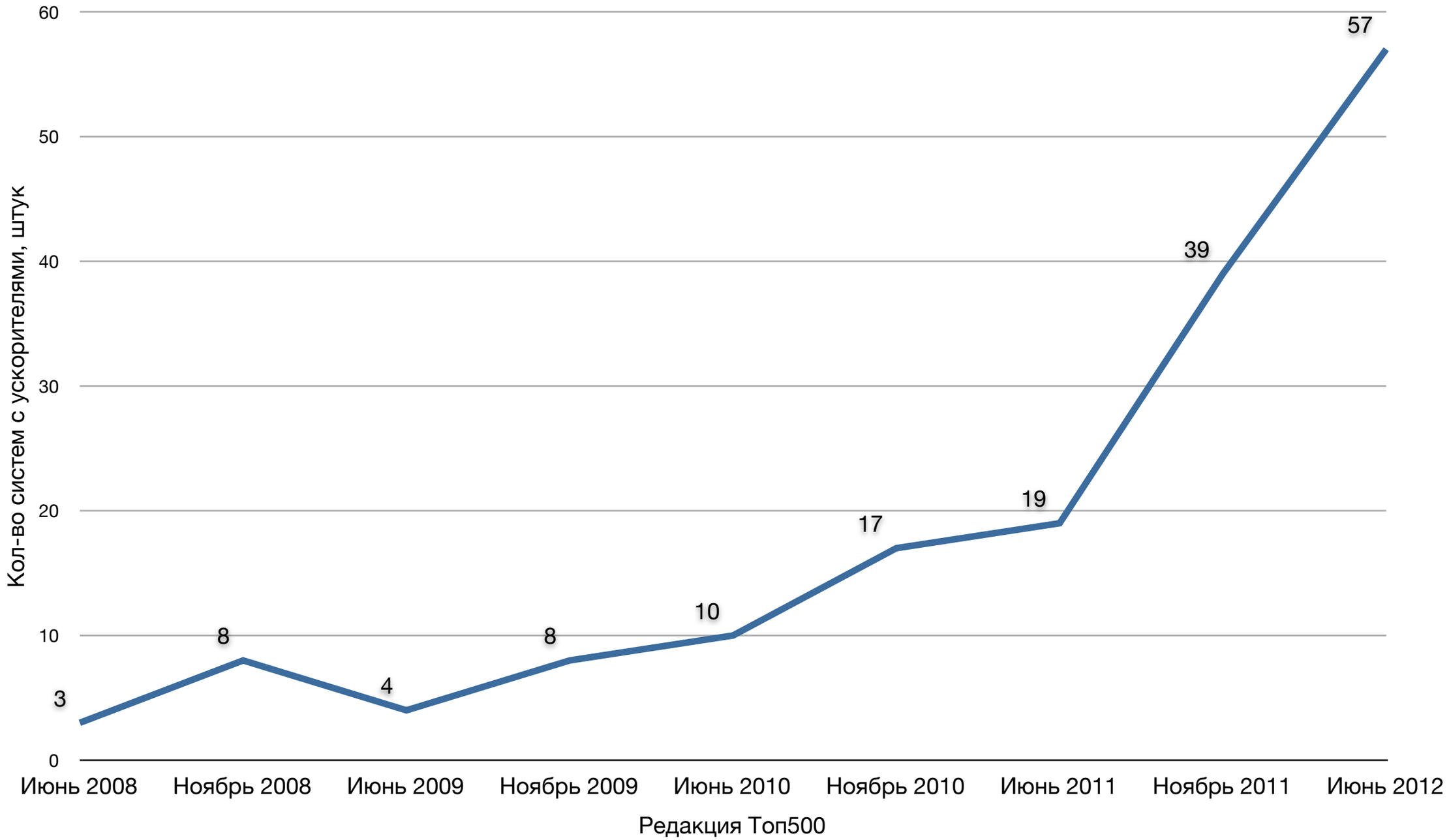
Язык программирования OpenCL

Адинец Андрей Викторович[1,2], к.ф.-м.н.
[1]НИВЦ МГУ имени М.В.Ломонсова
[2]Объединённый институт ядерных исследований
adinetz@gmail.com
twitter: @adinetz

Ускорители в Топ500



Как программировать

- Разные ускорители — разные средства
  - NVidia — CUDA
  - AMD — Brook+, CAL, Kernel C++
  - x86 — C/C++/Fortran + OpenMP
  - Intel (Larrabee, Knights XXX, Phi) — C++ + директивы
  - Intel HD Graphics GPU — ?
- Непереносимость
  - Код
  - Навыки программиста

