Семинар 17
Технология CUDA
Оптимизация доступа к памяти

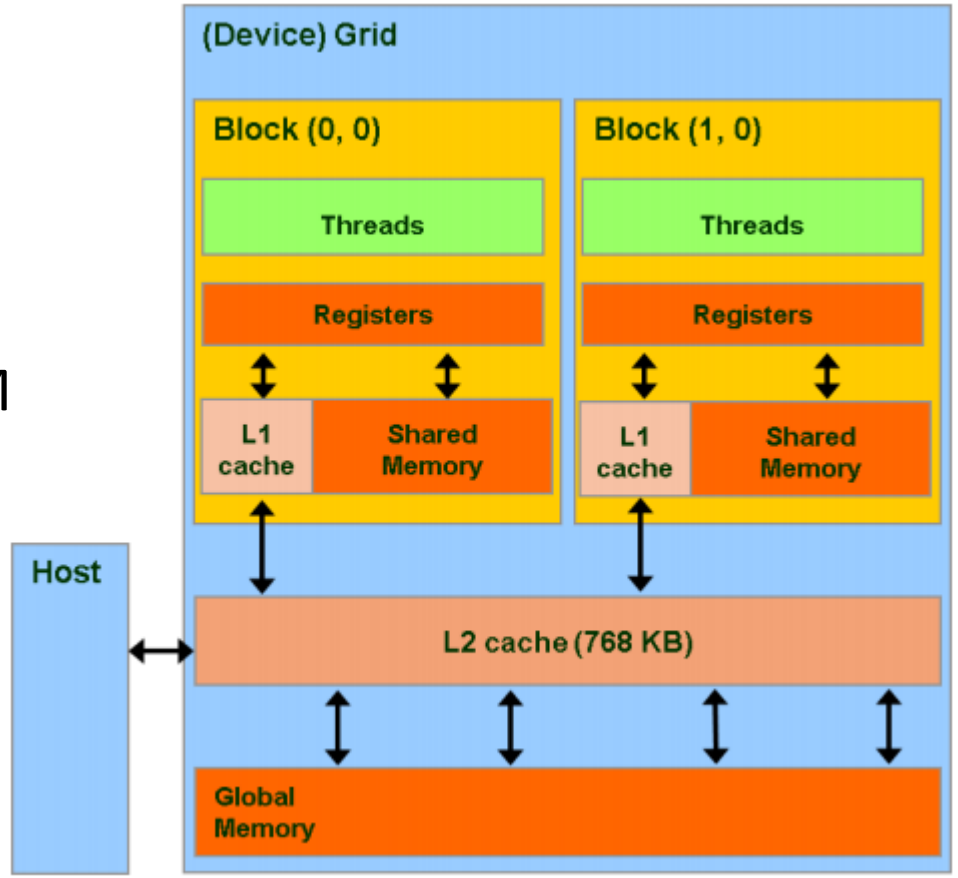
Михаил Курносов

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2015

Глобальная память (Fermi/Kepler)

- **Compute Capability >= 2.0**
- Длина строки кеш-памяти L1: 128 байт
- Длина строки кеш-памяти L2: 32 байта
- **Операция чтения данных** (load)
  - *Default*: Поиск в L1, затем в L2, далее в GMEM
  - *Альтернатива* (–Xptxas –dlcm=cg): Поиск L2, далее в GMEM
- **Операция записи данных** (store)
  - Invalidate L1, write-back L2

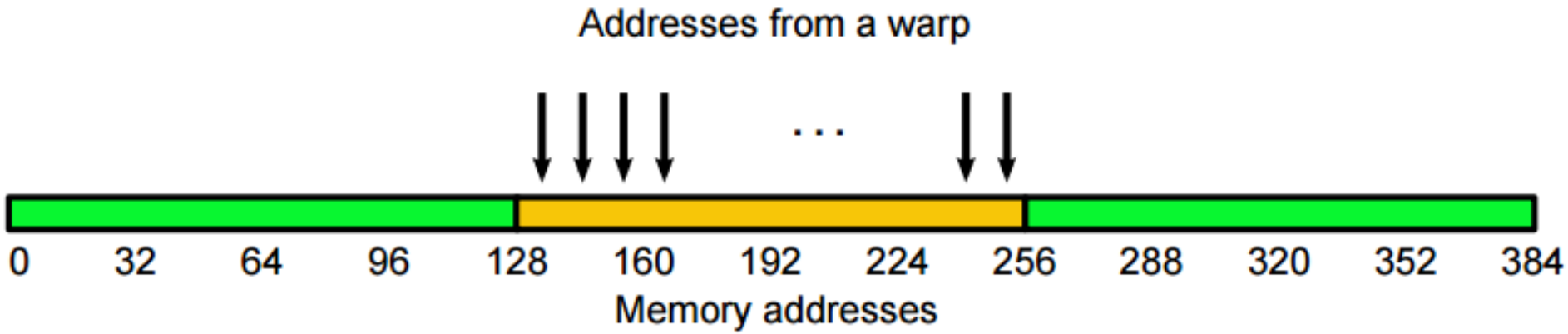


Глобальная память

- Операция обращения к памяти (load/store) выполняется всеми потоками группы (warp)
- Потоки предоставляют адреса
- GPU:
  1. Определяет сколько запросов к памяти необходимо выполнить потокам warp’а
  2. Группирует запросы (coalesce) в один (1x latency vs. 32x latency)

