Java на GPU: где мы сейчас?

И зачем?

Дмитрий Александров
@bercut2000 | T-Systems


S3 Trio 3D/2X On Board QCA2HC
86C368
0018 B3A1C
KOREA
VOLCANO
HN27C256HG-70
QC OK


3D fx

Что такое Видеокарта?

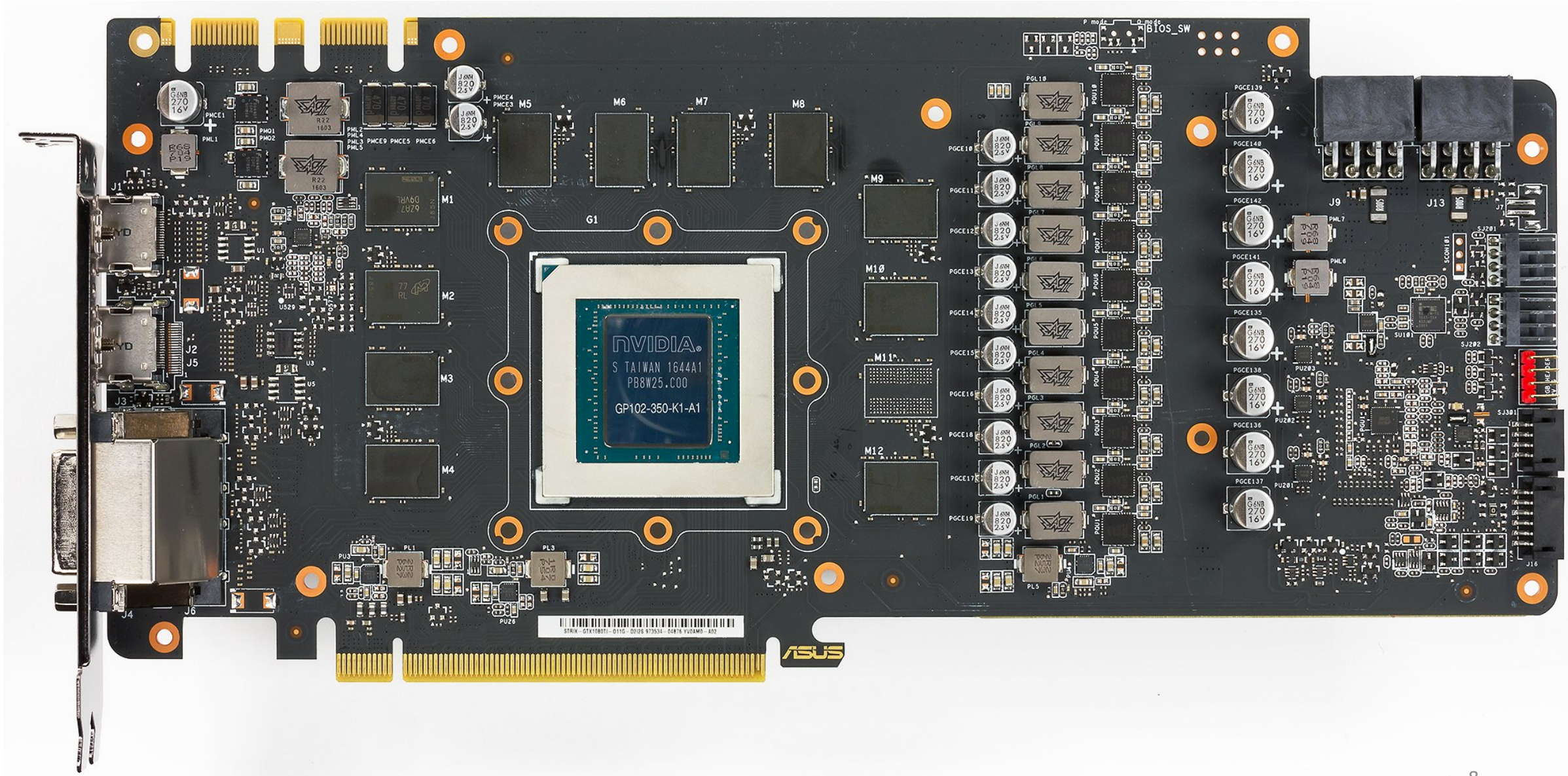
Устройство, преобразующее графический образ, хранящийся как содержимое памяти компьютера (или самого адаптера), в форму, пригодную для дальнейшего вывода на экран монитора.

Что такое Видеокарта?

Но сегодня:

видеокарты не ограничиваются простым выводом изображения, они имеют встроенный графический процессор, который может производить дополнительную обработку, снимая эту задачу с центрального процессора компьютера.

И что она делает?


NVIDIA
S TAIWAN 1644A1
PB8W25.COO
GP102-350-K1-A1
ASUS
BIOS_SW
G1
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
J1
J2
J3
J4
J5
J6
J9
J13

Что такое GPU?

- Graphics Processing Unit

Что такое GPU?

- Graphics Processing Unit
- Популяризировано Nvidia в1999

Что такое GPU?

- Graphics Processing Unit
- Популяризировано Nvidia в1999
- GeForce 256 называют «Первым в мире GPU»

Что такое GPU?

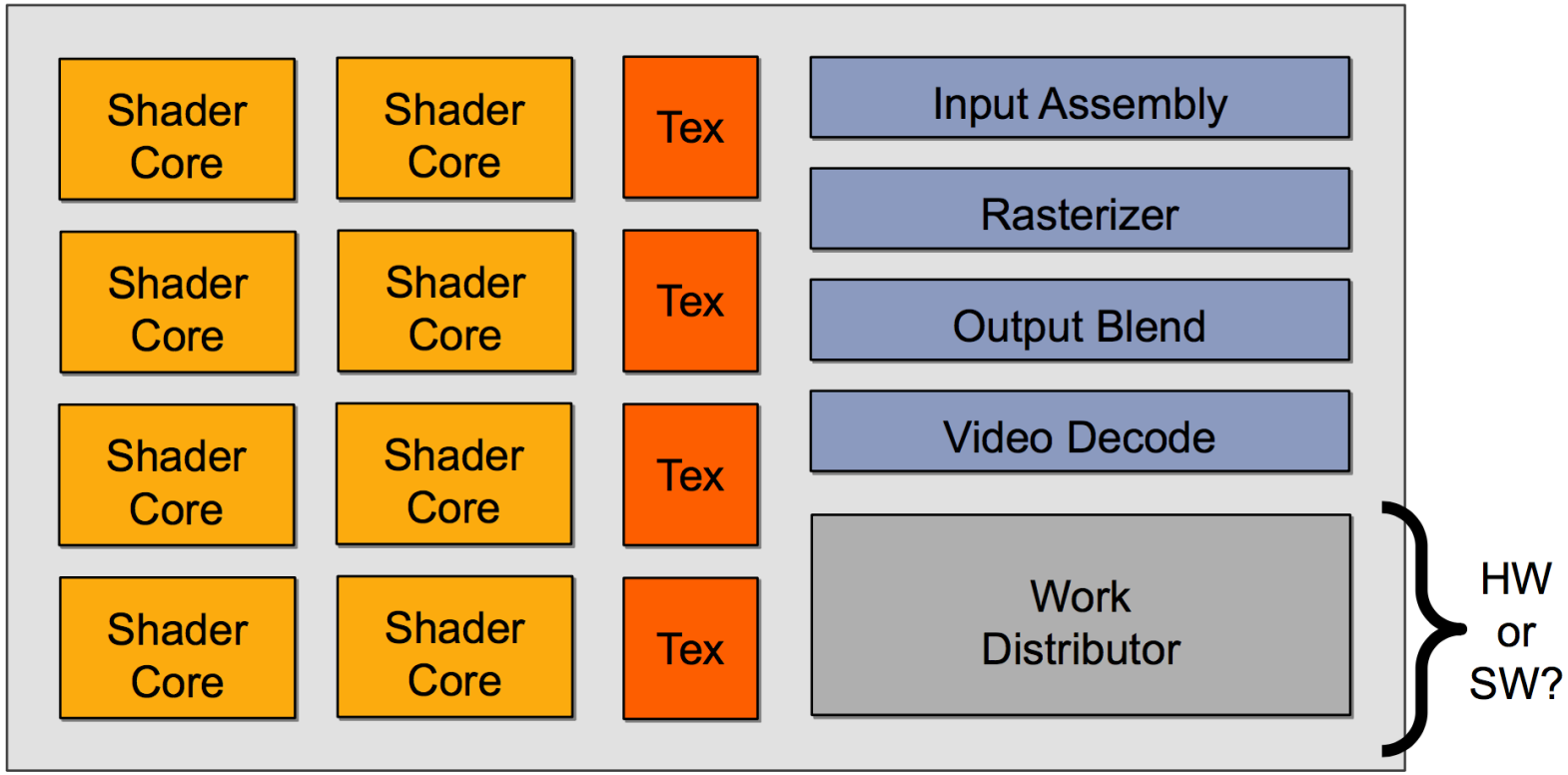
- В то время определялось как “одночипный процессор с интегрированными движками для обработки трансформаций, освещения и рендеринга способный обрабатывать минимум 10,000,000 полигонов в секунду”

Что такое GPU?

- В то время определялось как “одночипный процессор с интегрированными движками для обработки трансформаций, освещения и рендеринга способный обрабатывать минимум 10,000,000 полигонов в секунду”

- ATI их, правда, называла VPU..

По сути это выглядит примерно так



GPGPU

- General-purpose computing on graphics processing units

GPGPU

- General-purpose computing on graphics processing units
- Вычисления относящиеся не только к графике…

GPGPU

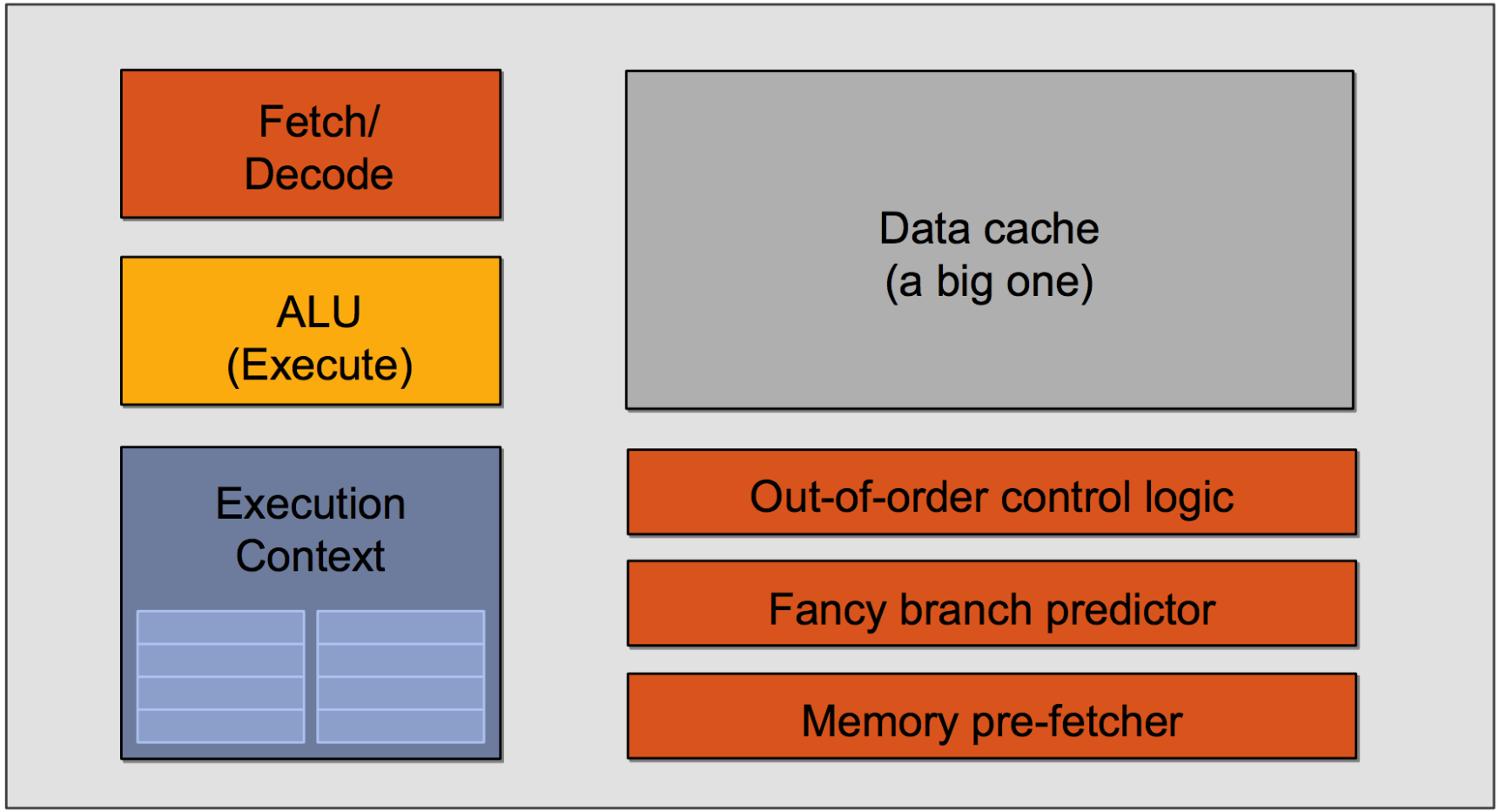
- General-purpose computing on graphics processing units
- Вычисления относящиеся не только к графике…
- … но и те, которые обычно делают CPU

Это круто, нам надо их применить!