Цели OpenCL

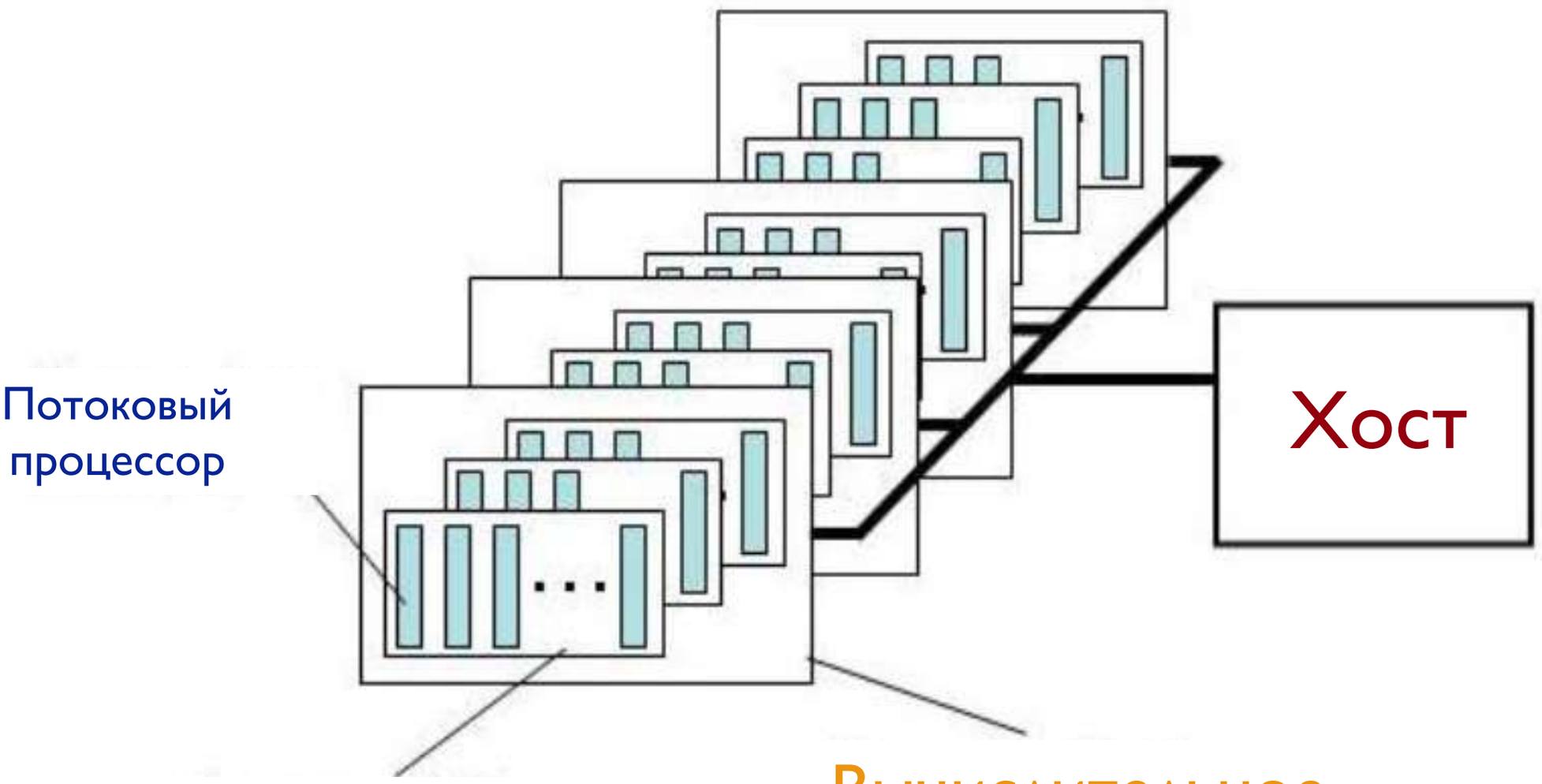
- Стандарт программирования
  - Многоядерные процессоры
  - Ускорители, ГПУ
  - Мобильные медиапроцессоры
- Стандарт функциональности
  - Производителям процессоров

Рабочая группа OpenCL

Поддерживается Khronos Group

Реализации OpenCL

Модель системы



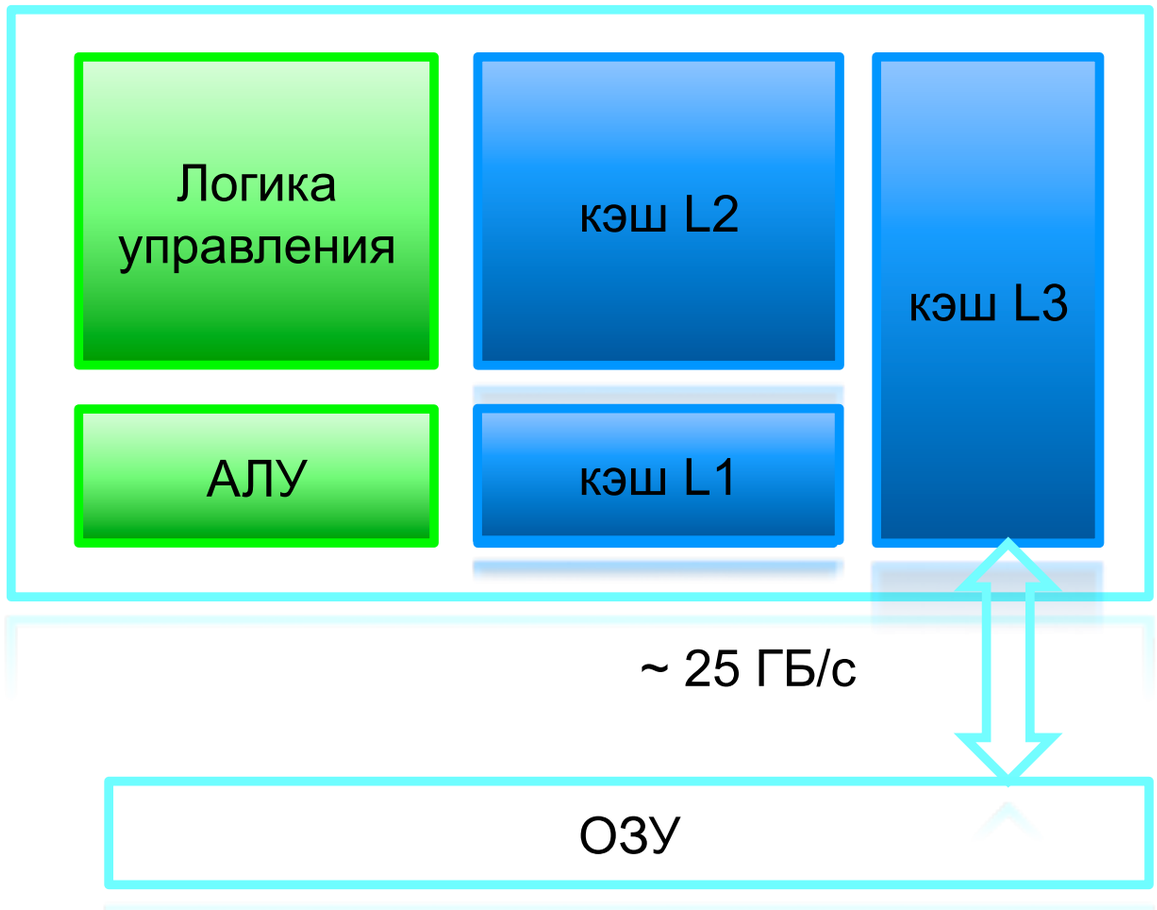
Модель программы

- Хост-программа
- Программа для устройства
- Набор ядер

Аппаратура	Программа
Потоковый процессор	Поток
Мультипроцессор	Блок потоков
Устройство	Ядро
Хост	Хост-программа

