекса](#)

25 302,00 ₽ в месяц ▾

[Тарифы и цены](#)

Intel Ice Lake with Nvidia T4.
100% vCPU — прерываемые
ВМ

6 681,60 ₽

Intel Ice Lake with Nvidia T4.
GPU — прерываемые ВМ

11 520,00 ₽

Intel Ice Lake with Nvidia T4.
RAM — прерываемые ВМ

6 451,20 ₽

Быстрое сетевое хранилище
(SSD)

476,40 ₽

Публичный IP-адрес

172,80 ₽

Суперкомпьютеры

- Ломоносов-2 в МГУ
- Суперкомпьютеры в Яндексе
- В Сбере

[wiki:Top500](#)



картинка с сайта [parallel.ru](#)

Штош

- у вас модель считается 10 минут на ноутбуке? Вам и так хорошо, оставайтесь на нём
- у вас гидродинамика, нейронки, конечные элементы, что-угодно-много-одинакового и в пространстве? Вам нужен **GPU**
- не лезет на один компьютер или GPU? **Облака** в помощь
- не лезет на один компьютер и в облаках тормозит? Остаётся **суперкомпьютер**



Параллельность в коде

OpenMP

wiki:OpenMP, openmp.cpp в примерах

```
// src/openmp.cpp
std::vector<double> a, b, c;
const auto N = 1'000'000;

a.resize(N); b.resize(N); c.resize(N);

omp_set_dynamic(0);      // запретить библиотеке openmp менять число потоков во время исполнения
omp_set_num_threads(10); // установить число потоков в 10

std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();

#pragma omp parallel for
for (size_t i = 0; i < N; ++i) {
    for (size_t j = 0; j < 1'000; ++j) {
        a[i] += std::log(j + 1 + i * 1.0);
        b[i] += std::sqrt(std::exp(i * 2.0)) + j;
        c[i] += std::sqrt(std::pow(a[i] + b[i], 3)) + j;
    }
}

std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

std::cout << std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count() << "[μs]" << std::endl;
return 0;
```

MPI

wiki:MPI, Лекция, Документация

```
// src/mpiepx.cpp
if (processRank != 0) {
    for (size_t i = 0; i < N / clusterSize; ++i) {
        for (size_t j = 0; j < 1'000; ++j) {
            a[i] += std::log(j + 1 + i * 1.0);
            b[i] += std::sqrt(std::exp(i * 2.0)) + j;
            c[i] += std::sqrt(std::pow(a[i] + b[i], 3)) + j;
        }
    }

    MPI_Send((const void*)&a[0], N, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD);
    MPI_Send((const void*)&b[0], N, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD);
    MPI_Send((const void*)&c[0], N, MPI_DOUBLE, 0, 3, MPI_COMM_WORLD);
} else {

    const auto chunkSize = N / (clusterSize - 1);

    MPI_Status mpiStatus;
    for (int i = 1; i < clusterSize; ++i) {
        MPI_Recv(&a[(i-1)*chunkSize], chunkSize, MPI_DOUBLE, i, 1, MPI_COMM_WORLD, &mpiStatus);
        MPI_Recv(&b[(i-1)*chunkSize], chunkSize, MPI_DOUBLE, i, 2, MPI_COMM_WORLD, &mpiStatus);
        MPI_Recv(&c[(i-1)*chunkSize], chunkSize, MPI_DOUBLE, i, 3, MPI_COMM_WORLD, &mpiStatus);
    }

    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count() << "[μs]" << std::endl;
}
```

ПОТОКИ

```
// src/thread.cpp

void long_task(std::vector<double>& a) {
    for (auto& aa : a) {
        aa = 3 * 3;
    }
}

int main(int, char**)
{
    std::vector<double> a;
    a.resize(100);

    std::thread th(long_task, std::ref(a));
    th.join();
    for (auto i = 0; i < 100; ++i) {
        std::cout << a[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Python: потоки, процессы и корутины

- в Python есть GIL [wiki:GIL](#)
- [the GIL and its effects on Python multithreading](#)
- multiprocessing
- asyncio

CUDA

```
__global__ void vectorAdd(const float *A, const float *B, float *C, int numElements) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements) {
        C[i] = A[i] + B[i] + 0.0f;
    }
}

int main(void) {
    // ...
    float *h_A = (float *)malloc(size);
    // ...

    for (int i = 0; i < numElements; ++i) {
        h_A[i] = rand() / (float)RAND_MAX;
        // ...
    }

    float *d_A = NULL;
    cudaMalloc((void **)&d_A, size);

    // ...
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    // ...