Посмотрим на железо

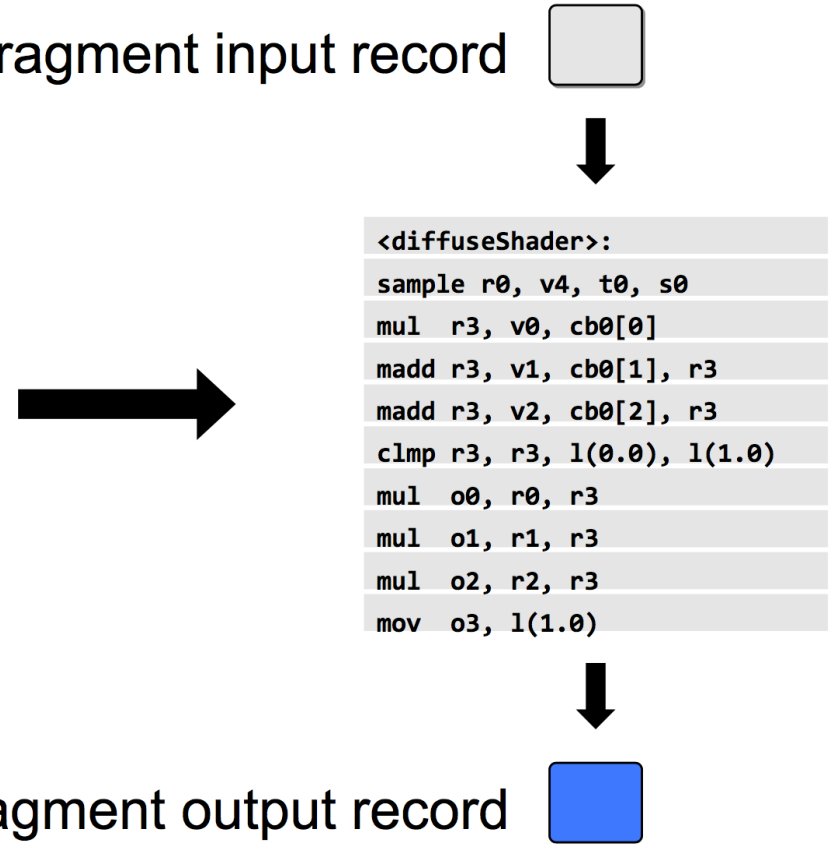
Based on
“From Shader Code to a Teraflop: How GPU Shader Cores Work”, By Kayvon Fatahalian, Stanford University

Примерно так выглядит CPU

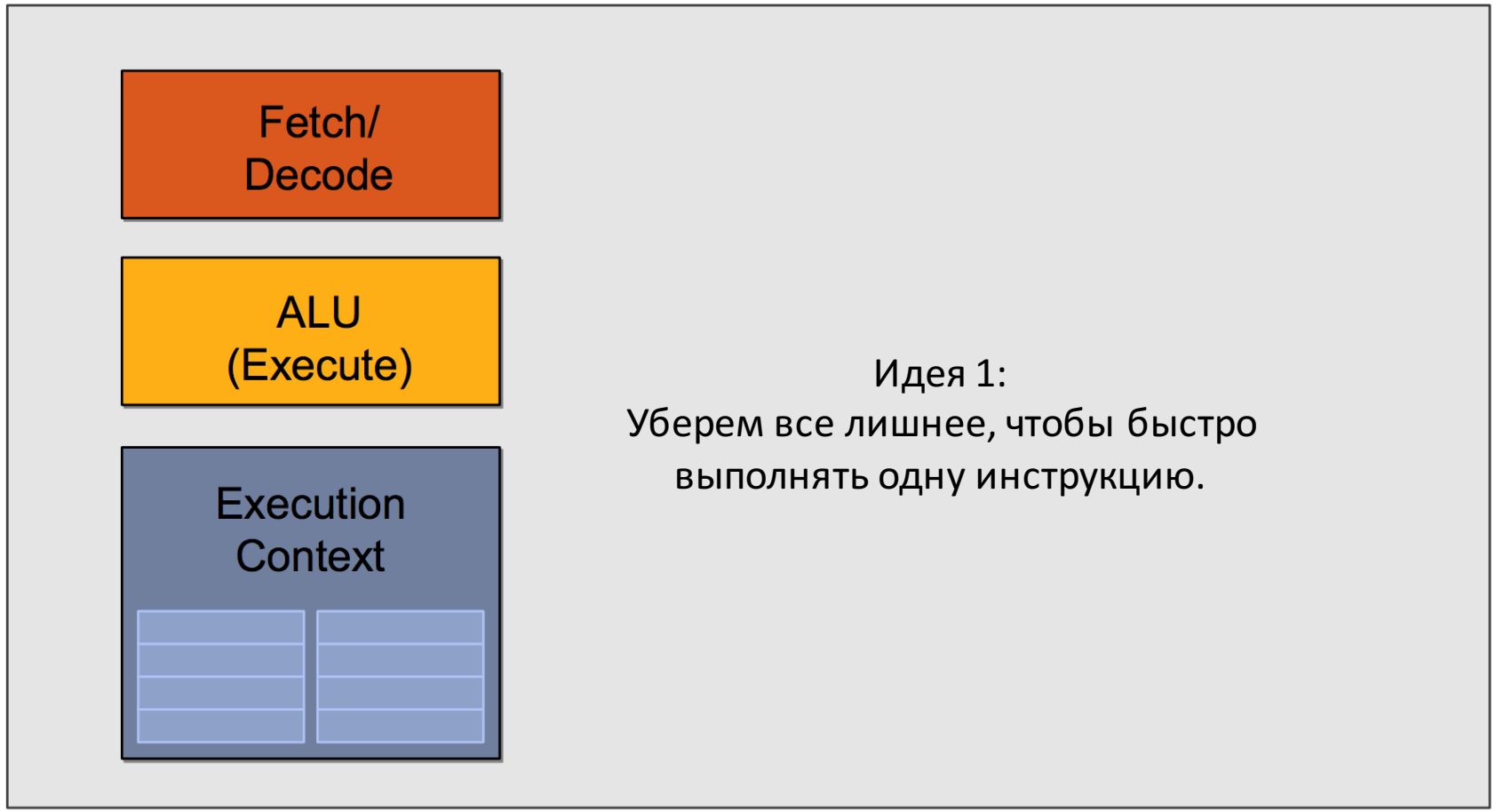


А что нам нужно?

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

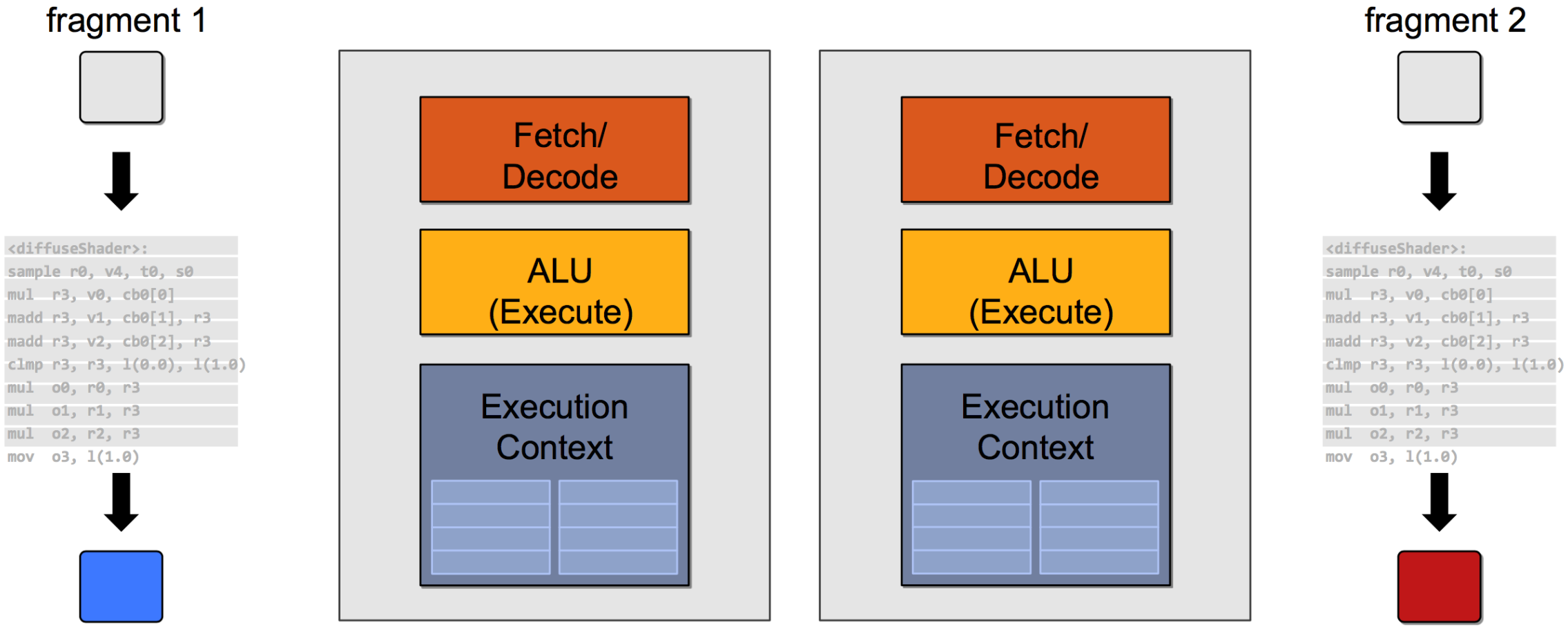


Упростим…

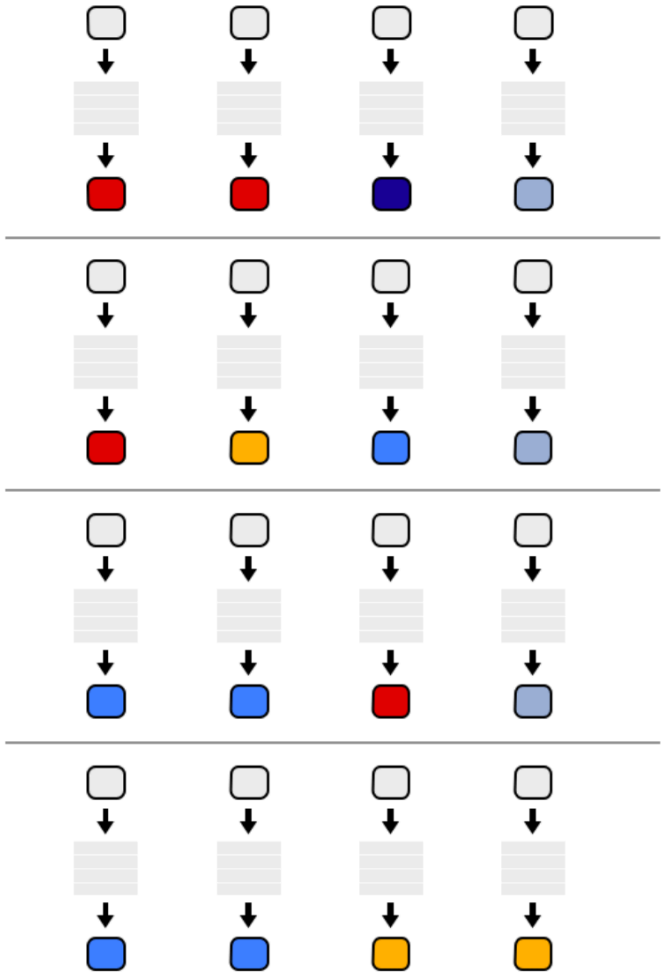


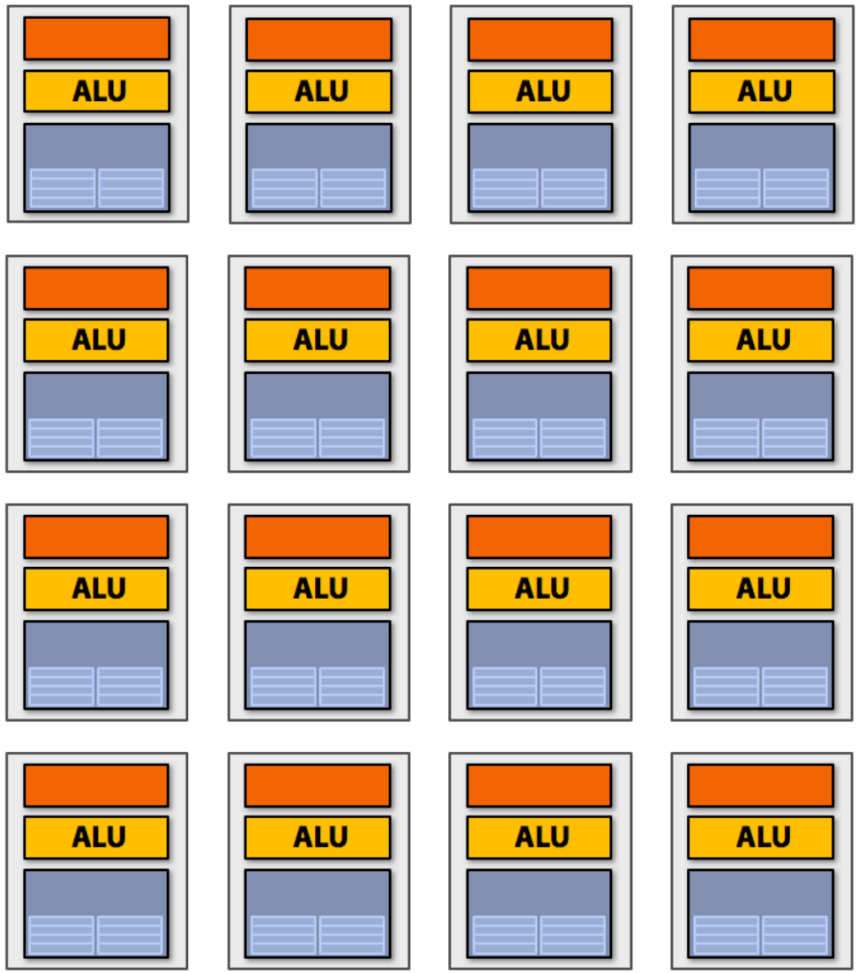
Идея 1:
Уберем все лишнее, чтобы быстро выполнять одну инструкцию.

Далее просто размножим



Чтобы было совсем много

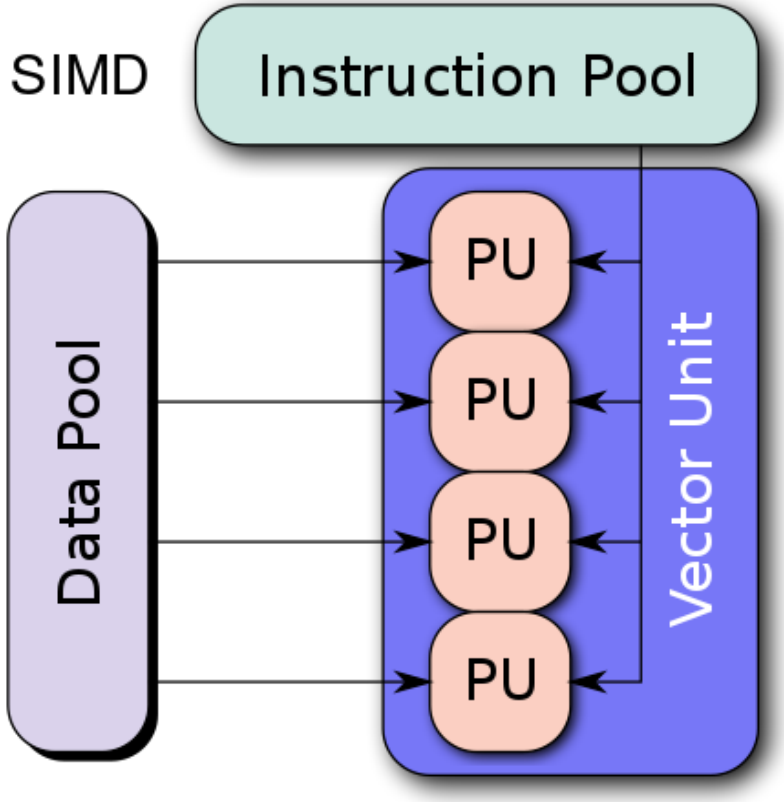




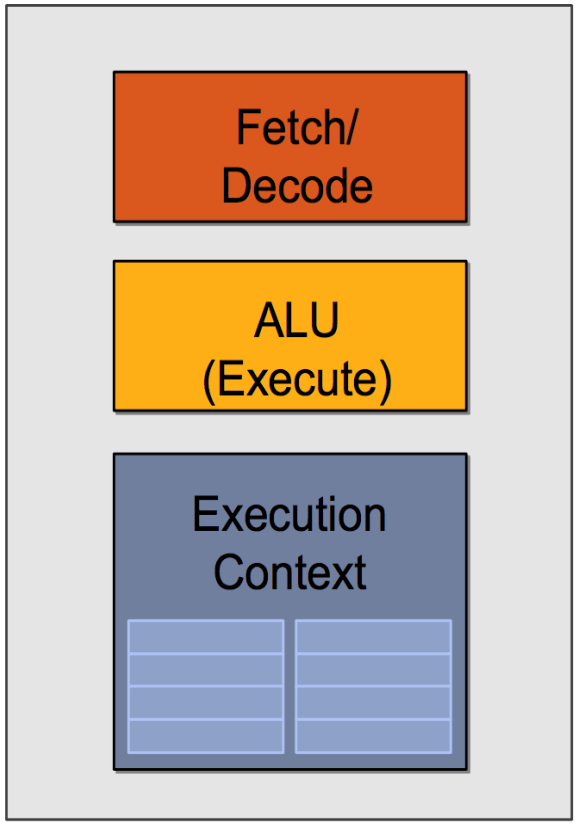
16 cores = 16 simultaneous instruction streams