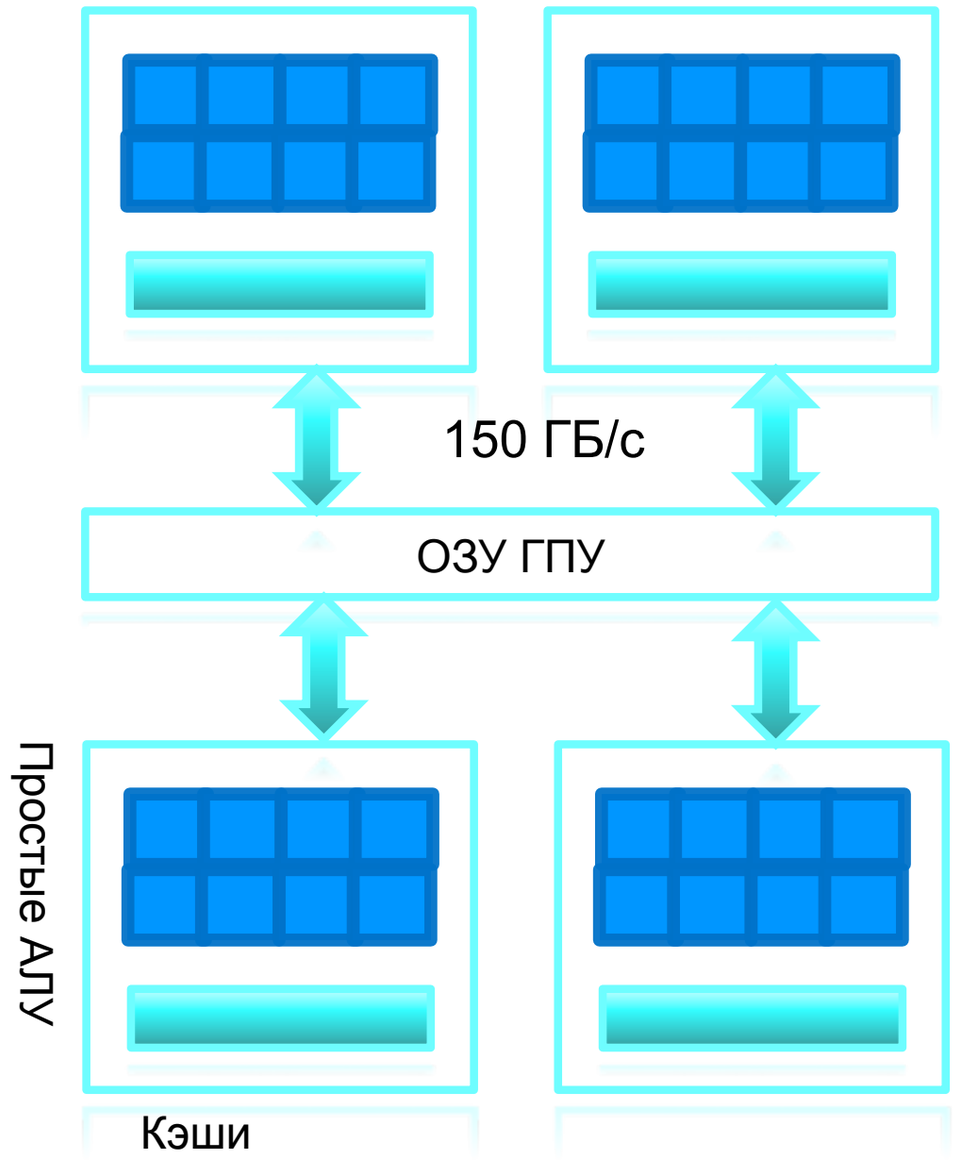
ЦПУ:
минимум задержки

- спекулятивное исполнение
- предсказание ветвлений
- кэши
- мало регистров
- 1 ядро — 1–2 потока