- Поддерживаемые размеры слов: 1, 2, 4, 8 и 16 байт
- Адрес слова должен быть выровнен на его размер:
  - float – на границу в 4 байта
  - double – на границу в 8 байт
- cudaMalloc() – адрес корректно выравнен для любого предопределенного типа (по меньшей мере на границу в 256 байт)

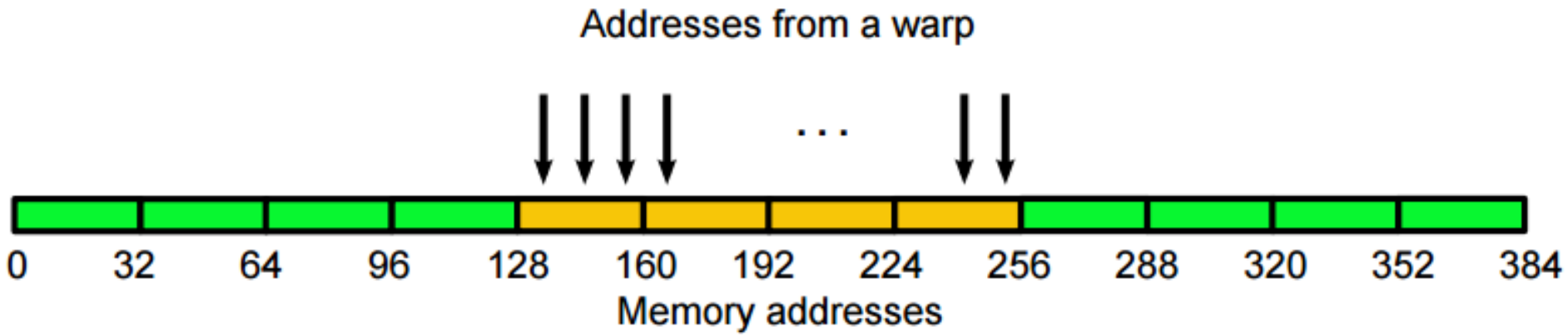
Caching load: aligned data



- Потоки группы (warp) запросили 32 смежных корректно выравненных слова по 4 байта
- Всего группе требуется: 128 байт
- Данные передаются за один запрос из L1

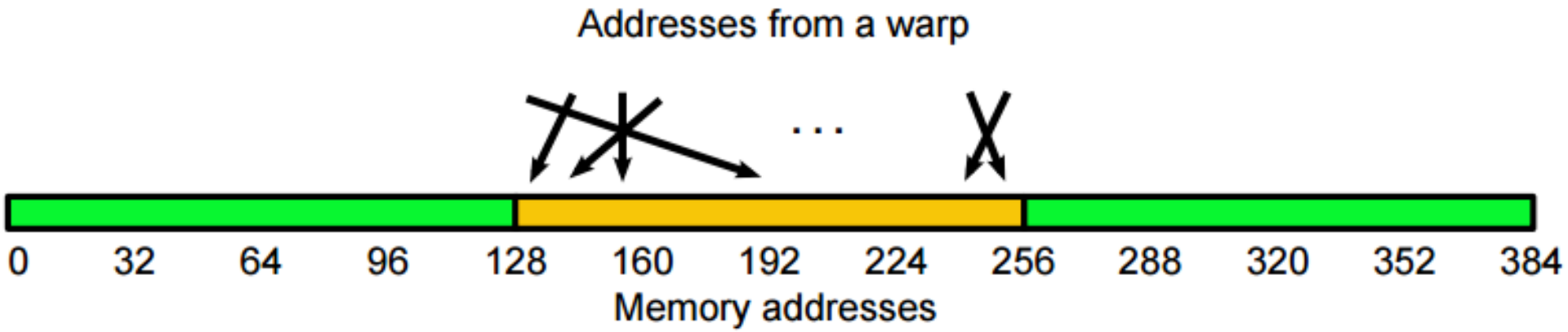
http://gpgpu.org/wp/wp-content/uploads/2013/09/06-opti-memory.pdf