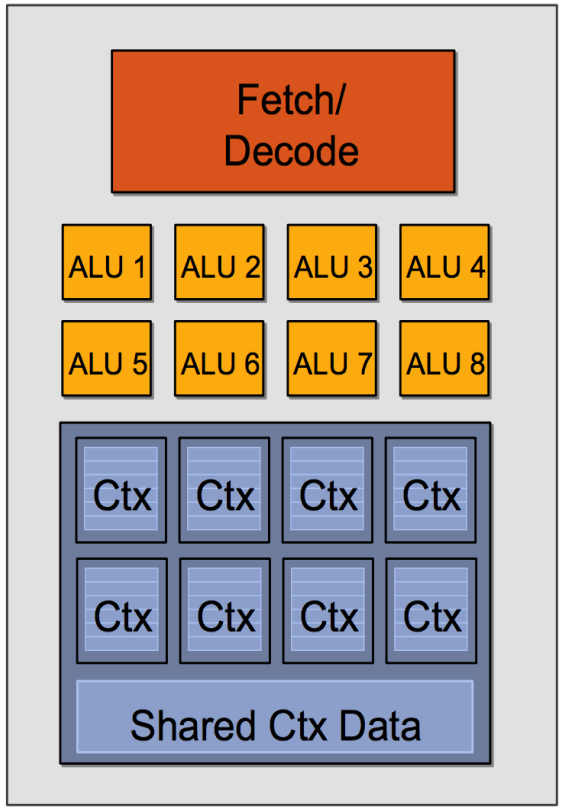
Но ведь мы делаем примерно одно и то же, но с разными данными

И мы доходим до SIMD парадигмы



И мы доходим до SIMD парадигмы

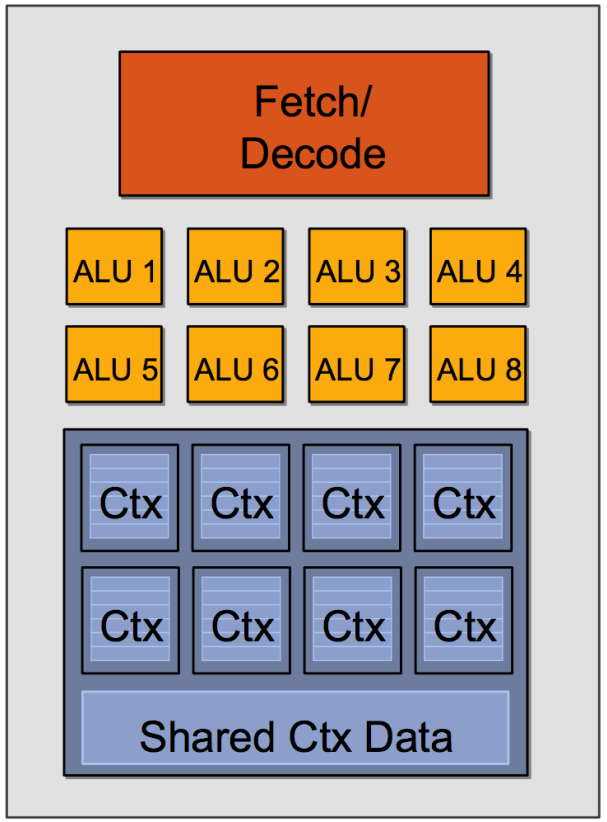


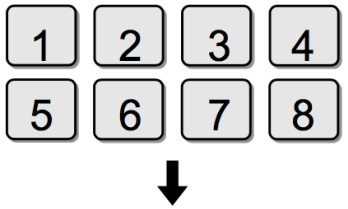


Идея 2:
Упростим менеджмент инструкций
и раскинем его на множество АЛУ

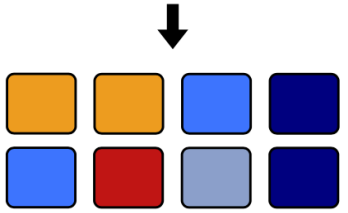
SIMD processing

И тут мы начинаем рассуждать в векторах

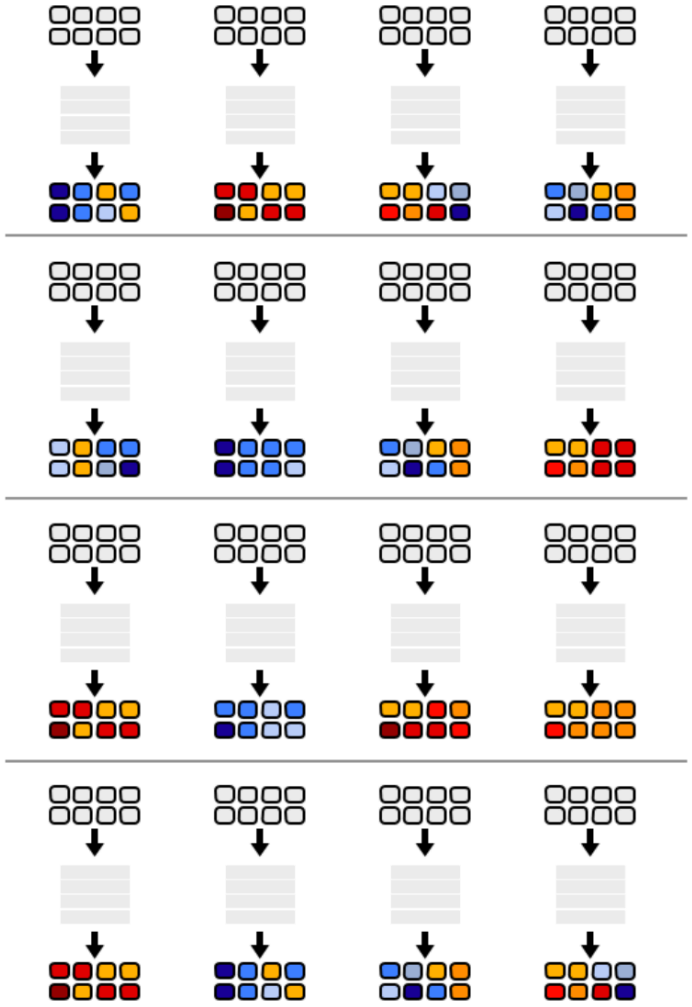


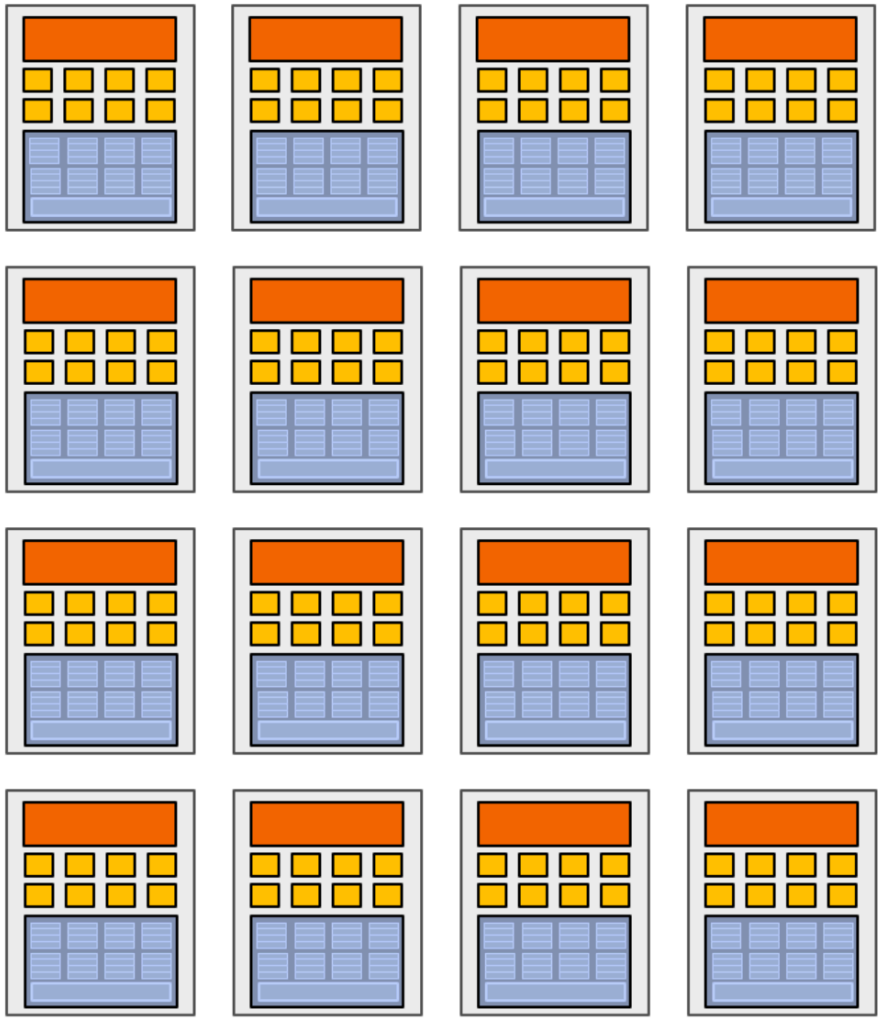



```
<VEC8_diffuseShader>:
VEC8_sample vec_r0, vec_v4, t0, vec_s0
VEC8_mul  vec_r3, vec_v0, cb0[0]
VEC8_madd vec_r3, vec_v1, cb0[1], vec_r3
VEC8_madd vec_r3, vec_v2, cb0[2], vec_r3
VEC8_clmp vec_r3, vec_r3, l(0.0), l(1.0)
VEC8_mul  vec_o0, vec_r0, vec_r3
VEC8_mul  vec_o1, vec_r1, vec_r3
VEC8_mul  vec_o2, vec_r2, vec_r3
VEC8_mov  o3, l(1.0)
```



… и доходим до примерно такого





16 cores = 128 ALUs , 16 simultaneous instruction streams

Собственно как под это все кодить?

Все это началось с шейдеров (Shaders)

- Крутые видеокарты могли могли разгружать CPU от некоторых задач

Все это началось с шейдеров (Shaders)

- Крутые видеокарты могли могли разгружать CPU от некоторых задач

- Но большинство алгоритмов были “хардкоднутыми”

Все это началось с шейдеров (Shaders)

- Крутые видеокарты могли могли разгружать CPU от некоторых задач
- Но большинство алгоритмов были “хардкоднутыми”
- Они считались “Стандартными”

Все это началось с шейдеров (Shaders)

- Крутые видеокарты могли могли разгружать CPU от некоторых задач
- Но большинство алгоритмов были “хардкоднутыми”
- Они считались “Стандартными”
- Программисты просто могли вызывать их

Все это началось с шейдеров (Shaders)

- Но, понятно, не все можно сделать «захардхоженными» алгоритмами

Все это началось с шейдеров (Shaders)

- Но, понятно, не все можно сделать «захардхоженными» алгоритмами
- Поэтому некоторые производители видеокарт «открыли доступ», чтобы программисты загружали свои программы

Все это началось с шейдеров (Shaders)

- Но, понятно, не все можно сделать «захардхоженными» алгоритмами
- Поэтому некоторые производители видеокарт «открыли доступ», чтобы программисты загружали свои программы
- Эти небольшие программы и называются Shaders

Все это началось с шейдеров (Shaders)

- Но, понятно, не все можно сделать «захардхоженными» алгоритмами
- Поэтому некоторые производители видеокарт «открыли доступ», чтобы программисты загружали свои программы
- Эти небольшие программы и называются Shaders
- С этого момента видеоадаптеры могли обрабатывать трансформации, геометрию и текстуру как угодно программисту

Все это началось с шейдеров (Shaders)

- Сначала шейдеры были разных типов:
  - Vertex
  - Geometry
  - Pixel

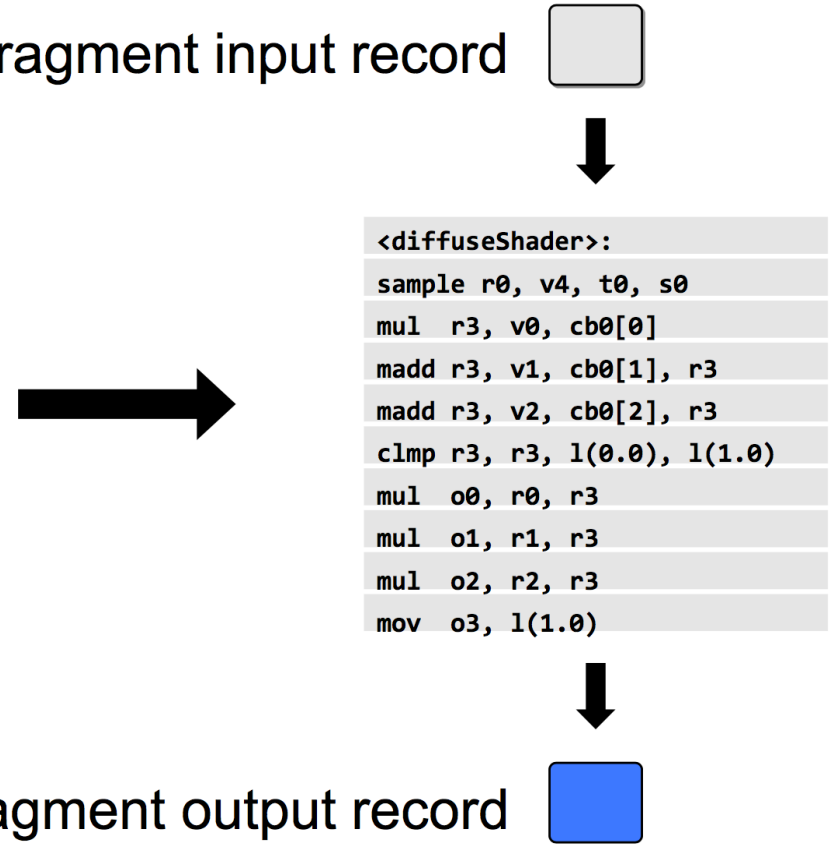
- Но потом их объединили Common Shader Architecture

Существует несколько шейдерных языков

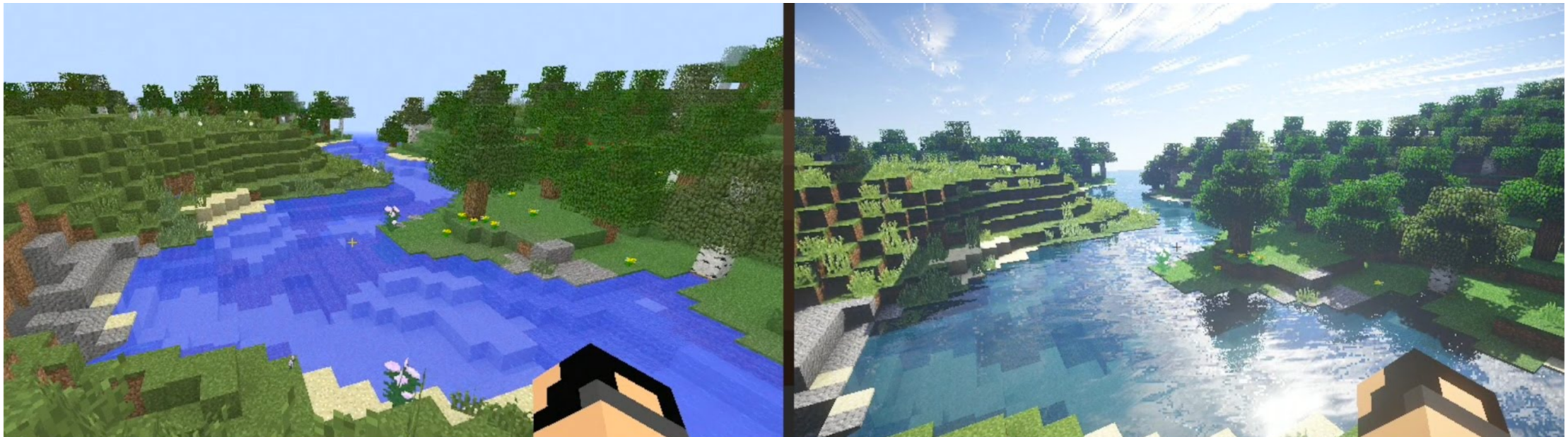
- **RenderMan**
- **OSL**
- **GLSL**
- **Cg**
- **DirectX ASM**
- **HLSL**
- **…**

Пример шейдера

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```



Эффект на лицо (на монитор)



Но хочется не порой не столь низкий уровень..