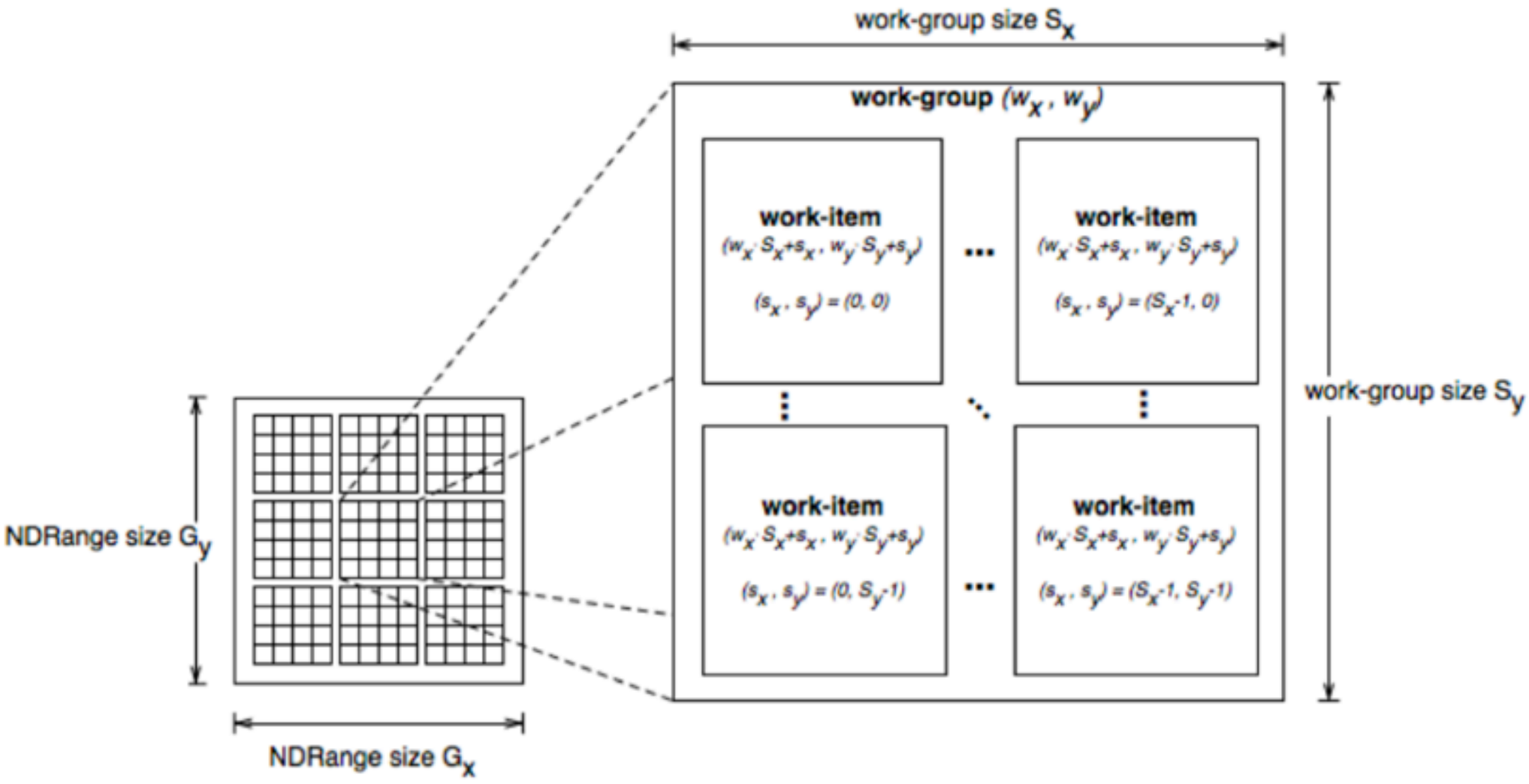


ГПУ:
максимум вычислений

- много простых ядер
- число потоков >> число ядер
- много регистров
- быстрое переключение
- сокрытие задержек