Non-caching load: aligned data



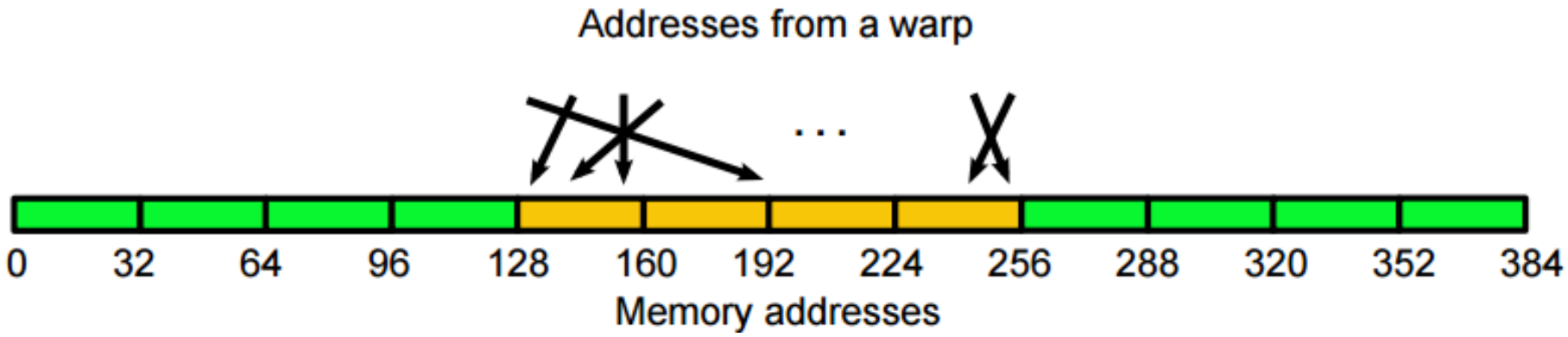
- Потоки группы (warp) запросили 32 смежных корректно выравненных слова по 4 байта
- Всего группе требуется: 128 байт
- Адреса попали в 4 сегмента (четыре L2-строки)
- Данные передаются за один запрос

Caching load: aligned data, permuted



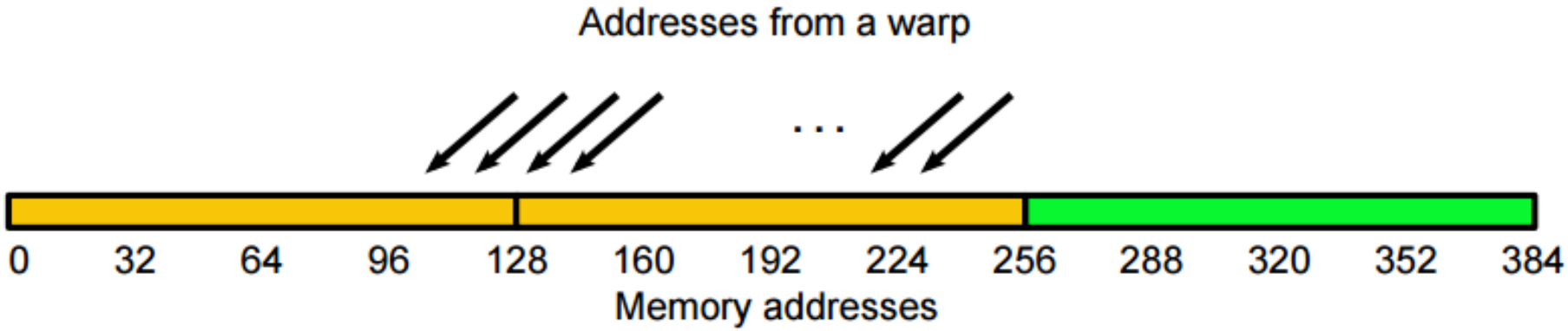
- Потоки группы (warp) запросили 32 несмежных выравненных слова по 4 байта
- Всего группе требуется: 128 байт
- Адреса попали в 1 строку L1
- Данные передаются за один запрос

Non-caching load: aligned data, permuted



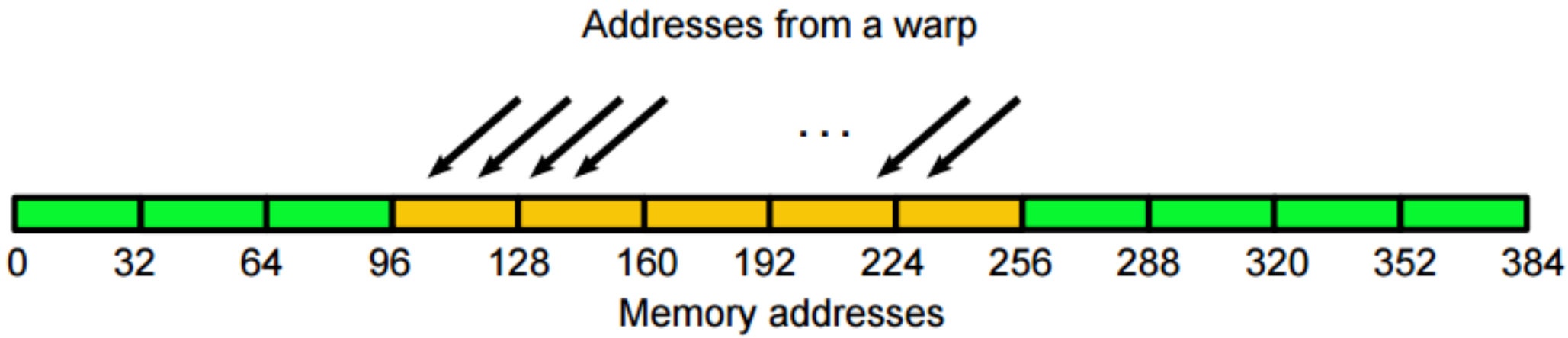
- Потоки группы (warp) запросили 32 несмежных выравненных слова по 4 байта
- Всего группе требуется: 128 байт
- Данные передаются за один запрос