Учитывая, что это все началось с игрушек..

Общеизвестные абстракции:

- OpenGL
  - это cross-language, cross-platform application programming interface (API) для рендеринг 2D и 3D векторной графики. Данное API типично ориентировано на GPU, для достижения hardware-accelerated rendering.
  - Silicon Graphics Inc., (SGI) начало разработку OpenGL в 1991 выпустило в январе 1992
- DirectX
  - Direct3D это графическое API для Microsoft Windows. Часть DirectX, Direct3D для рендеринга 3D векторной графики. Direct3D использует hardware acceleration если она доступна на видеоадаптере, позволяя полное или частичное видео ускорение.

Кстати, у нас тут конф по Java..

OpenGL в Java

- JSR – 231

OpenGL в Java

- JSR – 231
- Начали в 2003 г.

OpenGL в Java

- JSR – 231
- Начали в 2003 г.
- Крайний релиз 2008 г.

OpenGL в Java

- JSR – 231
- Начали в 2003 г.
- Крайний релиз 2008 г.
- Поддерживается OpenGL 2.0

OpenGL

- Ныне независимый проект GOGL

OpenGL

- Ныне независимый проект GOGL
- Поддерживается OpenGL up to 4.5

OpenGL

- Ныне независимый проект GOGL
- Поддерживается OpenGL up to 4.5
- Позволяет воспользоваться GLU и GLUT

OpenGL

- Ныне независимый проект GOGL
- Поддерживается OpenGL up to 4.5
- Позволяет воспользоваться GLU и GLUT
- Доступ до низкоуровневого API написанного на C через JNI

Но где-то после 2005-го года пришло осознание, что это нужно не только для игрушек!

BrookGPU

- Ранние попытки применить GPGPU

BrookGPU

- Ранние попытки применить GPGPU
- Собственное подмножество ANSI C

BrookGPU

- Ранние попытки применить GPGPU
- Собственное подмножество ANSI C
- Brook Streaming Language

BrookGPU

- Ранние попытки применить GPGPU
- Собственное подмножество ANSI C
- Brook Streaming Language
- Разработан в Stanford University

GPGPU

- *CUDA* — Nvidia проприетарная технология. C-подобный язык.
- *DirectCompute* — Microsoft проприетарный шейдерный язык, часть Direct3d, начиная с DirectX 10.
- *AMD FireStream* — ATI проприетарная технология.
- OpenACC – консорциум 4х производителей
- C++ AMP – Microsoft проприетарный язык
- *OpenCL* – Единый стандарт под контролем Kronos group.

Зачем вообще связывать Java и GPGPU?

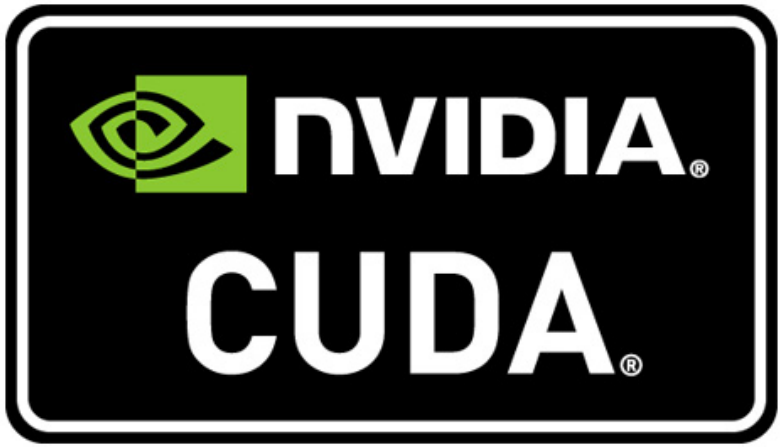
- Почему Java
  - Безопасная и гибкая
  - Portability (как бы “write once, run everywhere”)
  - Распространенная (прям везде)

Зачем вообще связывать Java и GPU

- Почему Java
  - Безопасная и гибкая
  - Portability (как бы “write once, run everywhere”)
  - Распространенная (прям везде)

- Где приделать GPU
  - Data Analytics and Data Science (Hadoop, Spark …)
  - Security analytics (log processing)
  - Finance/Banking

Для этого у нас есть:



Но ведь Java у нас работает на JVM.. Там же все на низком уровне..

Для низкого уровня уровня мы обычно используем:

- JNI (Java Native Interface)
- JNA (Java Native Access)

Но ведь это же безумие.. Так же можно с ума сойти...

Но были фанаты делать и так..

Может уже что-то сделано..?

Для OpenCL:

- JOCL

Для OpenCL:

- JOCL
- JogAmp

Для OpenCL:

- JOCL
- JogAmp
- JavaCL (уже не поддерживается)

.. или для Cuda

- JCuda
  - Cublas
  - JCufft
  - JCurand
  - JCusparse
  - JCusolver
  - Jnvgraph
  - Jcudpp
  - JNpp
  - JCudnn

Работать с GPU это сложно!

- Это не просто так запустить программку

Работать с GPU это сложно!

- Это не просто так запустить программку
- Необходимо знать на каком оборудовании работаешь

Работать с GPU это сложно!

- Это не просто так запустить программку
- Необходимо знать на каком оборудовании работаешь
- Работает на низком уровне

Сначала рассмотрим:

Что это такое?

- Сокращение Open Compute Language

Что это такое?

- Сокращение Open Compute Language
- Консорциум Apple, nVidia, AMD, IBM, Intel, ARM, Motorola и других компаний

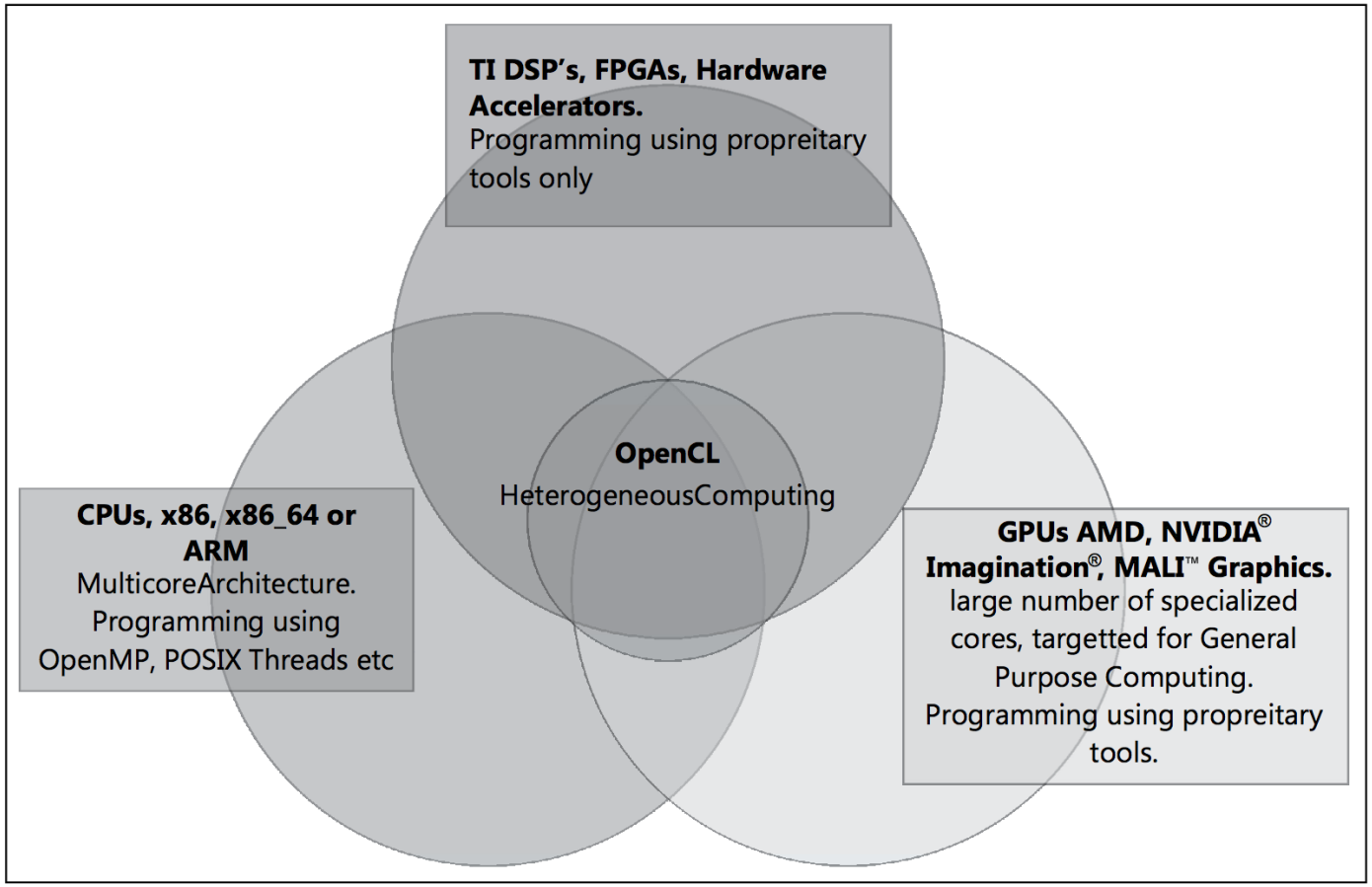
Что это такое?

- Сокращение Open Compute Language
- Консорциум Apple, nVidia, AMD, IBM, Intel, ARM, Motorola и других компаний
- Очень абстрактная модель

Что это такое?

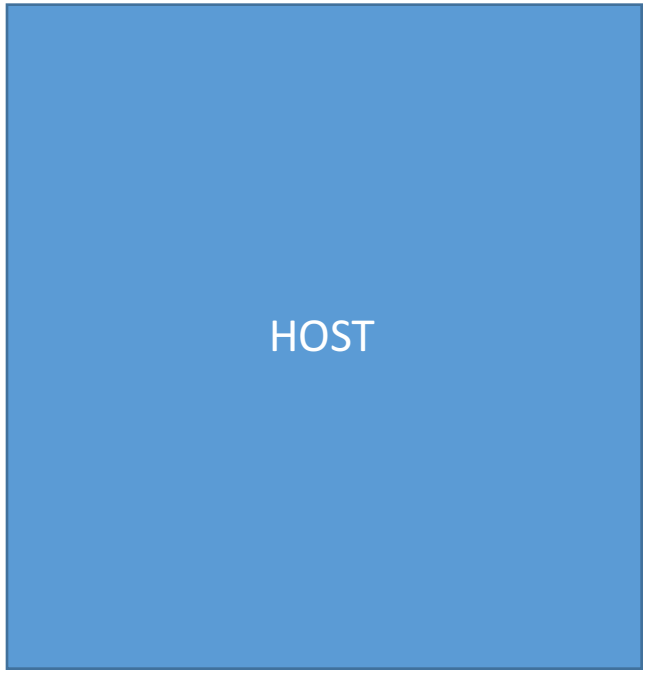
- Сокращение Open Compute Language
- Консорциум Apple, nVidia, AMD, IBM, Intel, ARM, Motorola и других компаний
- Очень абстрактная модель
- Работает и на GPU и на CPU

По идее должен работать на всем



В общем оно работает так:

В общем оно работает так:



В общем оно работает так:

Типичный жизненный цикл OpenCL приложения

- Создается context
- Создается command queue
- Создаются memory buffers/заполняются входными данными
- Создается программа из sources/грузятся binaries
- Компилируется (если надо)
- Создается kernel из программы
- Устанавливаются kernel аргументы
- Устанавливается ND range
- Выполняется
- Возвращается resulting data
- Освобождаются ресурсы

Лучше на это взглянуть...


OPENCL
IS DAMN HARD!
HD
HISTORY.COM
imgflip.com

1. Есть host code. Он на Java.

2. Есть device code. Это специфическое подмножество языка C.

3. Коммуникация между host и device происходит с помощь memory buffers.

Так что мы там можем передавать?

Данные там не совсем такие-же

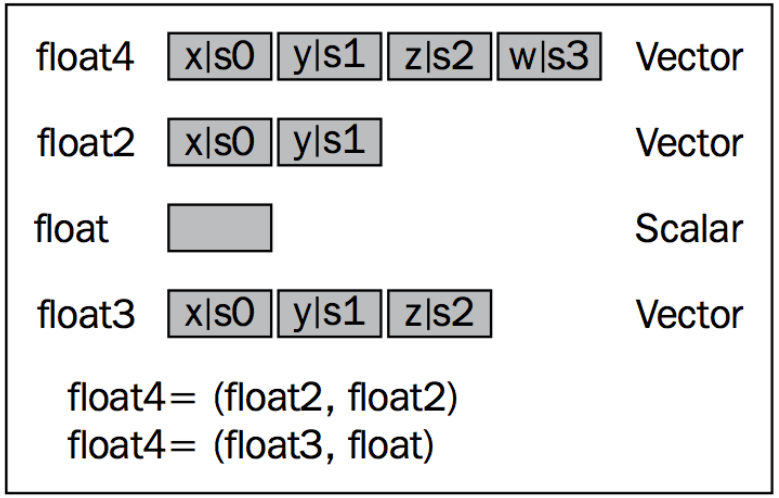
Типы данных: скаляры

Basic data types	Application data types
bool	n/a
char	cl_char
unsigned char, uchar	cl_uchar
short	cl_short
unsigned short, ushort	cl_ushort
int	cl_int
unsigned int, uint	cl_uint
long	cl_long
unsigned long, ulong	cl_ulong
float	cl_float
double	cl_double
half*	cl_half

Типы данных: векторы

Vector data types	Application vector data types
n/a	n/a
charn	cl_charn
ucharn	cl_ucharn
shortn	cl_shortn
ushortn	cl_ushortn
intn	cl_intn
uintn	cl_uintn
longn	cl_longn
ulongn	cl_ulongn
floatn	cl_floatn
doublen	cl_doublen
halfn	cl_halfn

```
float  f = 4.0f;
float3 f3 = (float3)(1.0f, 2.0f, 3.0f);
float4 f4 = (float4)(f3,  f);
//f4.x = 1.0f,
//f4.y = 2.0f,
//f4.z = 3.0f,
//f4.w = 4.0f
```