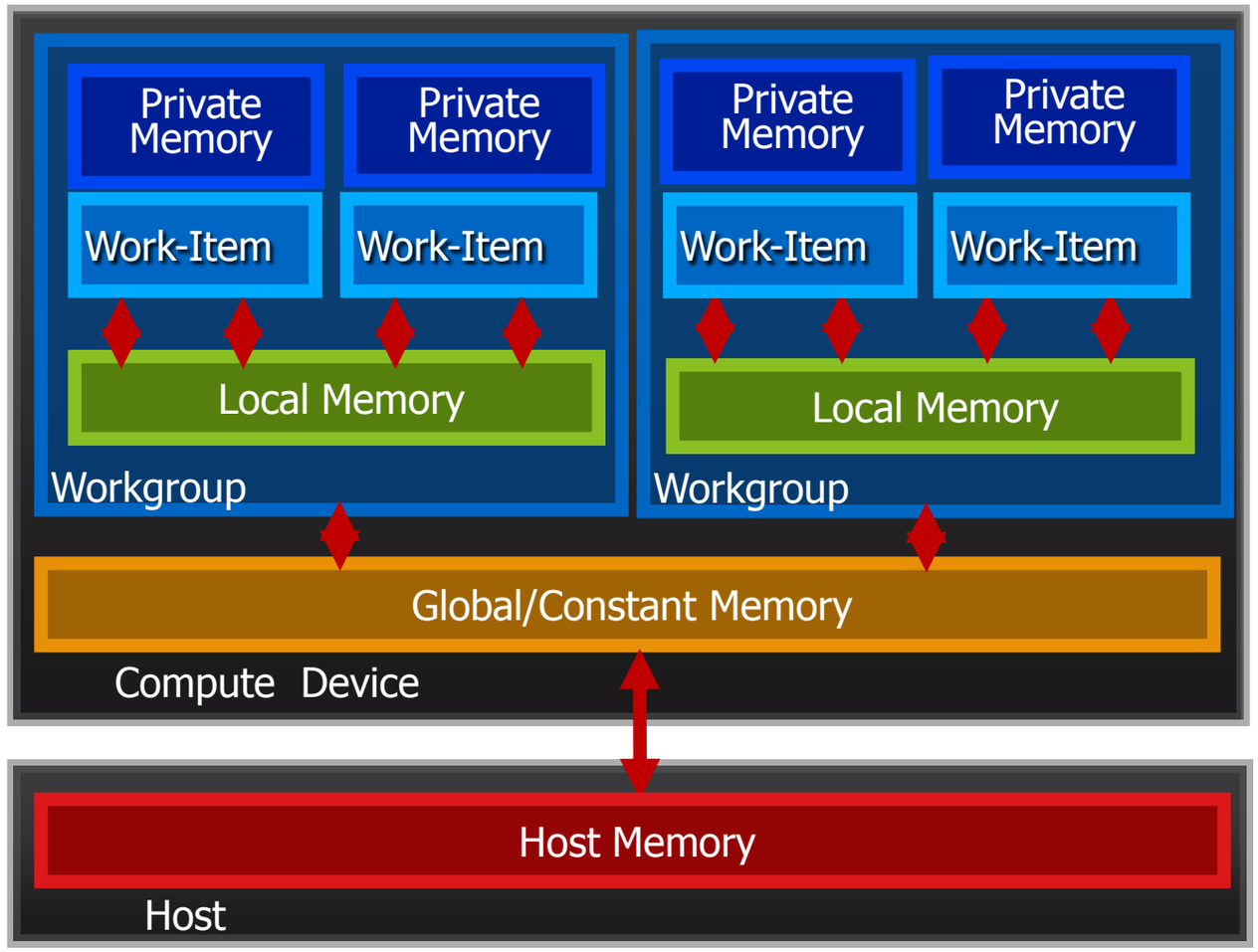


Исполнение ядра



Иерархия памяти

- Приватная
- Локальная
- Константная
- Глобальная
- Хост-память



Соответствие иерархий

Аппаратура	Исполнение	Память	Модификатор
Потоковый процессор	Поток	Приватная	**`private`**, —
Мультипроцессор	Блок потоков	Локальная	**`local`**
Устройство	Ядро	Глобальная Константная	**`global`** **`constant`**
Хост	Хост-программа	Хост-память	—

Модификаторы

- либо **__xxx**, либо **xxx**
- Модификатор **kernel**
  - Функция является ядром
- Модификаторы классов памяти
  - **private**, **local**, **constant**, **global**

Где я?

- Размер блока и решётки в потоках
  - `get_local_size(int dim)`,`get_global_size(int dim)`
- Размер решётки в блоках
  - `get_num_groups(int dim)`
- Номер потока в блоке и решётке
  - `get_local_id(int dim)`,`get_global_id(int dim)`
- Номер блока
  - `get_group_id(int dim)`

Сложение массивов

```
kernel void add_arrays(global float* c, global
  float* a, global float* b, int n) {
   int j = get_global_id(0);
    // support any array size!
    if(j < n) {
      c[j] = a[j] + b[j];
      }
}  // end of addArrays()
```

Чего нет

- Указатель на функцию
- Рекурсия
- Глобальных указателей на глобальные указатели
  - **`global * global p;`** — нельзя!
  - есть `intptr_t, uintptr_t`
- Стандартная библиотека C

Библиотека OpenCL

- Заголовок
  - `#include <cl.h>` (Mac OS X)
  - `#include <CL/cl.h>` (остальные)
- Библиотека
  - `gcc -lOpenCL myprog.c -omyprog`

Объекты библиотеки

- Создание
  - `cl_xxx clCreateXxxYyy(..., cl_int *err);`
- Счётчик ссылок
  - `+ : cl_int clRetainXxx(cl_xxx id);`
  - `– : cl_int clReleaseXxx(cl_xxx id);`
- Получить свойство
  - `cl_int clGetXxxInfo(cl_xxx id, cl_xxx_info param, size_t sz, void *val, size_t* sz_ret);`

Платформа

- Конкретная реализация OpenCL
  - Содержит >= 1 устройств

- Получить платформы

```
int clGetPlatformIDs(
  uint n,
  cl_platform_id *platfs,
  uint *nplatfs)
```

Устройство

- Конкретное вычислительное устройство
  - ГПУ, ускоритель, ЦПУ и т.д.
- Получить устройства

```
int clGetDeviceIDs(
  cl_platform_id platf,
  cl_device_type dev_type,
  uint n,
  cl_device_id *devs,
  uint *ndevs)
```

Демонстрация

Информация об устройстве

Контекст и очередь

- Контекст
  - ~ процесс — память, программы
  - >= 1 однотипных устройств
- Очередь
  - ~ поток — исполнение команд
  - = 1 устройство, 1 контекст

Инициализация

```
cl_platform_id platf;
if(clGetPlatformIds(1, &platf, 0))
  exit(-1);

cl_device_id dev;
if(clGetDeviceIDs(platf, CL_DEVICE_TYPE_GPU, 1, &dev, 0))
  exit(-1);
```

```
cl_context ctx = clCreateContext(0, 1, &dev, 0, 0, 0);
if(!ctx)
  exit(-1);

cl_command_queue queue = clCreateCommandQueue(ctx, dev, 0,
0);
if(!queue)
  exit(-1);
```

Программа

- Исполняемый код устройства
  - >= 1 ядер
- Создание
  - `clCreateProgramWith{Source|Binary}()`
- Сборка
  - `clBuildProgram()`
  - `clGetProgramBuildInfo()`

Ядро

- «Точка входа» в устройство
- Создание
  - `clCreateKernel()`, `clCreateKernelsInProgram()`

Программы и ядра

```
cl_program prog = clCreateProgramWithSource
  (ctx, nlines, programText, 0, 0);
if(!prog) exit(-1);

if(clBuildProgram(prog, 1, &dev, 0, 0, 0)) {
  // get & print build log
  char build_log[MAX_BUILD_LOG + 1];
  clGetProgramBuildInfo
    (prog, dev, CL_PROGRAM_BUILD_LOG, MAX_LOG_SIZE,
  build_log, 0);
  printf(«%s», build_log);
  exit(-1);
}

cl_kernel add_arrays_kernel =
  clCreateKernel(prog, «add_arrays», 0);
if(!add_arrays_kernel) exit(-1);
```

Команды

- Постановка в очередь
  - `clEnqueueXxx(cl_command_queue queue_id, ..., cl_uint nevents, const cl_event* wait_list, cl_event* ev);`
  - Событие `ev` — «идентификатор» команды
  - Исполнение
    - `clFlush(queue)` — отправить команды на устройство
    - `clFinish(queue)` — дождаться завершения