Caching load: misaligned data



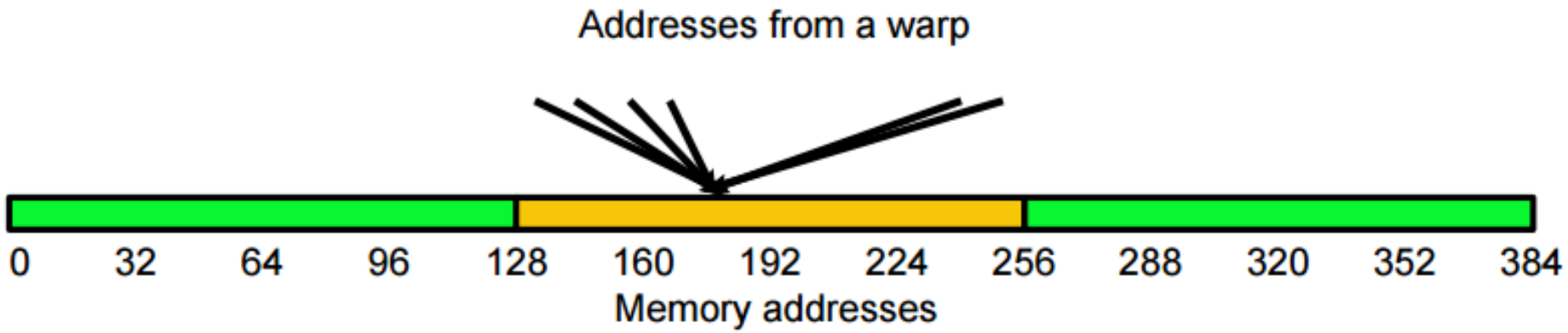
- Потоки группы (warp) запросили 32 смежных **не выравненных** слова по 4 байта
- Всего группе требуется: 128 байт
- Передается 256 байт – две строки L1

Non-caching load: misaligned data



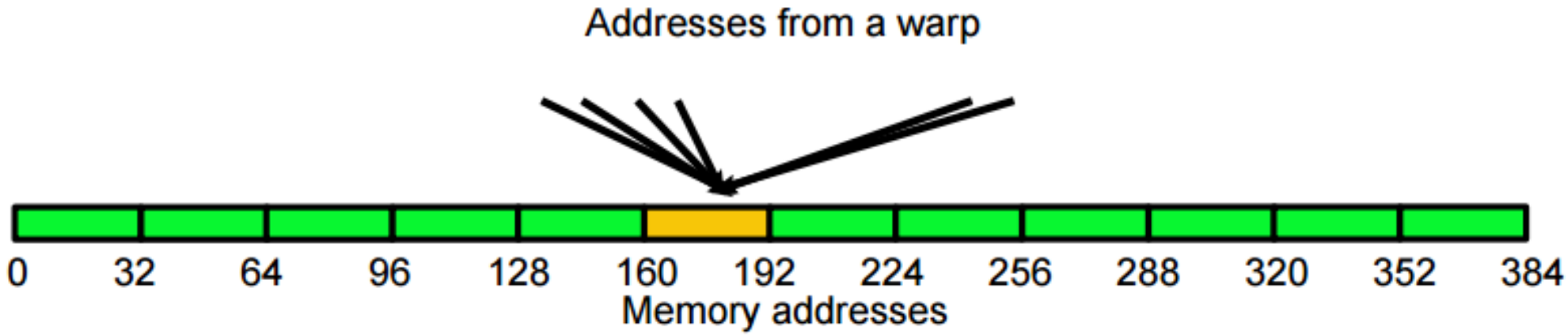
- Потоки группы (warp) запросили 32 смежных **не выравненных** слова по 4 байта
- Всего группе требуется: 128 байт
- Данные попали в 5 сегментов (5 строк L2)
- Передается 160 байт – 5 строк L2