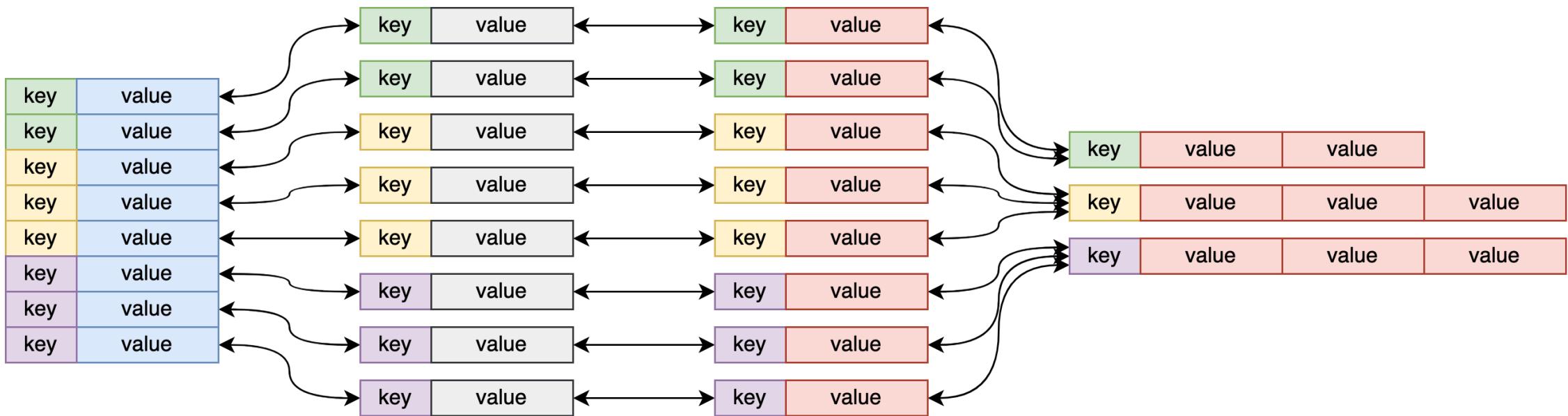
    int threadsPerBlock = 256;
    int blocksPerGrid = (numElements + threadsPerBlock - 1) / threadsPerBlock;

    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, numElements);