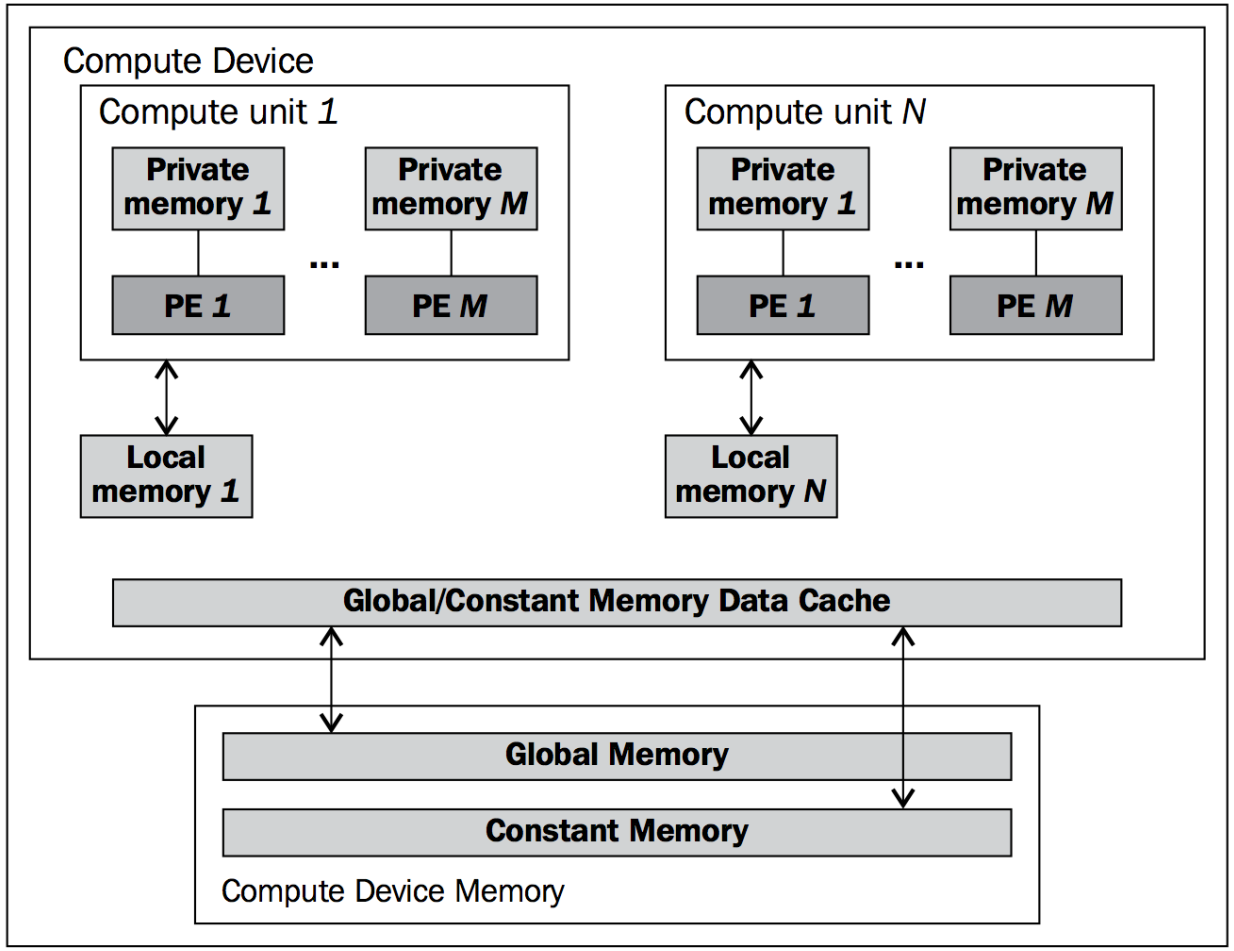
И как они там сохраняются?

И как они там сохраняются? Сложно...

Memory Model

- __global
- __constant
- __local
- __private

Memory Model



Но это еще не все...

Помните про SIMD?

Execution model

- У нас много данных

Execution model

- У нас много данных

- Над ними нужно проделать одну и ту же операцию

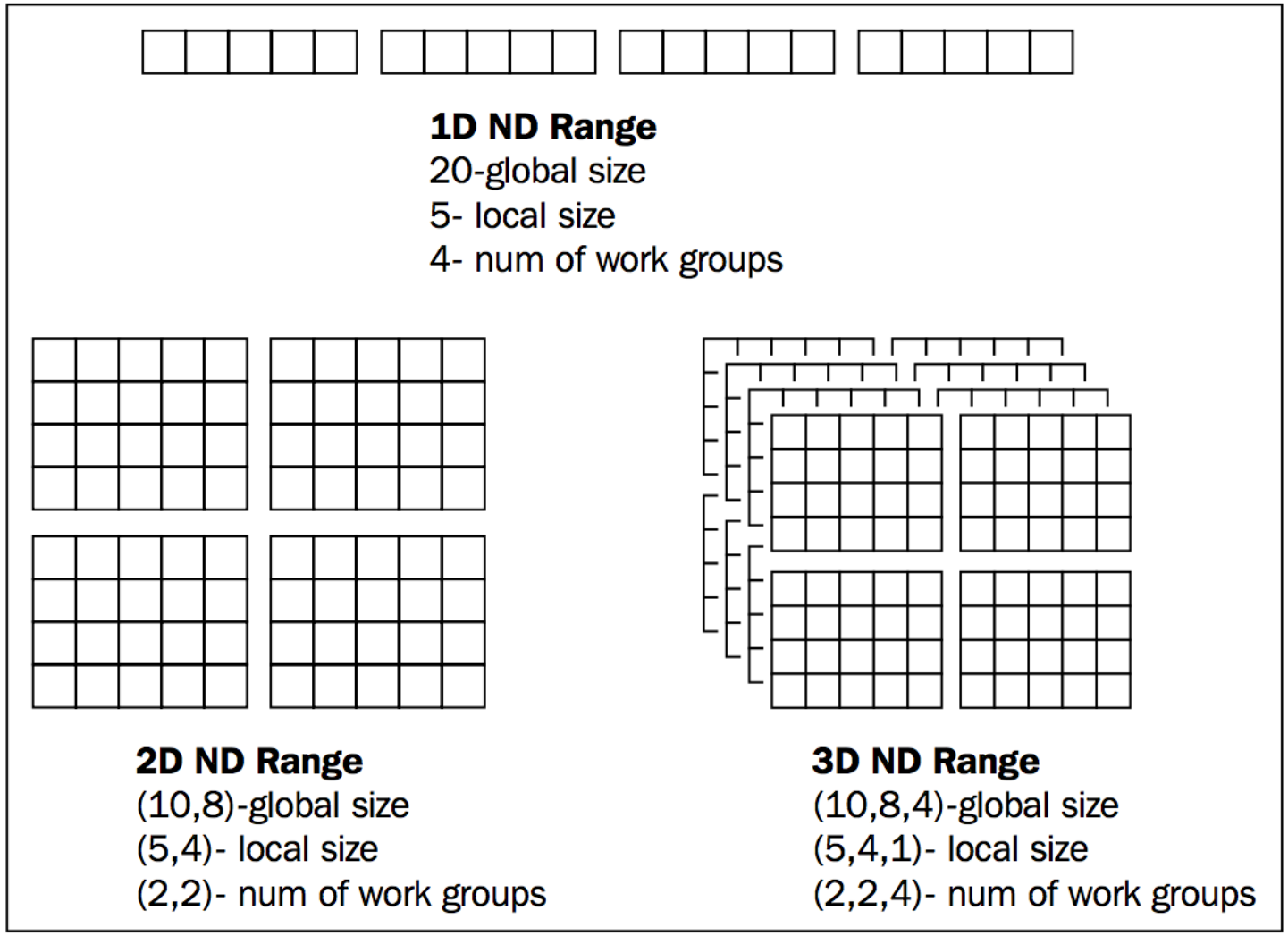
Execution model

- У нас много данных
- Над ними нужно проделать одну и ту же операцию
- Нам удобно из разделить на части и отдать каждую отдельному процессору..

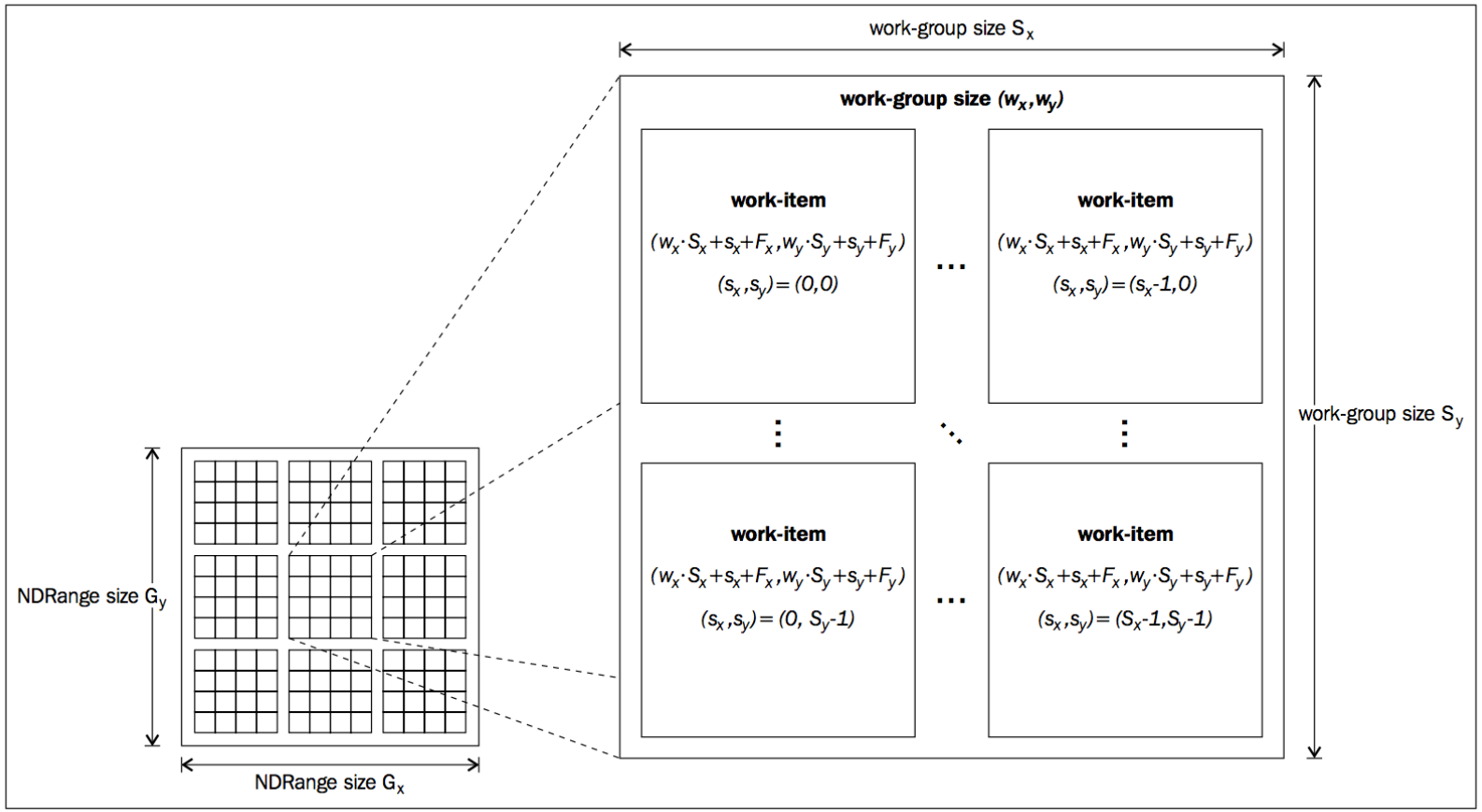
Execution model

- У нас много данных
- Над ними нужно проделать одну и ту же операцию
- Нам удобно из разделить на части и отдать каждую отдельному процессору..
- Тут нам OpenCL предоставляет подобную инфраструктуру

Execution model



ND Range - что это такое?

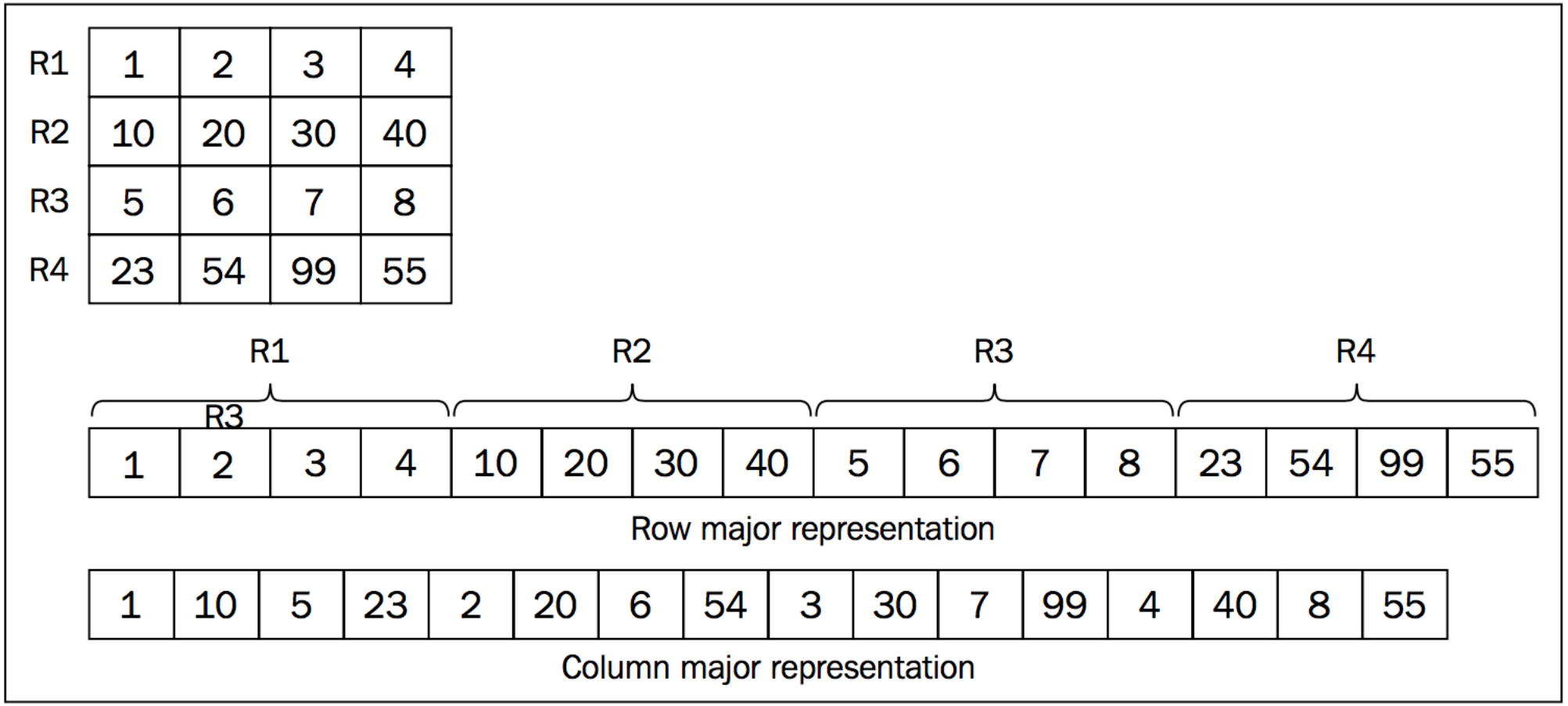


Например: умножаем матрицы

- Мы бы написали так наш код:

```
void MatrixMul_sequential(int dim, float *A, float *B, float *C) {
    for(int iRow=0; iRow<dim;++iRow) {
        for(int iCol=0; iCol<dim;++iCol) {
            float result = 0.f;
            for(int i=0; i<dim;++i) {
                result += A[iRow*dim + i]*B[i*dim + iCol];
            }
            C[iRow*dim + iCol] = result;
        }
    }
}
```

Например: умножаем матрицы



Например: умножаем матрицы

- На а на GPU:

```
void MatrixMul_kernel_basic(int dim,
                                    __global float *A, __global float *B, __global float *C) {
  //Get the index of the work-item
  int iCol = get_global_id(0);
  int iRow = get_global_id(1);
  float result = 0.0;
  for(int i=0;i< dim;++i) {
     result += A[iRow*dim + i]*B[i*dim + iCol];
  }
  C[iRow*dim + iCol] = result;
}
```

Например: умножаем матрицы

- На а на GPU:

```
void MatrixMul_kernel_basic(int dim,
                                    __global float *A, __global float *B, __global float *C) {
  //Get the index of the work-item
  int iCol = get_global_id(0);
  int iRow = get_global_id(1);
  float result = 0.0;
  for(int i=0;i< dim;++i) {
     result += A[iRow*dim + i]*B[i*dim + iCol];
  }
  C[iRow*dim + iCol] = result;
}
```