Caching load: same 4 byte



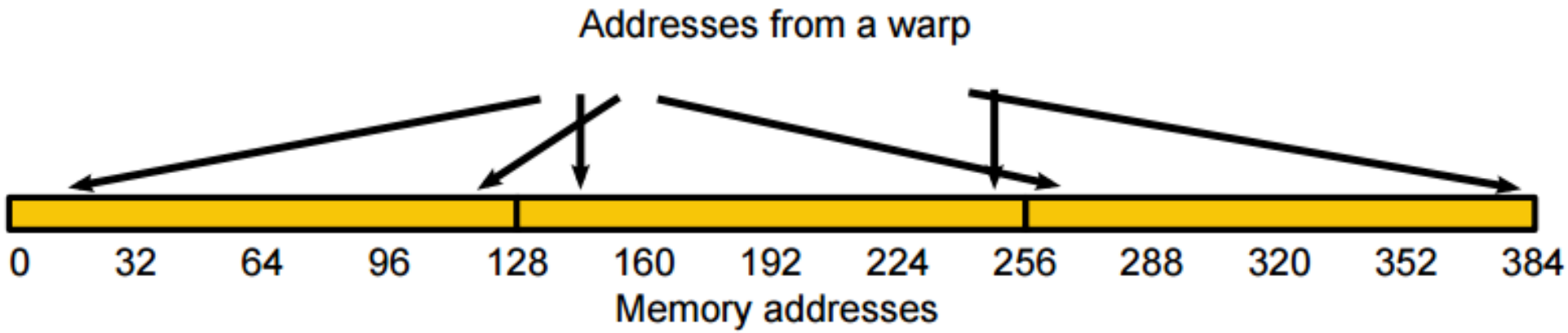
- Потоки группы (warp) запросили одно и тоже 4-х байтовое слово
- Группе требуется 4 байта
- Данные попали в 1 строку L1 (128 байт), передается 128 байт

Non-caching load: same 4 byte



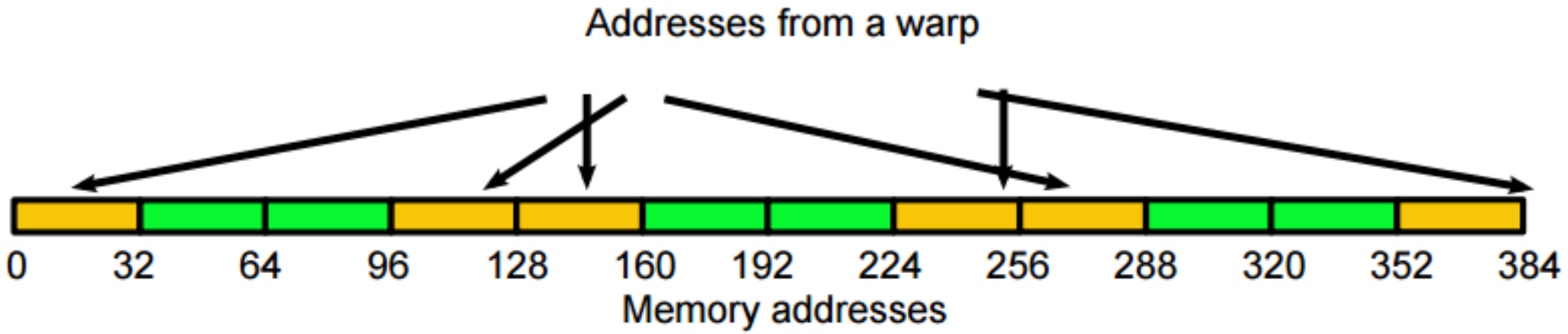
- Потоки группы (warp) запросили одно и тоже 4-х байтовое слово
- Группе требуется 4 байта
- Данные попали в 1 сегмент L2 (32 байта), передается 32 байта

Caching load: scattered



- Потоки группы (warp) запросили 32 слова с шагом > 128 байт
- Группе требуется 128 байт
- Данные попали в *N* строк L1 (128 байт), передается 128*N* байт

Non-caching load: scattered



- Потоки группы (warp) запросили 32 слова с шагом > 128 байт
- Группе требуется 128 байт
- Данные попали в *N* сегментов L2 (32 байта), передается 32*N* байт

Пример 1

```
__global__ void saxpy_1(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N)
        y[i] = y[i] + a * x[i];
}
```

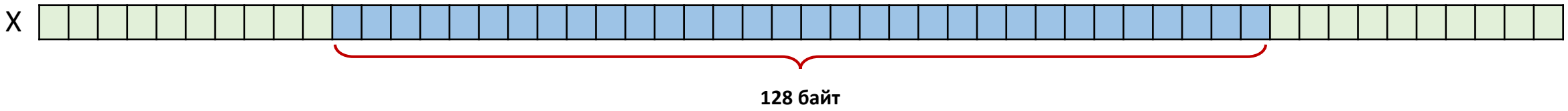
- **Load y[i], Load x[i], Store y[i]**
- **Количество строк кеш-памяти?**

Пример 1

```
__global__ void saxpy_1(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N)
        y[i] = y[i] + a * x[i];
}
```

- **Количество строк кеш-памяти?**
  - **Потоки обращаются по последовательным адресам**
  - Одна загрузка 128 байтовой строки для y
  - Одна загрузка 128 байтовой строки для x
  - 128 байт на запись y