Буфер данных

- Одномерный массив в памяти хоста или устройства
- Копирование
  - `clEnqueue{Read,Write,Copy}Buffer()`
  - Блокирующее/неблокирующее
- Отображение (ЦПУ, Fusion)
  - `clEnqueue{MapBuffer,UnmapMemObject}()`

Работа с данными

```
// create buffers
cl_mem da = clCreateBuffer(ctx, 0, n * sizeof(int), 0, 0);
cl_mem db = clCreateBuffer(ctx, 0, n * sizeof(int), 0, 0);
cl_mem dc = clCreateBuffer(ctx, 0, n * sizeof(int), 0, 0);

// before kernel call: copy to device
clEnqueueWriteBuffer
  (queue, da, CL_TRUE, 0, n * sizeof(int), ha, 0, 0, 0);
clEnqueueWriteBuffer
  (queue, db, CL_TRUE, 0, n * sizeof(int), hb, 0, 0, 0);

// ...
// we call the kernel here
// ...

// after kernel call: copy data back
clEnqueueReadBuffer
  (queue, dc, CL_TRUE, 0, n * sizeof(int), hc, 0, 0, 0);
```

Запуск ядра

- Установка фактических параметров
  - `clSetKernelArg()`
- Запуск на исполнение
  - `clEnqueueNDRangeKernel()`

Запуск ядра

```
// set kernel arguments
clSetKernelArg(add_arrays_kernel, 0, sizeof(cl_mem), &dc);
clSetKernelArg(add_arrays_kernel, 1, sizeof(cl_mem), &da);
clSetKernelArg(add_arrays_kernel, 2, sizeof(cl_mem), &db);
clSetKernelArg(add_arrays_kernel, 3, sizeof(int), &n);

// execution configuration
size_t lws[] = {LWS};
size_t gws[] = {(n / LWS * LWS + (n % LWS == 0 ? 0 : 1)};
size_t gwo[] = {0};  // not supported everywhere

// launch kernel
clEnqueueNDRangeKernel(queue, add_arrays_kernel, 1, gwo,
gws, lws, 0, 0, 0);
// wait for it to finish (optional)
clFinish(queue);
```

Демонстрация

Hello, OpenCL World!

Map/Unmap (1)

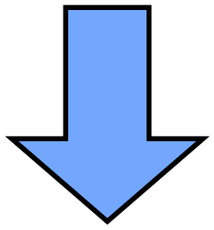
```
// create buffers
cl_mem da = clCreateBuffer(ctx, 0, n * sizeof(int), 0, 0);
cl_mem db = clCreateBuffer(ctx, 0, n * sizeof(int), 0, 0);
cl_mem dc = clCreateBuffer(ctx, 0, n * sizeof(int), 0, 0);

// before kernel call: map & initialize data
ha = (int*)clEnqueueMapBuffer (queue, da, CL_TRUE,
CL_MAP_WRITE, 0, n * sizeof(int), 0, 0, 0, 0);
hb = (int*)clEnqueueMapBuffer (queue, da, CL_TRUE,
CL_MAP_WRITE, 0, n * sizeof(int), 0, 0, 0, 0);
// ... we initialize data here ...
clEnqueueUnmapMemObject(queue, da, ha, 0, 0, 0);
clEnqueueUnmapMemObject(queue, da, ha, 0, 0, 0);

// ... we call the kernel here ...

// after kernel call: copy data back
hc = (float*)clEnqueueMapBuffer (queue, da, CL_TRUE,
CL_MAP_READ, 0, n * sizeof(int), 0, 0, 0, 0);

// work with data referenced by hc
```

Map/Unmap (2)

```
// alloc aligned data
float *ha, *hb, *hc;
posix_memalign((void**)&ha, CACHE_LINE, n * sizeof(int));
posix_memalign((void**)&hb, CACHE_LINE, n * sizeof(int));
posix_memalign((void**)&hc, CACHE_LINE, n * sizeof(int));

// create buffers
cl_mem da = clCreateBuffer(ctx, CL_USE_HOST_PTR, n * sizeof(int), ha, 0);
cl_mem db = clCreateBuffer(ctx, CL_USE_HOST_PTR, n * sizeof(int), hb, 0);
cl_mem dc = clCreateBuffer(ctx, CL_USE_HOST_PTR, n * sizeof(int), hc, 0);

// before kernel call: map & initialize data
clEnqueueMapBuffer(queue, da, CL_TRUE, CL_MAP_WRITE, 0,n * sizeof(int),0,0,0,0);
clEnqueueMapBuffer(queue, db, CL_TRUE, CL_MAP_WRITE, 0,n * sizeof(int),0,0,0,0);
// ... initialize data ...
clEnqueueUnmapMemObject(queue, da, ha, 0, 0, 0);
clEnqueueUnmapMemObject(queue, da, ha, 0, 0, 0);

// ... we call the kernel here ...

// after kernel call: copy data back
clEnqueueMapBuffer (queue, da, CL_TRUE, CL_MAP_READ, 0,n * sizeof(int),0,0,0,0);

// work with data referenced by hc
```

Умножение матриц

```
for(int i = 0; i < n; i++)
  for(int j = 0; j < n; j++) {
    float r = 0;
    for(int k = 0; k < n; k++)
      r += a[i * n + k] * b[k * n + j];
    c[i * n + j] = r;
  }
```



```
kernel void matmul(global float *c, global float *a, global
float *b, int n) {
  int i = get_global_id(1), j = get_global_id(0);
  float r = 0;
  for(int k = 0; k < n; k++)
    r += a[i * n + k] * b[k * n + j];
  c[i * n + j] = r;
}
```