Типичное GPU

--- Info for device GeForce GT 650M: ---

CL_DEVICE_NAME:	GeForce GT 650M
CL_DEVICE_VENDOR:	NVIDIA
CL_DRIVER_VERSION:	10.14.20 355.10.05.15f03
CL_DEVICE_TYPE:	CL_DEVICE_TYPE_GPU
CL_DEVICE_MAX_COMPUTE_UNITS:	2
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:	3
CL_DEVICE_MAX_WORK_ITEM_SIZES:	1024 / 1024 / 64
CL_DEVICE_MAX_WORK_GROUP_SIZE:	1024
CL_DEVICE_MAX_CLOCK_FREQUENCY:	900 MHz
CL_DEVICE_ADDRESS_BITS:	64
CL_DEVICE_MAX_MEM_ALLOC_SIZE:	256 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:	1024 MByte
CL_DEVICE_ERROR_CORRECTION_SUPPORT:	no
CL_DEVICE_LOCAL_MEM_TYPE:	local
CL_DEVICE_LOCAL_MEM_SIZE:	48 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:	64 KByte
CL_DEVICE_QUEUE_PROPERTIES:	CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:	1
CL_DEVICE_MAX_READ_IMAGE_ARGS:	256
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:	16
CL_DEVICE_SINGLE_FP_CONFIG:	CL_FP_DENORM CL_FP_INF_NAN CL_FP_ROUND_TO_NEAREST CL_FP_ROUND_TO_ZERO CL_FP_ROUND_TO_INF CL_FP_CORRECTLY_ROUNDED_DIVIDE_SQRT
CL_DEVICE_2D_MAX_WIDTH	16384
CL_DEVICE_2D_MAX_HEIGHT	16384
CL_DEVICE_3D_MAX_WIDTH	2048
CL_DEVICE_3D_MAX_HEIGHT	2048
CL_DEVICE_3D_MAX_DEPTH	2048
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>	CHAR 1, SHORT 1, INT 1, LONG 1, FLOAT 1, DOUBLE 1

Типичное CPU

--- Info for device Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz: ---

CL_DEVICE_NAME:	Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz
CL_DEVICE_VENDOR:	Intel
CL_DRIVER_VERSION:	1.1
CL_DEVICE_TYPE:	CL_DEVICE_TYPE_CPU
CL_DEVICE_MAX_COMPUTE_UNITS:	8
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:	3
CL_DEVICE_MAX_WORK_ITEM_SIZES:	1024 / 1 / 1
CL_DEVICE_MAX_WORK_GROUP_SIZE:	1024
CL_DEVICE_MAX_CLOCK_FREQUENCY:	2600 MHz
CL_DEVICE_ADDRESS_BITS:	64
CL_DEVICE_MAX_MEM_ALLOC_SIZE:	2048 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:	8192 MByte
CL_DEVICE_ERROR_CORRECTION_SUPPORT:	no
CL_DEVICE_LOCAL_MEM_TYPE:	global
CL_DEVICE_LOCAL_MEM_SIZE:	32 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:	64 KByte
CL_DEVICE_QUEUE_PROPERTIES:	CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:	1
CL_DEVICE_MAX_READ_IMAGE_ARGS:	128
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:	8
CL_DEVICE_SINGLE_FP_CONFIG:	CL_FP_DENORM CL_FP_INF_NAN CL_FP_ROUND_TO_NEAREST CL_FP_ROUND_TO_ZERO CL_FP_ROUND_TO_INF CL_FP_FMA CL_FP_CORRECTLY_ROUNDED_DIVIDE_SQRT
CL_DEVICE_2D_MAX_WIDTH	8192
CL_DEVICE_2D_MAX_HEIGHT	8192
CL_DEVICE_3D_MAX_WIDTH	2048
CL_DEVICE_3D_MAX_HEIGHT	2048
CL_DEVICE_3D_MAX_DEPTH	2048
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>	CHAR 16, SHORT 8, INT 4, LONG 2, FLOAT 4, DOUBLE 2

Ну а что с CUDA?

Ну а что с CUDA?
Ну.. Оно, как бы, проще

Ну а что с CUDA?
Ну.. Оно, как бы, проще для С програмеров..

CUDA kernel

```
#define N   10

__global__ void add( int *a, int *b, int *c ) {
      int tid = blockIdx.x;    // this thread handles the data at its thread id
      if (tid < N)
      c[tid] = a[tid] + b[tid];
}
```

CUDA setup

```
int a[N], b[N], c[N];
int *dev_a, *dev_b, *dev_c;

// allocate the memory on the GPU
cudaMalloc( (void**)&dev_a, N * sizeof(int) );
cudaMalloc( (void**)&dev_b, N * sizeof(int) );
cudaMalloc( (void**)&dev_c, N * sizeof(int) );

// fill the arrays 'a' and 'b' on the CPU
for (int i=0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
}
```

CUDA copy to memory and run

```
   // copy the arrays 'a' and 'b' to the GPU
cudaMemcpy(dev_a, a, N *sizeof(int),
                                    cudaMemcpyHostToDevice);
                                        cudaMemcpy(dev_b,b,N*sizeof(int),
                                            cudaMemcpyHostToDevice);

add<<<N,1>>>(dev_a,dev_b,dev_c);

// copy the array 'c' back from the GPU to the CPU
cudaMemcpy(c,dev_c,N*sizeof(int),
cudaMemcpyDeviceToHost);
```

CUDA get results

```
// display the results
for (int i=0; i<N; i++) {
    printf( "%d + %d = %d\n", a[i], b[i], c[i] );
}

// free the memory allocated on the GPU
cudaFree( dev_a );
cudaFree( dev_b );
cudaFree( dev_c );
```

Но CUDA сильна в другом

- Cublas – про матрицы..
- JCufft – Fast Frontier Transformation
- Jcurand – про рандом
- JCusparse – про разряженные матрицы
- Jcusolver – факторизация и прочий ужас
- Jnvgraph – про графы
- Jcudpp – CUDA Data Parallel Primitives Library, даже про сортировку
- JNpp – обработка изображений на GPU
- Jcudnn – Deep Neural Network library (аж страшно)

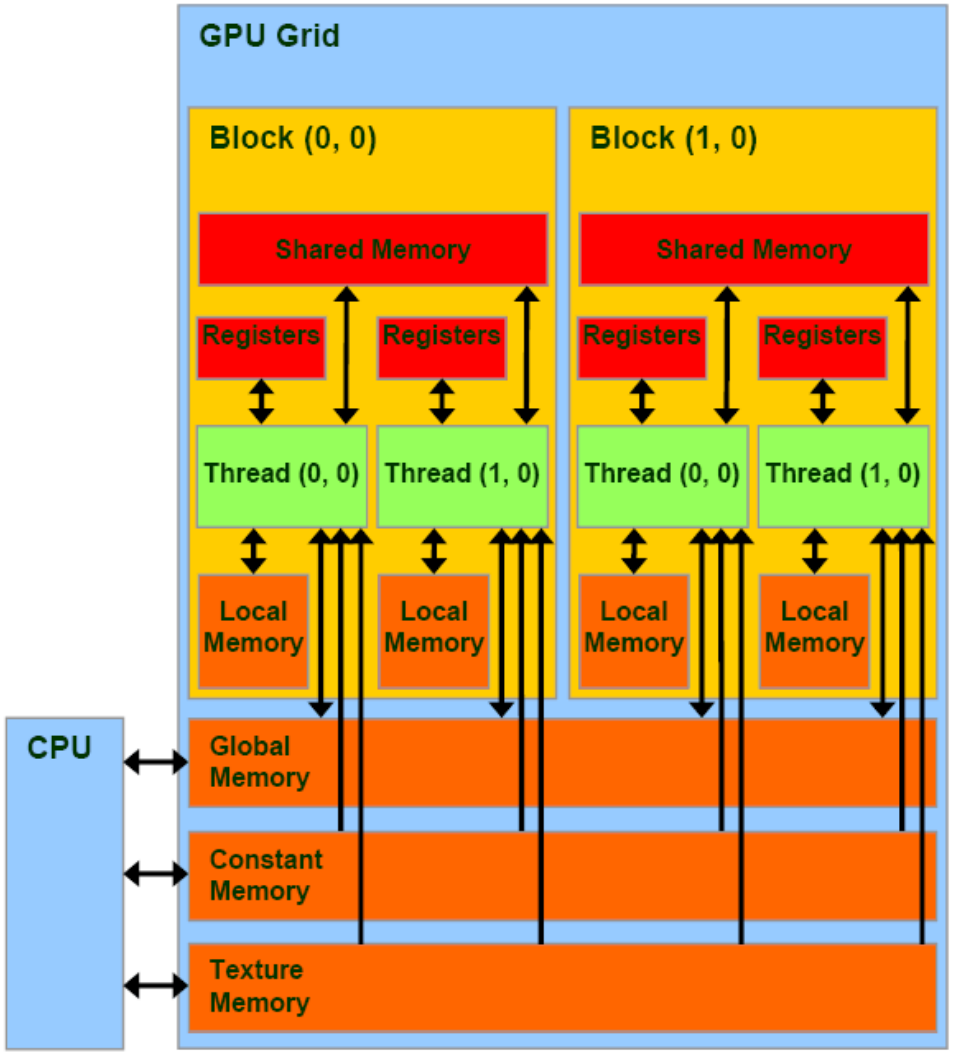
Например, нам нужен хороший rand

```
int n = 100;
curandGenerator generator = new curandGenerator();

float hostData[] = new float[n];

Pointer deviceData = new Pointer();
cudaMalloc(deviceData, n * Sizeof.FLOAT);

curandCreateGenerator(generator, CURAND_RNG_PSEUDO_DEFAULT);
curandSetPseudoRandomGeneratorSeed(generator, 1234);

curandGenerateUniform(generator, deviceData, n);

cudaMemcpy(Pointer.to(hostData), deviceData,
        n * Sizeof.FLOAT, cudaMemcpyDeviceToHost);

System.out.println(Arrays.toString(hostData));

curandDestroyGenerator(generator);
cudaFree(deviceData);
```

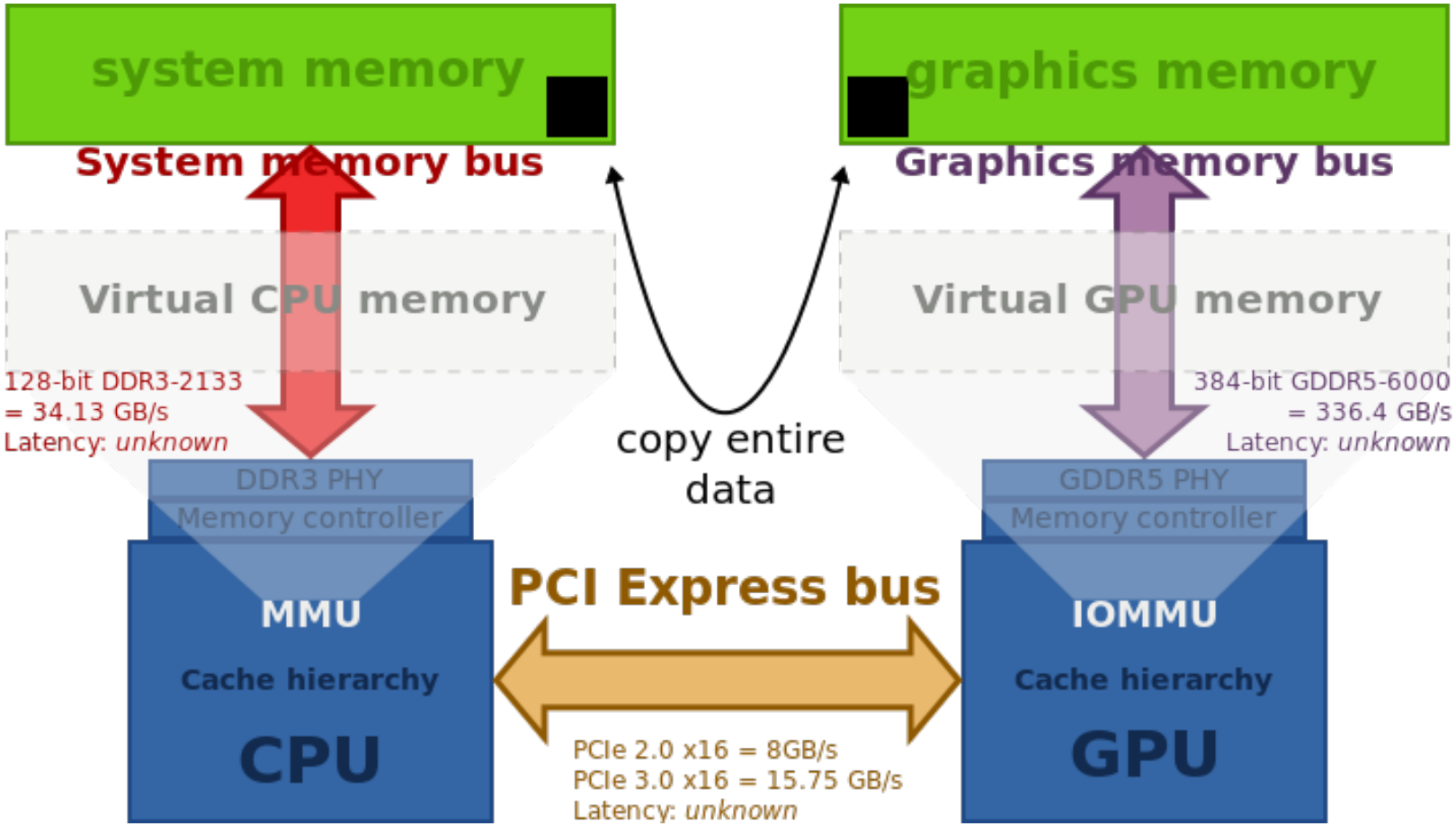
Например нам нужен хороший rand

- Приятно, что за этим всем есть сильная теория
- Разработанная нашим математиком Ильёй Мееровичем Соболем в 1967 г.
- https://en.wikipedia.org/wiki/Sobol_sequence

Память у nVidia выглядит примерно так:



Кстати о памяти…



©Wikipedia

Оптимизации…

```
__kernel void MatrixMul_kernel_basic(int dim,
                __global float *A,
                __global float *B,
                __global float *C){

   int iCol = get_global_id(0);
   int iRow = get_global_id(1);
   float result = 0.0;
   for(int i=0;i< dim;++i)
   {
      result +=
      A[iRow*dim + i]*B[i*dim + iCol];
   }
   C[iRow*dim + iCol] = result;
}
```