Пропускная способность:
Bus utilization / bw efficiency =
128 / 128 = 100 %



Пример 2

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N)
        if (i % 2 == 0) y[i] = y[i] + a * x[i];
}
```

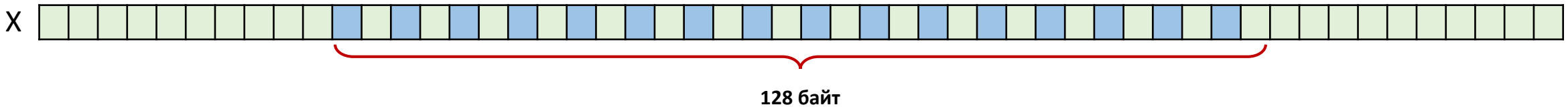
- **Только четные индексы: Load y[i], Load x[i], Store y[i]**
- **Количество строк кеш-памяти?**

Пример 2

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N)
        if (i % 2 == 0) y[i] = y[i] + a * x[i];
}
```

- **Количество строк кеш-памяти?**
  - **Потоки обращаются по последовательным адресам**
  - Одна загрузка 128 байтовой строки для y
  - Одна загрузка 128 байтовой строки для x
  - 128 байт на запись y

Пропускная способность используется менее эффективно:
Bus utilization / bw efficiency =
128 / 64 = 50 %



Пример 3

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N)
        if (i % 2 == 0) y[i] = y[i] + a * x[i];
        else y[i] = y[i] - a * x[i];

}
```

- **Количество строк кеш-памяти?**

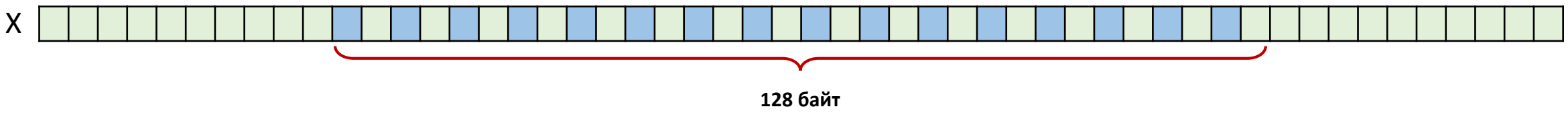
Пример 3

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N)
        if (i % 2 == 0) y[i] = y[i] + a * x[i];
        else y[i] = y[i] - a * x[i];

}
```

- **Количество строк кеш-памяти?**
- Первая ветвь (16 потоков): загрузка 2-х строк по 128 байт
- Вторая ветвь (16 потоков): данные читаются из кеша

Пропускная способность используется менее эффективно: Bus utilization / bw efficiency = 128 / 64 = 50 %



Пример 4

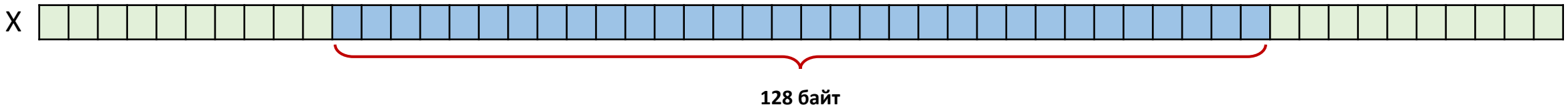
```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N) {
        float yy = y[i]; float xx = x[i];       // Load to regs
        if (i % 2 == 0) yy = yy + a * xx;
        else yy = yy - a * xx;
        y[i] = yy;                              // Store
    }
}
```

Пример 4

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N) {
        float yy = y[i]; float xx = x[i];      // Load to regs
        if (i % 2 == 0) yy = yy + a * xx;
        else yy = yy - a * xx;
        y[i] = yy;                             // Store
    }
}
```

- **Потоки обращаются по последовательным адресам**
- Две загрузки 128 байт, одна запись 128 байт
- 128 байт на запись y

Пропускная способность:
Bus utilization / bw efficiency =
128 / 128 = 100 %



Пример 5

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N / 2)
        y[i] = y[i] + a * x[i]
    else if (i < N)
        y[i] = y[i] i a * x[i]
}
```

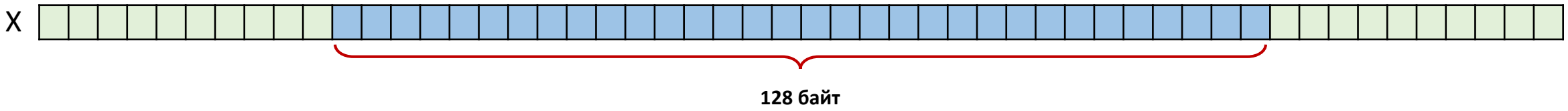
Пример 5

```
__global__ void saxpy_2(float* x, float* y, float a, unsigned long N)
{
    unsigned long i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < N / 2)
        y[i] = y[i] + a * x[i]
    else if (i < N)
        y[i] = y[i] i a * x[i]
}
```

- Все потоки группы (warp) попадают в одну из ветвей:
  две загрузки 128 байт, запись 128 байт
- Потоки группы пересекают границу (N / 2):
  - Первая ветвь (16 потоков): загрузка 2-х строк по 128 байт
  - Вторая ветвь (16 потоков): данные читаются из кеша