Демонстрация

Простое умножение матриц

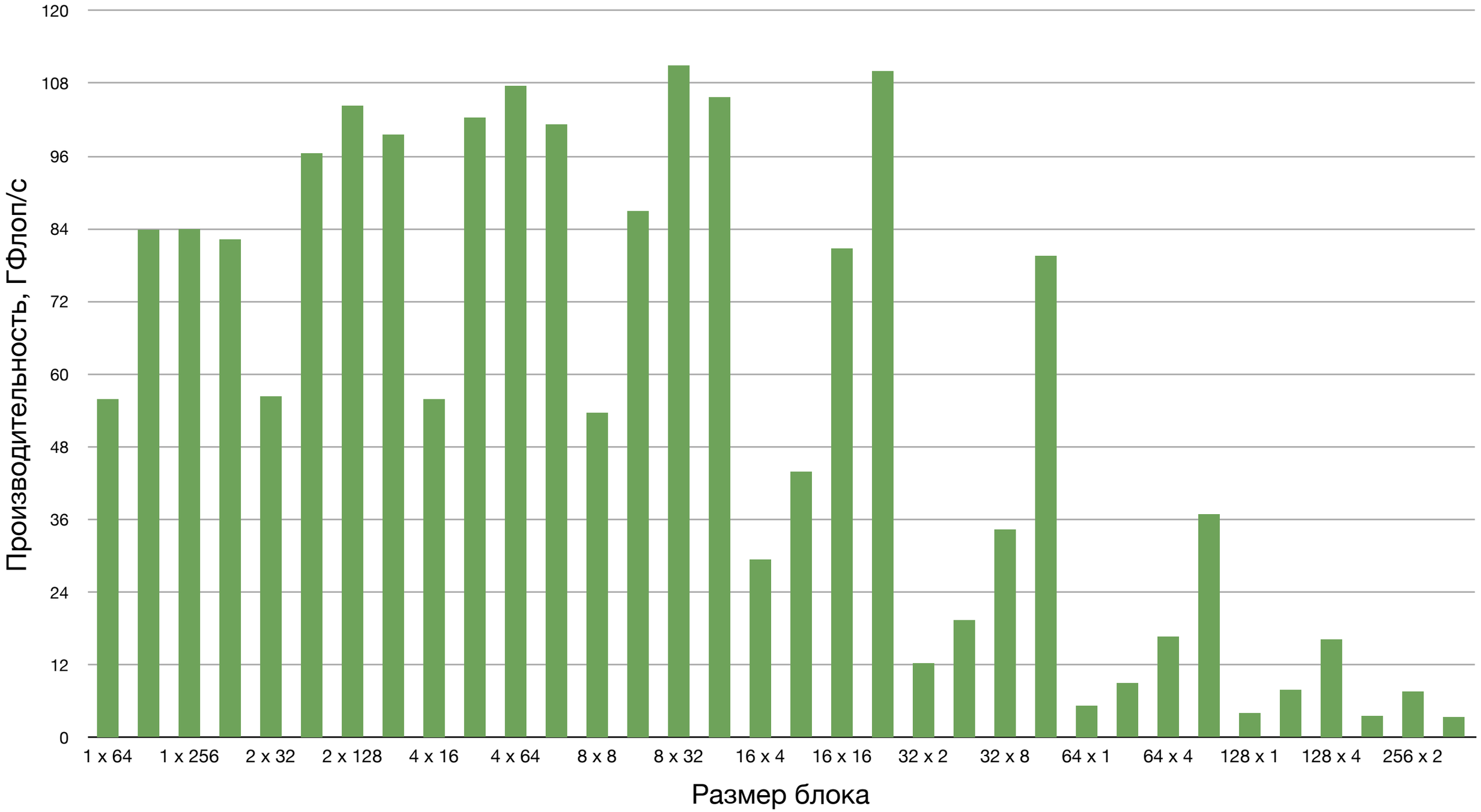
Профилировка

- `clCreateCommandQueue(ctx, dev, CL_QUEUE_PROFILING_ENABLE, 0);`
- `clEnqueueXxx(queue, ..., 0, 0, &event);`
- `clGetEventProfilingInfo(&event, name, sz, &value, 0);`
- `name = CL_PROFILING_COMMAND_{QUEUED, SUBMIT, START, END}`

Оптимизации

- Размер рабочей группы
- Расположение данных, коалесинг
- Развёртка циклов
  - 1 поток — несколько элементов
- Локальная память
  - Совместная работа рабочей группы
- Векторные команды
- Изменение алгоритма

Разный размер блока



Барьеры и заборы

- `CL_{LOCAL,GLOBAL}_MEM_FENCE`
- Забор
  - `mem_fence(cl_mem_fence_flags flags)`
  - `{read|write}_mem_fence(cl_mem_fence_flags flags)`
- Барьер

Использование локальной памяти

```
// array in local memory
local float la[BS][BS];

// iterate over blocks
for(int kb = 0; kb < n; kb += BS) {

  barrier(CLK_LOCAL_MEM_FENCE);
  // load data global -> local
  barrier(CLK_LOCAL_MEM_FENCE);

  // do some useful work

}
```