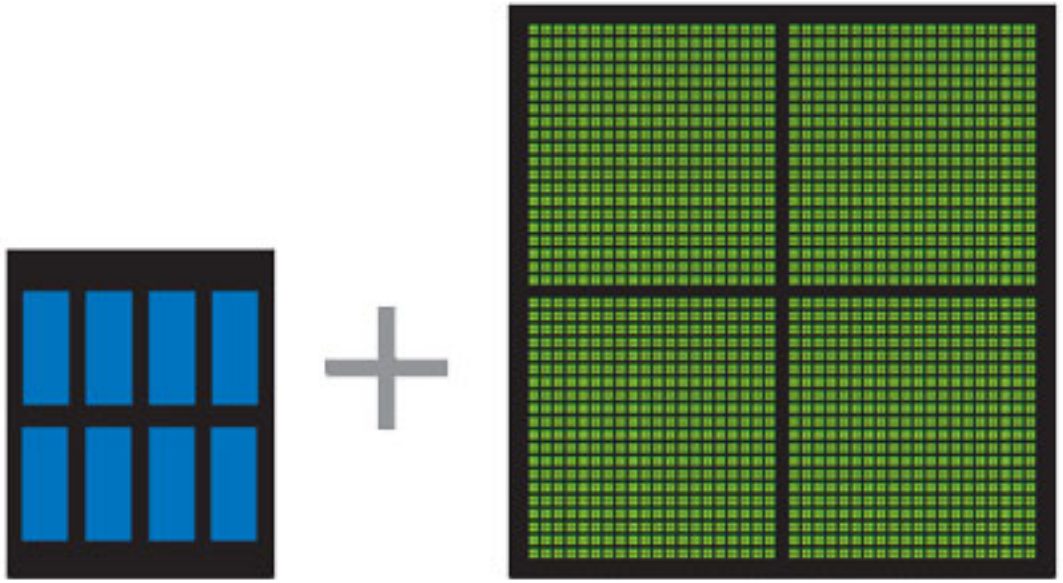
<<—Оптимизации...

```
#define VECTOR_SIZE 4
__kernel void MatrixMul_kernel_basic_vector4(int dim,
                __global float4 *A,
                __global float4 *B,
                __global float *C)

   int localIdx = get_global_id(0);
   int localIdy = get_global_id(1);
   float result = 0.0;
   float4 Bvector[4];
   float4 Avector, temp;
   float4 resultVector[4] = {0,0,0,0};
   int    rowElements = dim/VECTOR_SIZE;
   for(int i=0; i<rowElements; ++i){
      Avector = A[localIdy*rowElements + i];
      Bvector[0] = B[dim*i + localIdx];
      Bvector[1] = B[dim*i + rowElements + localIdx];
      Bvector[2] = B[dim*i + 2*rowElements + localIdx];
      Bvector[3] = B[dim*i + 3*rowElements + localIdx];
      temp = (float4)(Bvector[0].x, Bvector[1].x, Bvector[2].x, Bvector[3].x);
      resultVector[0] += Avector * temp;
      temp = (float4)(Bvector[0].y, Bvector[1].y, Bvector[2].y, Bvector[3].y);
      resultVector[1] += Avector * temp;
      temp = (float4)(Bvector[0].z, Bvector[1].z, Bvector[2].z, Bvector[3].z);
      resultVector[2] += Avector * temp;
      temp = (float4)(Bvector[0].w, Bvector[1].w, Bvector[2].w, Bvector[3].w);
      resultVector[3] += Avector * temp;
   }
   C[localIdy*dim + localIdx*VECTOR_SIZE] = resultVector[0].x + resultVector[0].y + resultVector[0].z + resultVector[0].w;
   C[localIdy*dim + localIdx*VECTOR_SIZE + 1] = resultVector[1].x + resultVector[1].y + resultVector[1].z + resultVector[1].w;
   C[localIdy*dim + localIdx*VECTOR_SIZE + 2] = resultVector[2].x + resultVector[2].y + resultVector[2].z + resultVector[2].w;
   C[localIdy*dim + localIdx*VECTOR_SIZE + 3] = resultVector[3].x + resultVector[3].y + resultVector[3].z + resultVector[3].w;
}
```

<<—Оптимизации...

```
#define VECTOR_SIZE 4
__kernel void MatrixMul_kernel_basic_vector4(int dim,
                __global float4 *A,
                __global float4 *B,
                __global float *C)

   int localIdx = get_global_id(0);
   int localIdy = get_global_id(1);
   float result = 0.0;
   float4 Bvector[4];
   float4 Avector, temp;
   float4 resultVector[4] = {0,0,0,0};
   int    rowElements = dim/VECTOR_SIZE;
   for(int i=0; i<rowElements; ++i){
      Avector = A[localIdy*rowElements + i];
      Bvector[0] = B[dim*i + localIdx];
      Bvector[1] = B[dim*i + rowElements + localIdx];
      Bvector[2] = B[dim*i + 2*rowElements + localIdx];
      Bvector[3] = B[dim*i + 3*rowElements + localIdx];
      temp = (float4)(Bvector[0].x, Bvector[1].x, Bvector[2].x, Bvector[3].x);
      resultVector[0] += Avector * temp;
      temp = (float4)(Bvector[0].y, Bvector[1].y, Bvector[2].y, Bvector[3].y);
      resultVector[1] += Avector * temp;
      temp = (float4)(Bvector[0].z, Bvector[1].z, Bvector[2].z, Bvector[3].z);
      resultVector[2] += Avector * temp;
      temp = (float4)(Bvector[0].w, Bvector[1].w, Bvector[2].w, Bvector[3].w);
      resultVector[3] += Avector * temp;
   }
   C[localIdy*dim + localIdx*VECTOR_SIZE] = resultVector[0].x + resultVector[0].y + resultVector[0].z + resultVector[0].w;
   C[localIdy*dim + localIdx*VECTOR_SIZE + 1] = resultVector[1].x + resultVector[1].y + resultVector[1].z + resultVector[1].w;
   C[localIdy*dim + localIdx*VECTOR_SIZE + 2] = resultVector[2].x + resultVector[2].y + resultVector[2].z + resultVector[2].w;
   C[localIdy*dim + localIdx*VECTOR_SIZE + 3] = resultVector[3].x + resultVector[3].y + resultVector[3].z + resultVector[3].w;
}
```

Можно все вместе



Ну а если совсем не хочется писать на C ...

И если не хочется думать про host и device и думать о памяти

Можно частично использовать GPU

Проект Sumatra

- Исследовательский проект

Проект Sumatra

- Исследовательский проект
- Заточен под Java 8

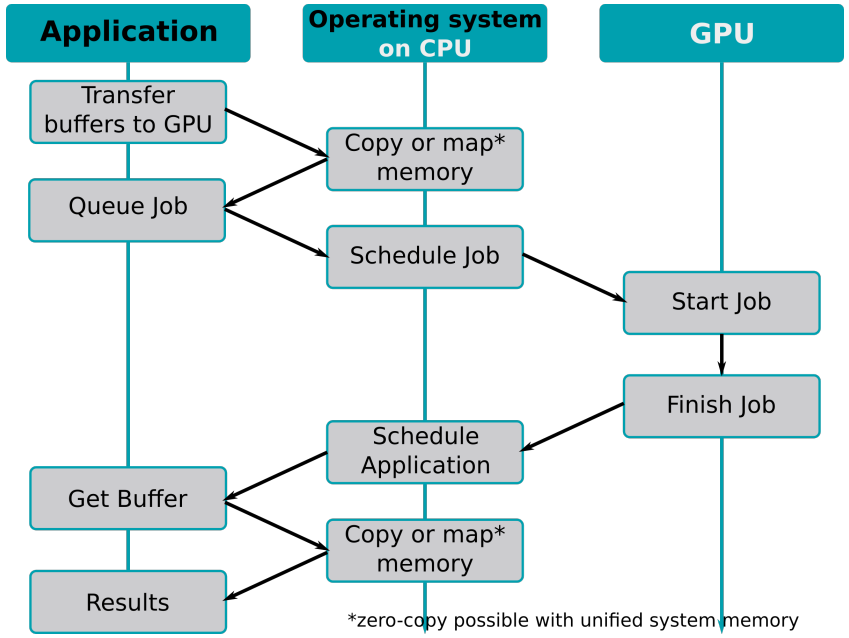
Проект Sumatra

- Исследовательский проект
- Заточен под Java 8
- … а точнее под Stream-ы

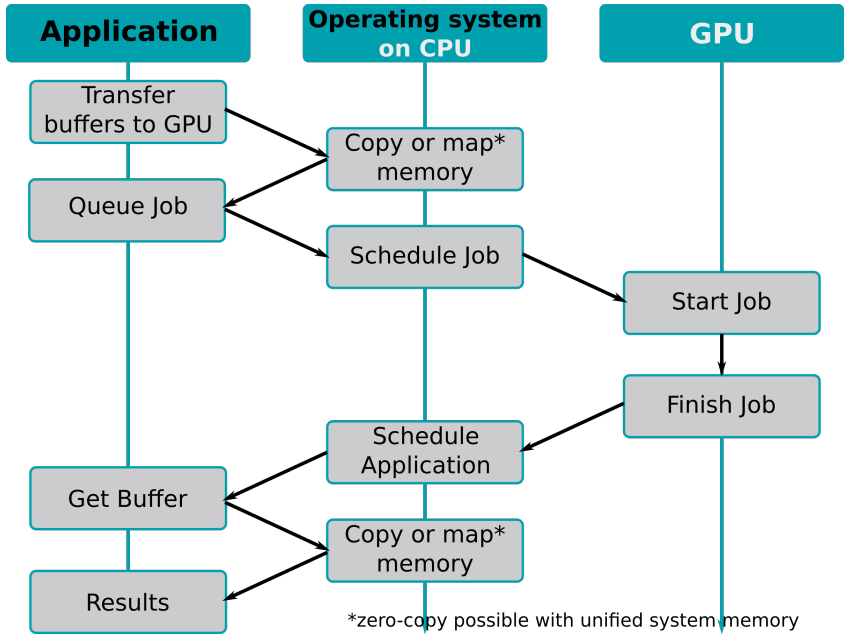
Проект Sumatra

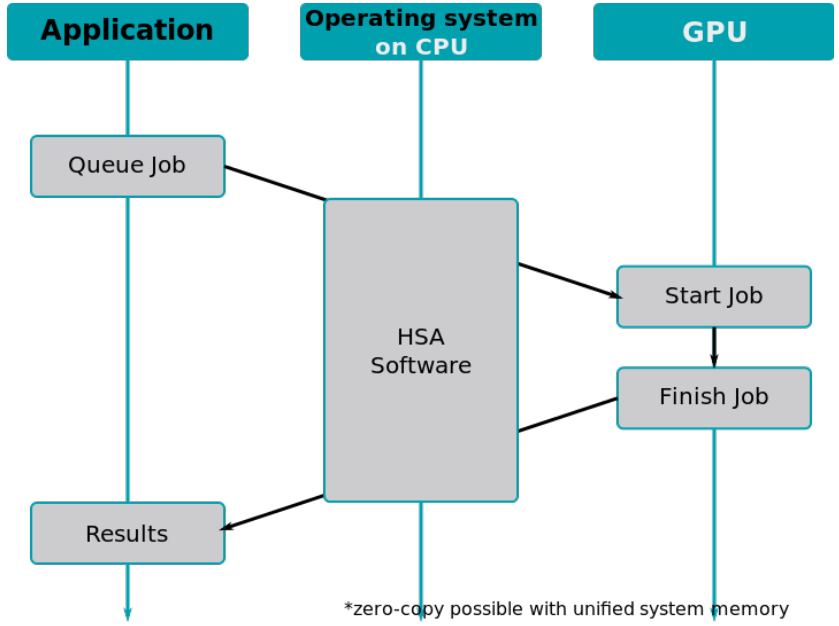
- Исследовательский проект
- Заточен под Java 8
- … а точнее под Stream-ы
- … а совсем точно под лямбды и .forEach()

AMD HSAIL


Application
Operating system on CPU
GPU
Transfer buffers to GPU
Copy or map* memory
Queue Job
Schedule Job
Start Job
Finish Job
Schedule Application
Get Buffer
Copy or map* memory
Results
*zero-copy possible with unified system memory

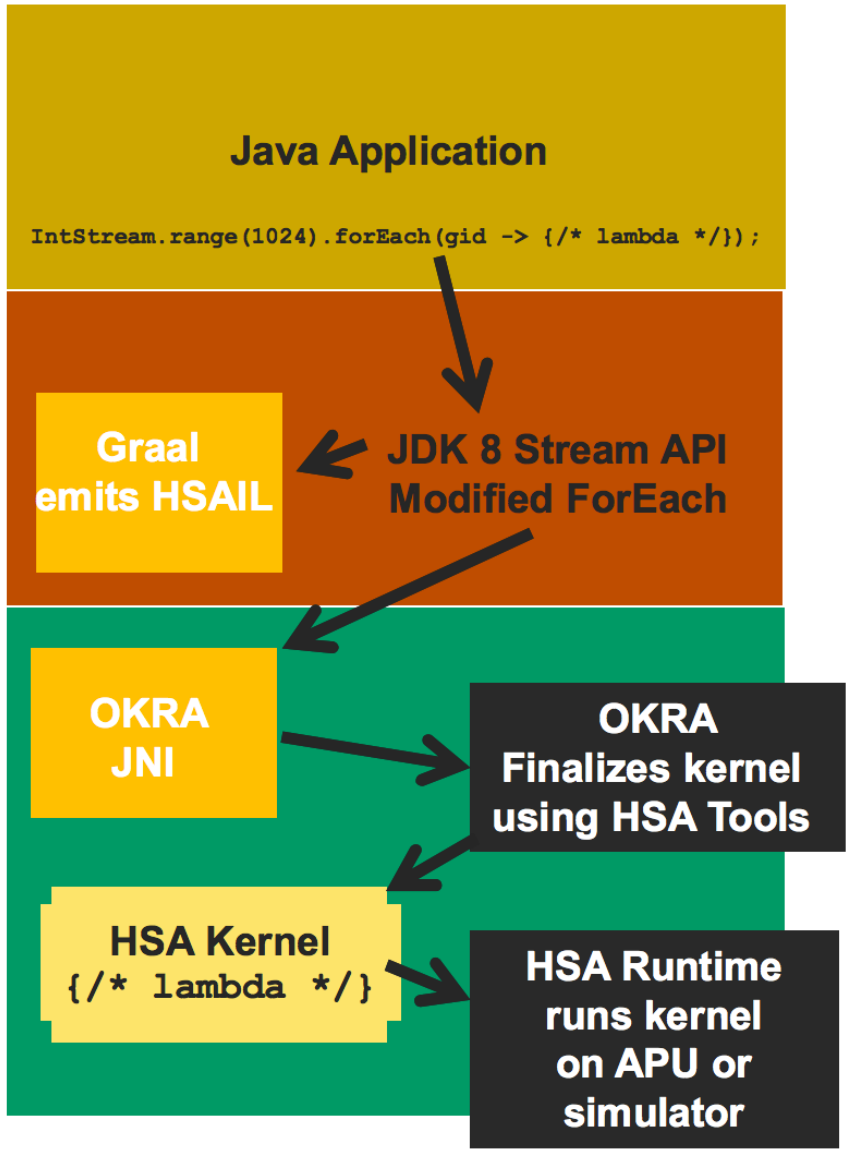
AMD HSAIL


Application
Operating system on CPU
GPU
Transfer buffers to GPU
Copy or map* memory
Queue Job
Schedule Job
Start Job
Finish Job
Schedule Application
Get Buffer
Copy or map* memory
Results
*zero-copy possible with unified system memory

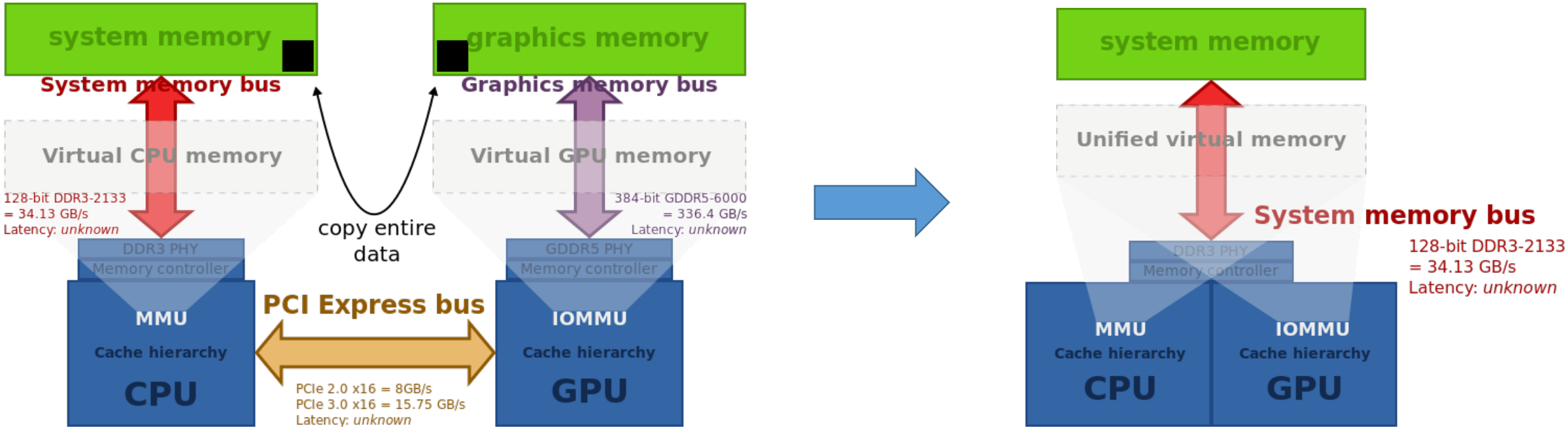

Application
Operating system on CPU
GPU
Queue Job
HSA Software
Start Job
Finish Job
Results
*zero-copy possible with unified system memory

AMD HSAIL

- Распознает блок forEach()
- Через Graal получаем HSAIL
- На низком уровне передаем переделанную лямбду как kernel



AMD APU пытается решить основную проблему!