Пропускная способность:
Bus utilization / bw efficiency =
128 / 128 = 100 %

Пропускная способность используется менее эффективно:
Bus utilization / bw efficiency =
128 / 64 = 50 %



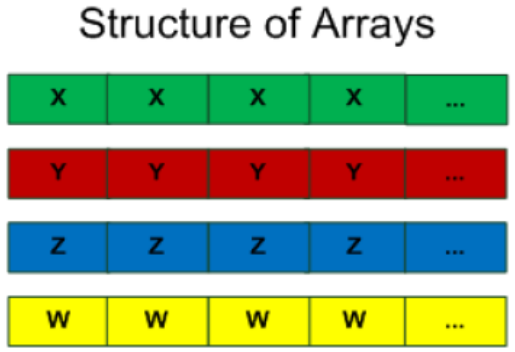
Оптимизация структур

Array of Structures (AOS) vs. Structure of Arrays (SOA)

```
struct point {float x; float y; float z; float w;};
struct point *points;
cudaMalloc(&(void**)points, N * sizeof(*points));
```

VS.

```
struct points {float *x; float *y; float *z; float *w;};
struct points points;
cudaMalloc(&(void**)points.x, N * sizeof(float));
cudaMalloc(&(void**)points.y, N * sizeof(float));
cudaMalloc(&(void**)points.z, N * sizeof(float));
cudaMalloc(&(void**)points.w, N * sizeof(float));
```



Оптимизация структур

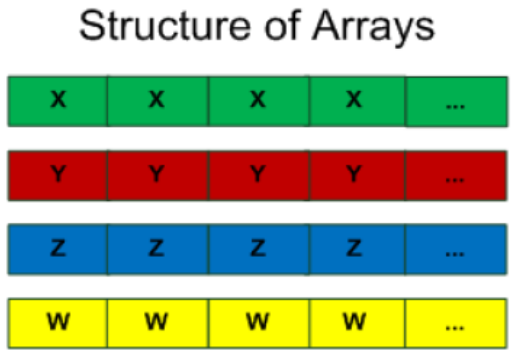
Array of Structures (AOS) vs. Structure of Arrays (SOA)

```
struct point {float x; float y; float z; float w;};
struct point *points;
cudaMalloc(&(void**)points, N * sizeof(*points));
```

Обращения по несмежным адресам,
больше обращений к памяти (строкам)

vs.

```
struct points {float *x; float *y; float *z; float *w;};
struct points points;
cudaMalloc(&(void**)points.x, N * sizeof(float));
cudaMalloc(&(void**)points.y, N * sizeof(float));
cudaMalloc(&(void**)points.z, N * sizeof(float));
cudaMalloc(&(void**)points.w, N * sizeof(float));
```



**Обращения по смежным адресам,
меньше обращений к памяти (строкам)**

Задача *N*-тел (NBody)

```
const float eps = 0.0001f;
const float dt = 0.01f;
const int N = 128 * 1024;

#define coord struct body
struct body {
    float x;
    float y;
    float z;
};

__global__ void integrate(coord *new_p, coord *new_v, coord *p, coord *v,
                          int n, float dt)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    // Каждый поток вычисляет силу, действующую на тело index
    coord body_pos = p[index];
    coord body_vel = v[index];
    coord f;
    f.x = 0;
    f.y = 0;
    f.z = 0;
```

Задача *N*-тел (NBody)

```
    for (int i = 0; i < n; i++) {
        coord pi = p[i];
        coord r;
        // Вектор от тела p[i] к телу body_pos
        r.x = pi.x - body_pos.x;
        r.y = pi.y - body_pos.y;
        r.z = pi.z - body_pos.z;

        float invDist = 1.0 / sqrtf(r.x * r.x + r.y * r.y + r.z * r.z + eps * eps);
        float s = invDist * invDist * invDist;
        // Корректируем силу - вносим вклад тела i
        f.x += r.x * s;
        f.y += r.y * s;
        f.z += r.z * s;
    }

    // Корректируем параметры тела: скорость, положение
    body_vel.x += f.x * dt;
    body_vel.y += f.y * dt;
    body_vel.z += f.z * dt;
    body_pos.x += body_vel.x * dt;
    body_pos.y += body_vel.y * dt;
    body_pos.z += body_vel.z * dt;

    new_p[index] = body_pos;
    new_v[index] = body_vel;
}
```

Задача *N*-тел (NBody)

```
int main()
{
    double tgpu = 0, tmem = 0;
    size_t size = sizeof(coord) * N;
    coord *p = (coord *)malloc(size);  coord *v = (coord *)malloc(size);
    coord *d_p[2] = {NULL, NULL};  coord *d_v[2] = {NULL, NULL};
    init_rand(p, N); init_rand(v, N);

    tmem = -wtime();
    cudaMalloc((void **)&d_p[0], size);
    cudaMalloc((void **)&d_p[1], size);
    cudaMalloc((void **)&d_v[0], size);
    cudaMalloc((void **)&d_v[1], size);
    cudaMemcpy(d_p[0], p, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_v[0], v, size, cudaMemcpyHostToDevice);
    tmem += wtime();

    tgpu = -wtime();
    int threadsPerBlock = 1024;
    dim3 block(threadsPerBlock);
    dim3 grid((N + threadsPerBlock - 1) / threadsPerBlock);
    int index = 0;
    for (int i = 0; i < 2; i++, index ^= 1)
        integrate<<<grid, block>>>(d_p[index ^ 1], d_v[index ^ 1], d_p[index], d_v[index], N, dt);
    cudaDeviceSynchronize();
    tgpu += wtime();
```