    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

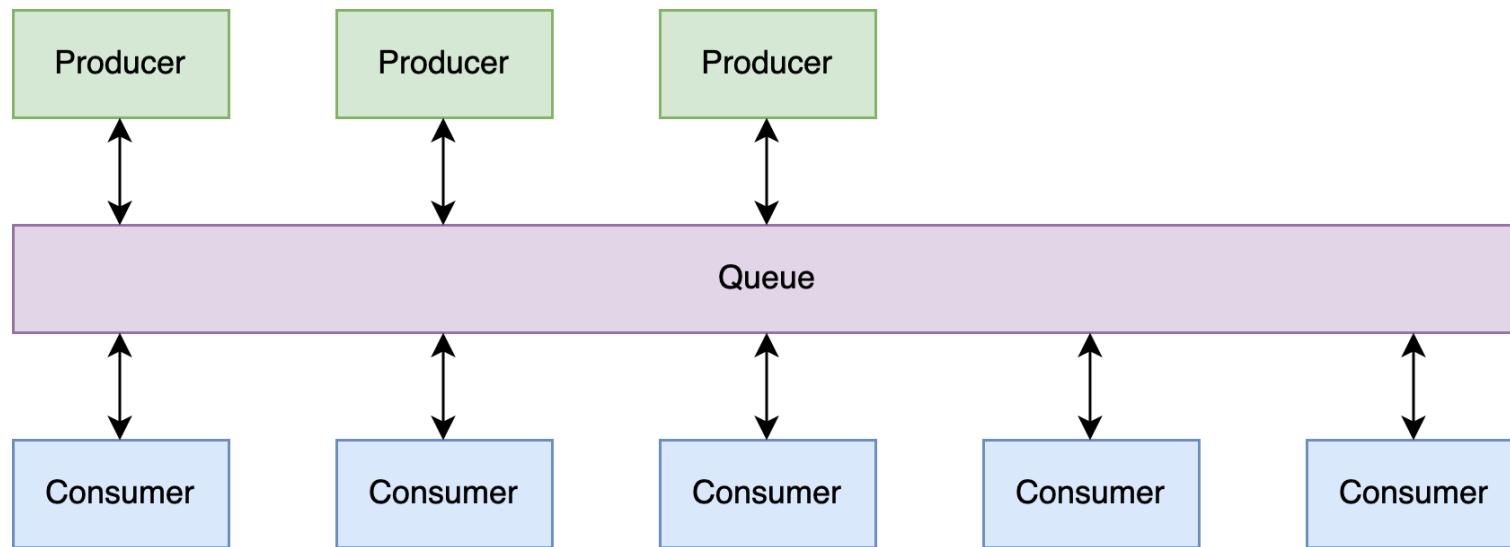
Полезно знать

MapReduce



Подлодка #258 – Распределенные вычисления

Брокер сообщений

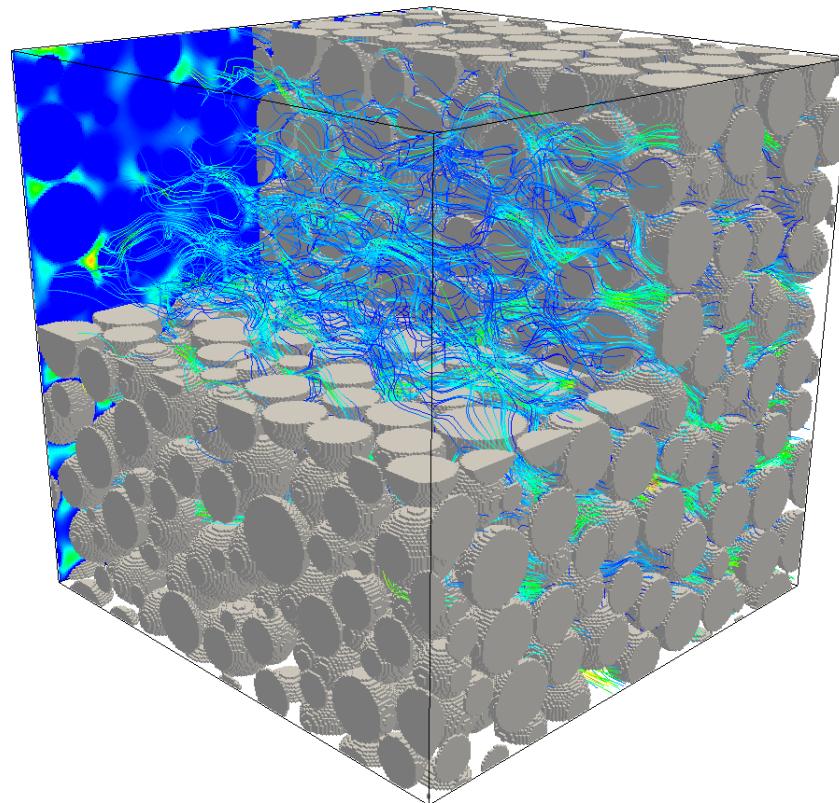
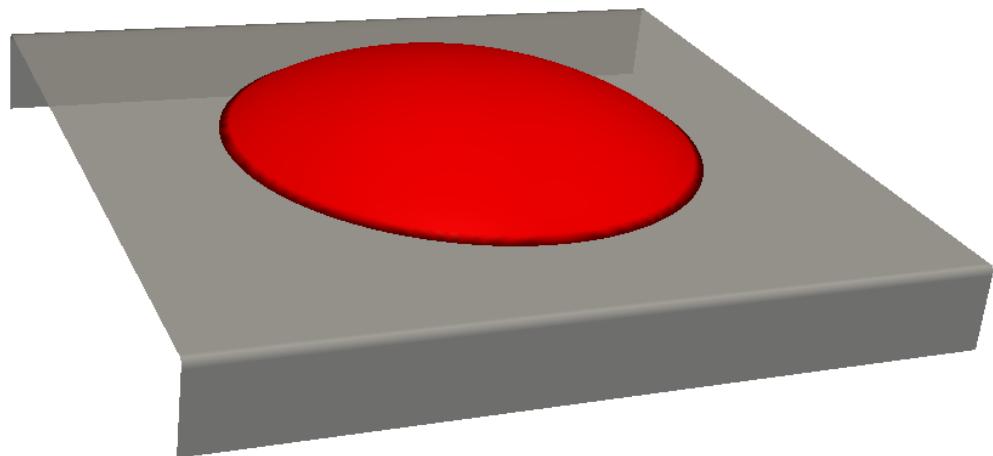