Векторные типы

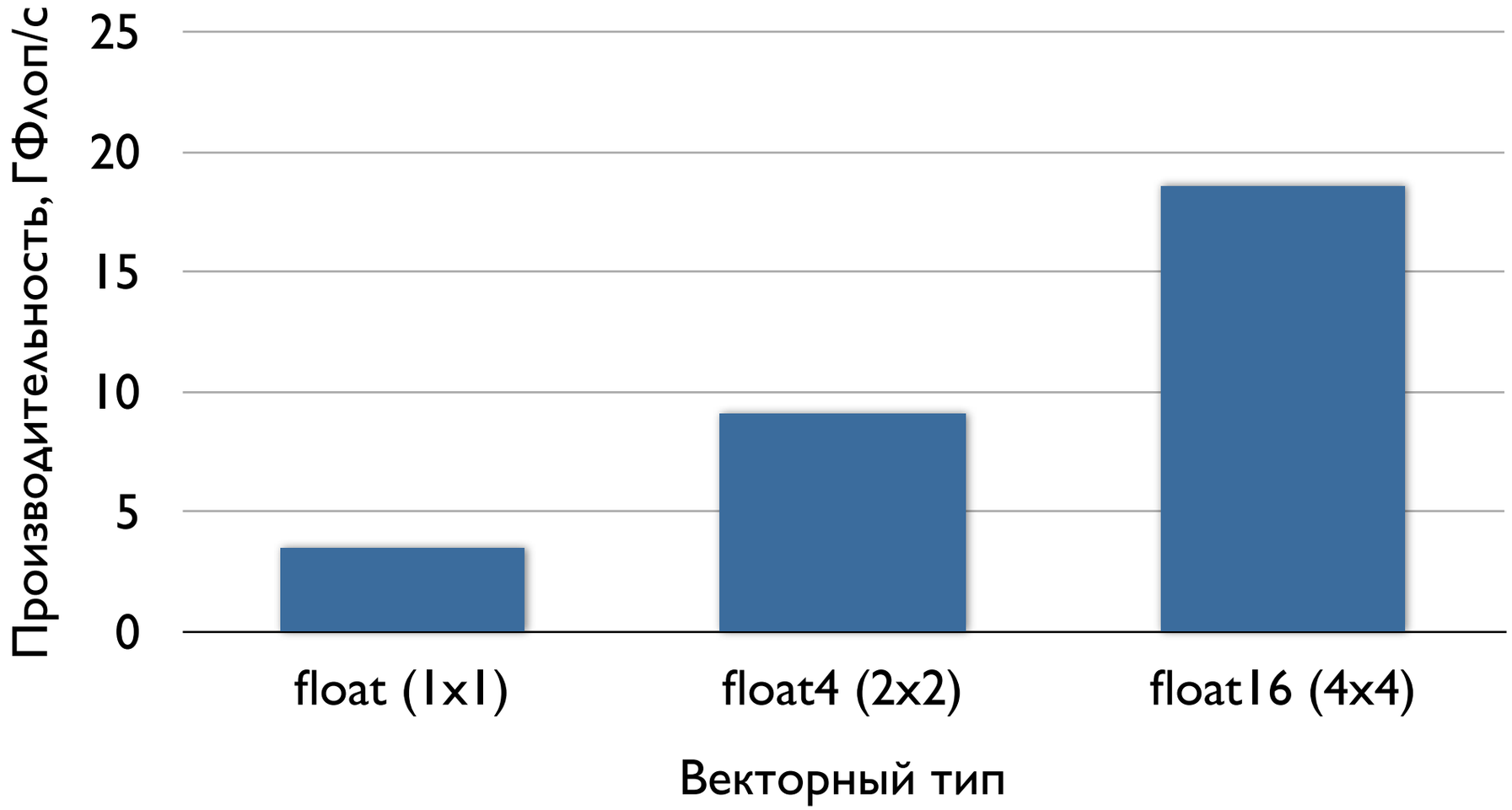
- `type{2,3,4,8,16}` — `uchar8`, `float4`
- Перестановки (swizzles)
  - `b = a.zzyy; b.s0123 = a;`
  - `c.odd = b;`
- Арифметические операции, стандартные функции
- Полезны для ЦПУ

Демонстрация

Оптимизация умножения матриц

Векторные типы

Производительность AB^T от используемого типа
Пик 51.6 ГФлоп/с



Расширения

- `cl_khr_<имя-расширения>`
- Определения
  - `enum CL_<имя-перечисления>_KHR`
  - `cl<имя-функции>KHR`
- Использовать в ядре:
  - `#pragma OPENCL EXTENSION cl_khr_<имя-расширения> : {enable|disable}`

Расширения (2)

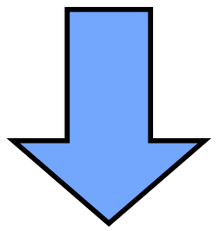
- Взаимодействие с графикой:
  - cl_khr_gl_sharing, cl_khr_d3d10_sharing
- Новые типы данных:
  - cl_khr,_fp64, cl_amd_fp64, cl_khr_half
- Атомарные операции
  - cl_khr_{local|global}_int{32|64}_{base|extended}_atomics

Атомарные операции

- `T atom_op(T *p, T val)`
  - `T=`**`int, unsigned int, long, unsigned long`**
  - `op=add, sub, xchg, min, max, and, or, xor`
- `T atom_op(T *p)`
  - `op=inc, dec`
- `T atom_cmpxchg(T *p, T cmp, T val)`
- Атомарность
  - «Одновременно» исполняет только один поток

Гистограмма

```
for(int i = 0; i < n; i++) {
  bins[data[i]]++;
}
```



```
kernel void histo(global int *bins,
global uchar* data, int n) {
  int i = get_global_id(0);
  atom_inc(&bins[data[i]]);
}
```

Демонстрация

Гистограмма.

Локальные и глобальные атомарные операции.

Чего не хватает

- Один код для хоста и ускорителя
- Нормальные указатели
- Более широкая поддержка возможностей «железа»

Практикум

- ауд. 574, 15:30 – 20:30
- Задание 0: «Графит»
  - Запустить сэмпл
- Задание 1: массивы
  - w[i] = a * x[i] + b * y[i] * z[i]
- Задание 2: самый большой простой uint
  - Простой способ, решето Эратосфена, max perf
- Задание 3: сортировка 1 ГБ целых чисел
  - Битоническая, побитовая, max perf

Ссылки

- http://www.khronos.org/opencl/
- http://www.khronos.org/conformance/adopters/conformant-products/
- http://software.intel.com/en-us/articles/vcsource-tools-opencl-sdk/
- http://nvidia.com/opencl.html
- http://developer.amd.com/zones/OpenCLZone
- http://www.alphaworks.ibm.com/tech/opencl