Задача *N*-тел (NBody)

```
    tmem -= wtime();
    cudaMemcpy(p, d_p[index], size, cudaMemcpyDeviceToHost);
    cudaMemcpy(v, d_v[index], size, cudaMemcpyDeviceToHost);
    tmem += wtime();

    printf("sizeof(coord) = %d\n", sizeof(coord));
    printf("GPU version (sec.): %.6f\n", tgpu);
    printf("Memory ops. (sec.): %.6f\n", tmem);
    printf(" Total time (sec.): %.6f\n", tgpu + tmem);

    cudaFree(d_p[0]);
    cudaFree(d_p[1]);
    cudaFree(d_v[0]);
    cudaFree(d_v[1]);
    free(p);
    free(v);
    cudaDeviceReset();
    return 0;
}
```

```
sizeof(coord) = 12
GPU version (sec.): 2.382872
Memory ops. (sec.): 0.252231
 Total time (sec.): 2.635103
```

Задача *N*-тел (NBody): float3

```
const float eps = 0.0001f;
const float dt = 0.01f;
const int N = 128 * 1024;

#define coord float3

__global__ void integrate(coord *new_p, coord *new_v, coord *p, coord *v,
                          int n, float dt)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    // Каждый поток вычисляет силу, действующую на тело index
    coord body_pos = p[index];
    coord body_vel = v[index];
    coord f;
    f.x = 0;
    f.y = 0;
    f.z = 0;
```

Задача *N*-тел (NBody): padding

```
const float eps = 0.0001f;
const float dt = 0.01f;
const int N = 128 * 1024;

#define coord struct body
struct body {
    float x;
    float y;
    float z;
    float __padding;
};

__global__ void integrate(coord *new_p, coord *new_v, coord *p, coord *v,
                          int n, float dt)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    // Каждый поток вычисляет силу, действующую на тело index
    coord body_pos = p[index];
    coord body_vel = v[index];
    coord f;
    f.x = 0;
    f.y = 0;
    f.z = 0;
```

```
sizeof(coord) = 16
GPU version (sec.): 1.946044
Memory ops. (sec.): 0.228796
 Total time (sec.): 2.174840
```

Задача *N*-тел (NBody): float4

```
const float eps = 0.0001f;
const float dt = 0.01f;
const int N = 128 * 1024;

#define coord float4

__global__ void integrate(coord *new_p, coord *new_v, coord *p, coord *v,
                          int n, float dt)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    // Каждый поток вычисляет силу, действующую на тело index
    coord body_pos = p[index];
    coord body_vel = v[index];
    coord f;
    f.x = 0;
    f.y = 0;
    f.z = 0;
```

```
sizeof(coord) = 16
GPU version (sec.): 1.720787
Memory ops. (sec.): 0.226160
 Total time (sec.): 1.946947
```

Задача *N*-тел (NBody): float4 + shared memory

```
__global__ void integrate(coord *new_p, coord *new_v, coord *p, coord *v, int n, float dt)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    coord body_pos = p[index];
    coord body_vel = v[index];
    coord f;
    f.x = 0;
    f.y = 0;
    f.z = 0;

    __shared__ coord sp[block_size];

    // Assert: n % block_size == 0
    for (int ind = 0; ind < n; ind += block_size) {
        sp[threadIdx.x] = p[ind + threadIdx.x];

        __syncthreads();
```

Задача *N*-тел (NBody): float4 + shared memory

```
        for (int i = 0; i < block_size; i++) {
            // Vector from p[i] to body
            coord r;
            r.x = sp[i].x - body_pos.x;
            r.y = sp[i].y - body_pos.y;
            r.z = sp[i].z - body_pos.z;

            float invDist = 1.0 / sqrtf(r.x * r.x + r.y * r.y + r.z * r.z + eps * eps);
            float s = invDist * invDist * invDist;
            // Add force of body i
            f.x += r.x * s;
            f.y += r.y * s;
            f.z += r.z * s;
        }
        __syncthreads();
    }
    // Correct velocity
    body_vel.x += f.x * dt;     body_vel.y += f.y * dt;    body_vel.z += f.z * dt;
    body_pos.x += body_vel.x * dt;  body_pos.y += body_vel.y * dt;  body_pos.z += body_vel.z * dt;

    new_p[index] = body_pos;  new_v[index] = body_vel;
}
```

```
GPU version (sec.): 2.055868
Memory ops. (sec.): 0.249816
 Total time (sec.): 2.305684
```