- [wiki:Брокер сообщений](#)
- [RabbitMQ](#)
- [Kafka](#)

Podlodka #277 - Менеджеры очередей

Примеры параллельных вычислений вокруг нас

Гидродинамика: Lattice Boltzmann Method



Нейросети

[github:Stable Diffusion](#)

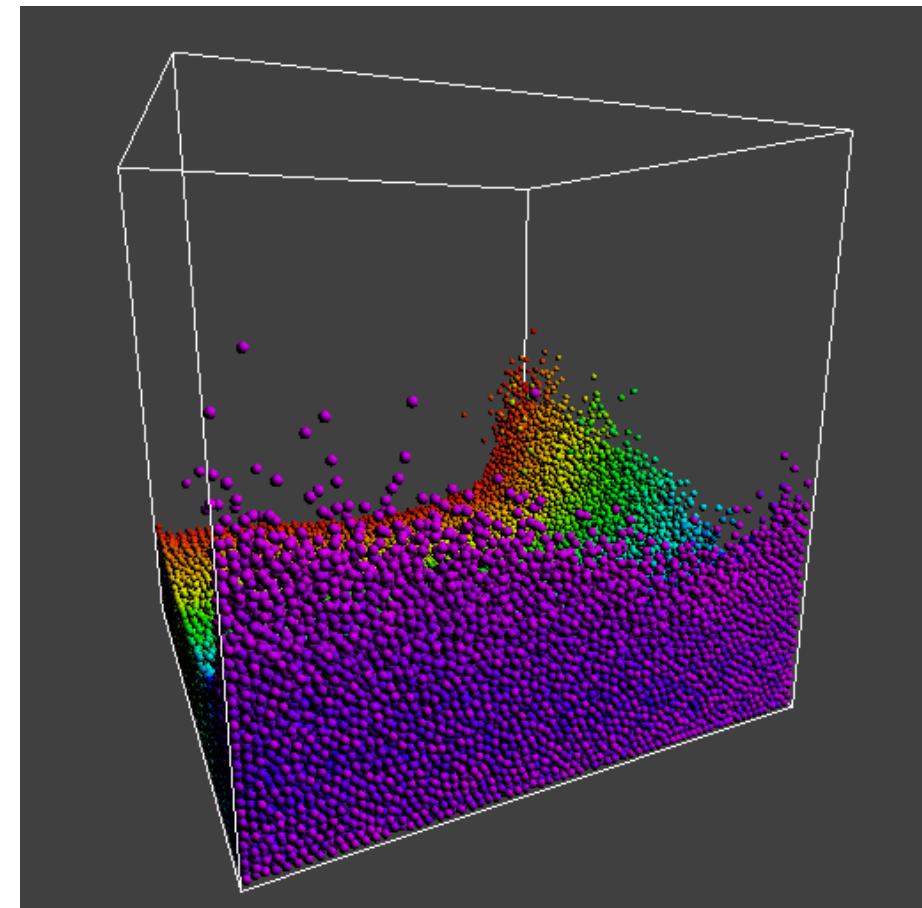


Картина с [stability.ai](#)

Моделирование частиц

Картишка из [примеров CUDA](#)

почитайте код, там много
оптимизаций :)



Спасибо! Вопросы?

Классификация Флинна, По доступу к памяти, Другие классификации,
Фактически доступно, Собственный ноутбук, Видеокарты, Облака,
Суперкомпьютеры, Штош

OpenMP, MPI, Потоки, Python: потоки, процессы и корутины, CUDA,
MapReduce, Брокер сообщений

Гидродинамика: Lattice Boltzmann Method, Нейросети, Моделирование
частиц