©Wikipedia

Ну а если не под AMD и экзотику..

IBM patched JVM for GPU

- Решили сосредоточиться исключительно на CUDA (пока)

IBM patched JVM for GPU

- Решили сосредоточиться исключительно на CUDA (пока)
- Сосредоточились на Stream API

IBM patched JVM for GPU

- Решили сосредоточиться исключительно на CUDA (пока)
- Сосредоточились на Stream API
- Решили дописать свой обработчик parallel()

IBM patched JVM for GPU

Представьте себе:

```
void fooJava(float A[], float B[], int n) {
    // similar to for (idx = 0; i < n; i++)
    IntStream.range(0, N).parallel().forEach(i -> { b[i] = a[i] * 2.0; });
}
```

IBM patched JVM for GPU

Представьте себе:

```
void fooJava(float A[], float B[], int n) {
    // similar to for (idx = 0; i < n; i++)
    IntStream.range(0, N).parallel().forEach(i -> { b[i] = a[i] * 2.0; });
}
```

… хотелось бы чтобы автоматически конвертировать…

IBM patched JVM for GPU

При крупном n код в лямбде исполняется на GPU:

```
class Par {
    void foo(float[] a, float[] b, float[] c, int n) {
        IntStream.range(0, n).parallel()
                 .forEach(i -> {
                     b[i] = a[i] * 2.0;
                     c[i] = a[i] * 3.0;
                 });
    }
}
```

*пока в лямбдах с одномерными массивами из примитивов.

IBM patched JVM for GPU

Оптимизации IBM JIT компилятора:

- Использование read-only cache
  - Уменьшение количества копирования данных в глобальную память GPU
- Оптимизация копирования данных из Host в Device
  - Уменьшения количества данных
- Элиминирование лишних проверок эксепшанов
  - В Kernel-е GPU

IBM patched JVM for GPU

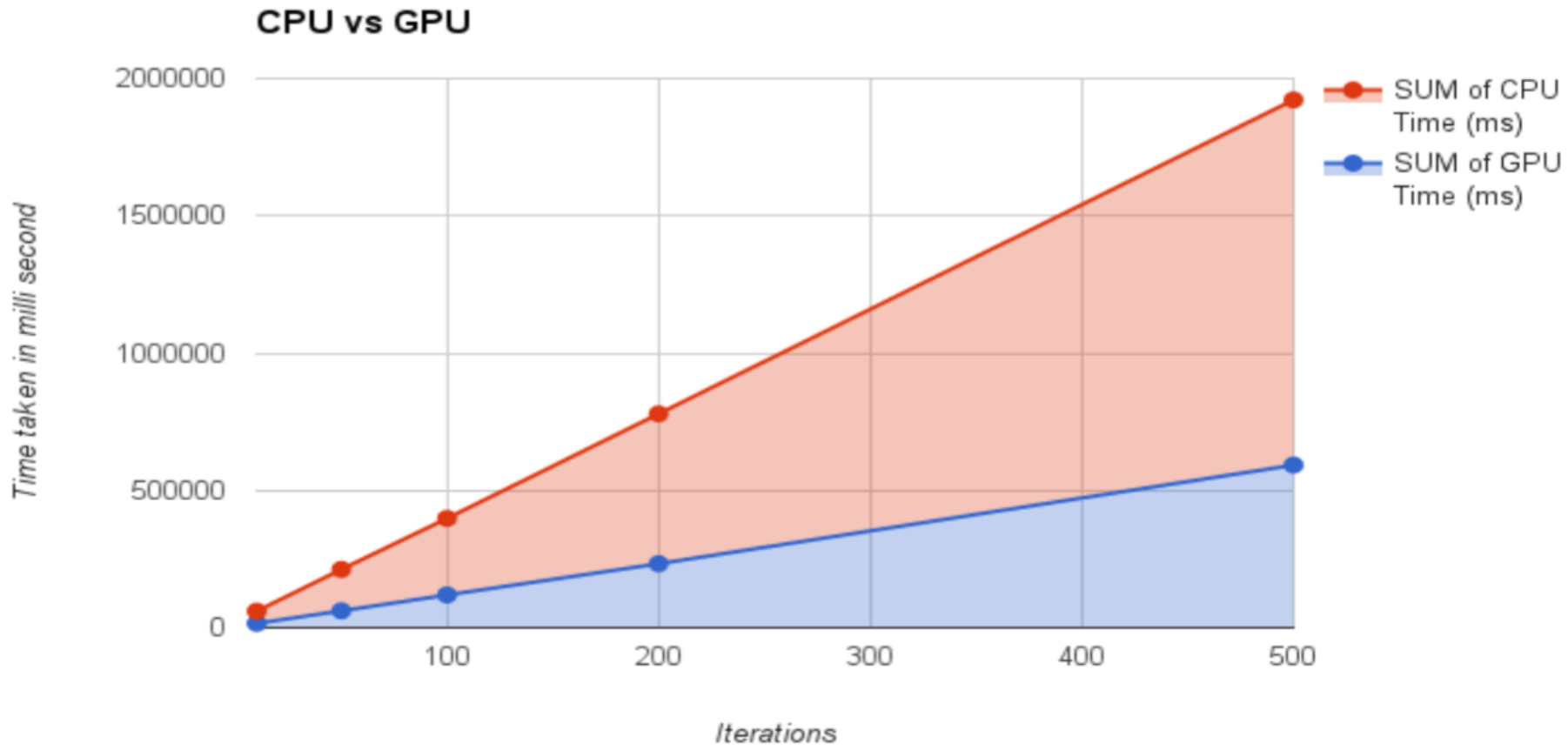
- Success story:

+

+

IBM patched JVM for GPU

- Утверждают что:



IBM patched JVM for GPU

- Больше инфо:

https://github.com/IBMSparkGPU/GPUEnabler

А нельзя прям писать на Java, но чтобы было как в OpenCL/CUDA?

Можно!

И звать сие Aparapi!

Aparapi

- Сокращение «A PARallel API»

Aparapi

- Сокращение «A PARallel API»
- Почти как Hibernate для баз данных

Aparapi

- Сокращение «A PARallel API»
- Почти как Hibernate для баз данных
- Динамически конвертирует JVM Bytecode в код для Host и Device

Aparapi

- Сокращение «A PARallel API»
- Почти как Hibernate для баз данных
- Динамически конвертирует JVM Bytecode в код для Host и Device
- На основе OpenCL

Aparapi

- Начата AMD

Aparapi

- Начата AMD
- Потом запущено…

Aparapi

- Начата AMD
- Потом запущено…
- Через 5 лет отдано в Opensourse под Apache 2.0 license

Aparapi

- Начата AMD
- Потом запущено…
- Через 5 лет отдано в Opensourse под Apache 2.0 license
- Опять живое!!!

Aparapi - все стало намного проще!

```
public static void main(String[] _args) {
    final int size = 512;
    final float[] a = new float[size];
    final float[] b = new float[size];
    for (int i = 0; i < size; i++) {
        a[i] = (float) (Math.random() * 100);
        b[i] = (float) (Math.random() * 100);
    }
    final float[] sum = new float[size];

    Kernel kernel = new Kernel(){
        @Override public void run() {
            int gid = getGlobalId();
            sum[gid] = a[gid] + b[gid];
        }
    };

    kernel.execute(Range.create(size));
    for (int i = 0; i < size; i++) {
        System.out.printf("%6.2f + %6.2f = %8.2f\n", a[i], b[i], sum[i]);
    }
    kernel.dispose();
}
```

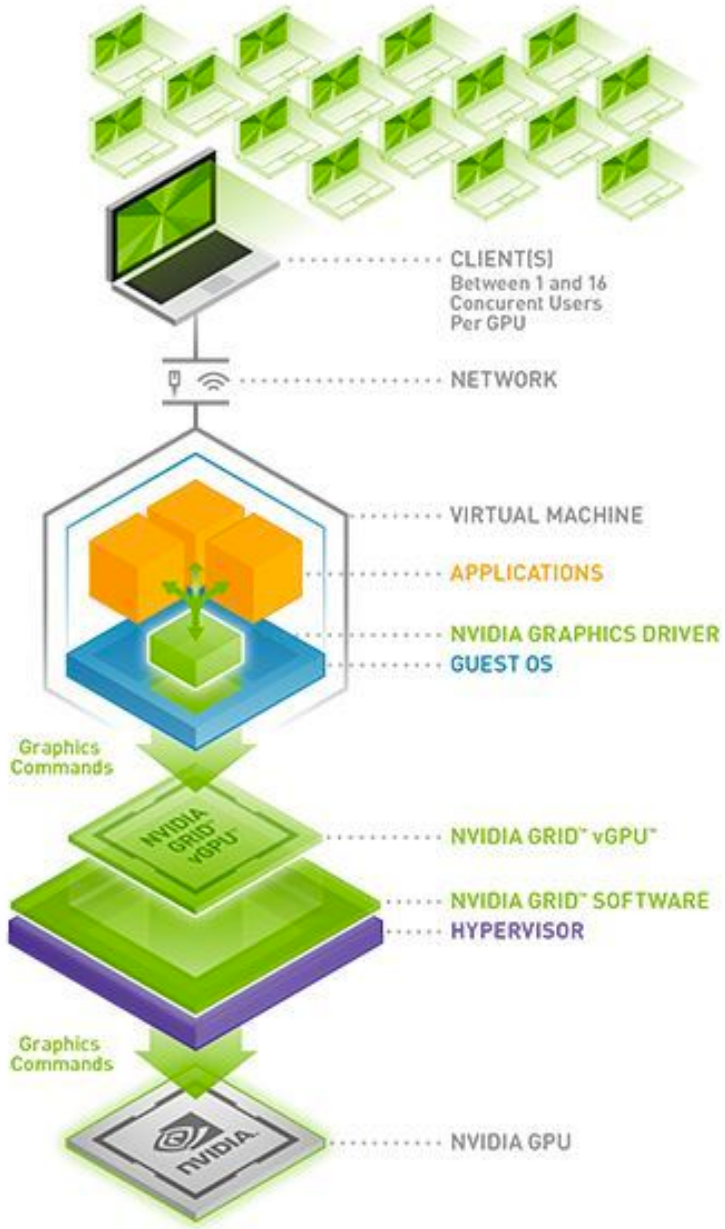
Ну а как же облака?

Без слова cloud нам наш софт не продать…

nVidia вам в помощь

nVidia GRID

- Анонс в 2012 г.
- Уже референтная
- Работает на большинстве гипервизоров
- Как и в облаках

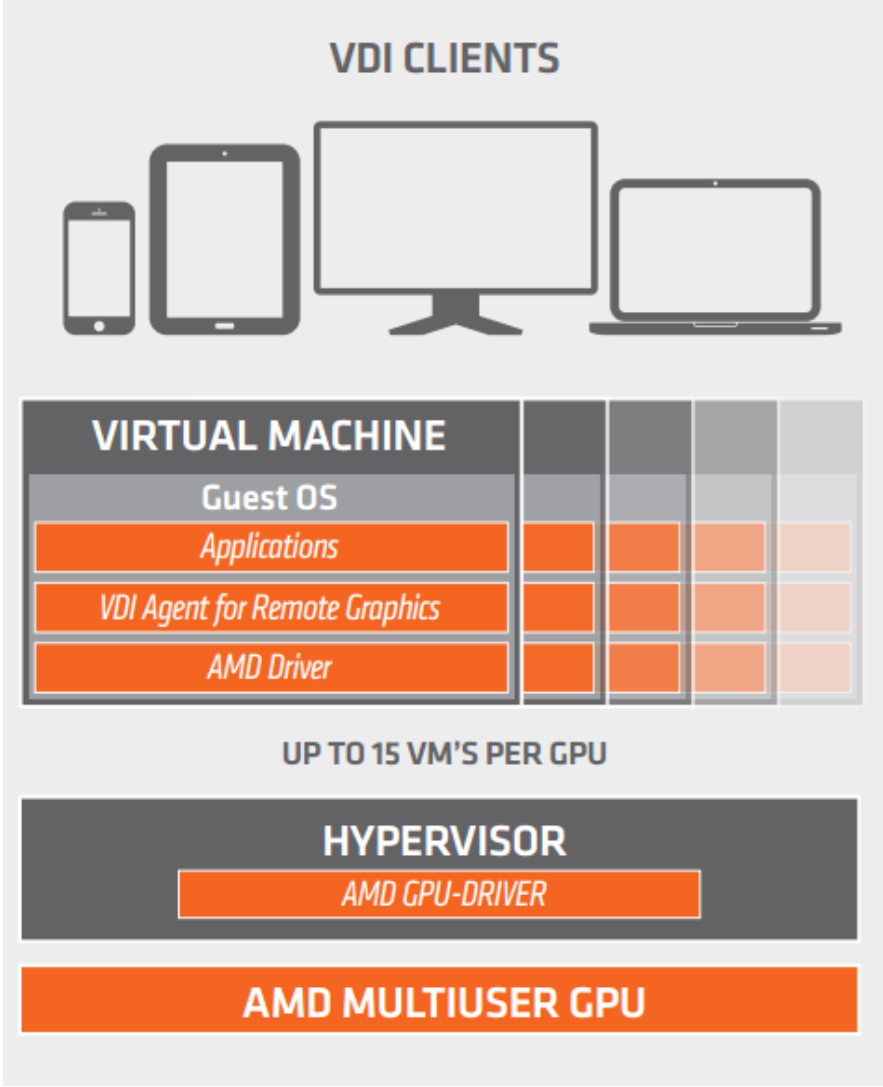


nVidia GRID

	TESLA M10	TESLA M6	TESLA M60
Number of GPUs	Quad Mid-Level Maxwell	Single High-End Maxwell	Dual High-End Maxwell
Total NVIDIA CUDA® Cores	2,560 (640 per GPU)	1,536	4,096 (2,048 per GPU)
Total Memory Size	32 GB GDDR5 (8 GB per GPU)	8 GB GDDR5	16 GB GDDR5 (8 GB per GPU)
Max vGPU instances	64	16	32
Form Factor	PCIE 3.0 Dual slot (rack servers)	MXM (blade servers)	PCIE 3.0 Dual slot (rack servers)
Power	225 W	100 W (75 W opt)	240 W / 300 W (225 W opt)
Cooling Solution	Passive	Bare Board	Active / Passive
Board Dimensions	10.5" x 4.4"	3.2" x 4.1"	10.5" x 4.4"
	USER DENSITY optimized	**BLADE** optimized	**PERFORMANCE** optimized

nVidia GRID

… а AMD пока отстают…



И тем не менее:
оно уже тут!

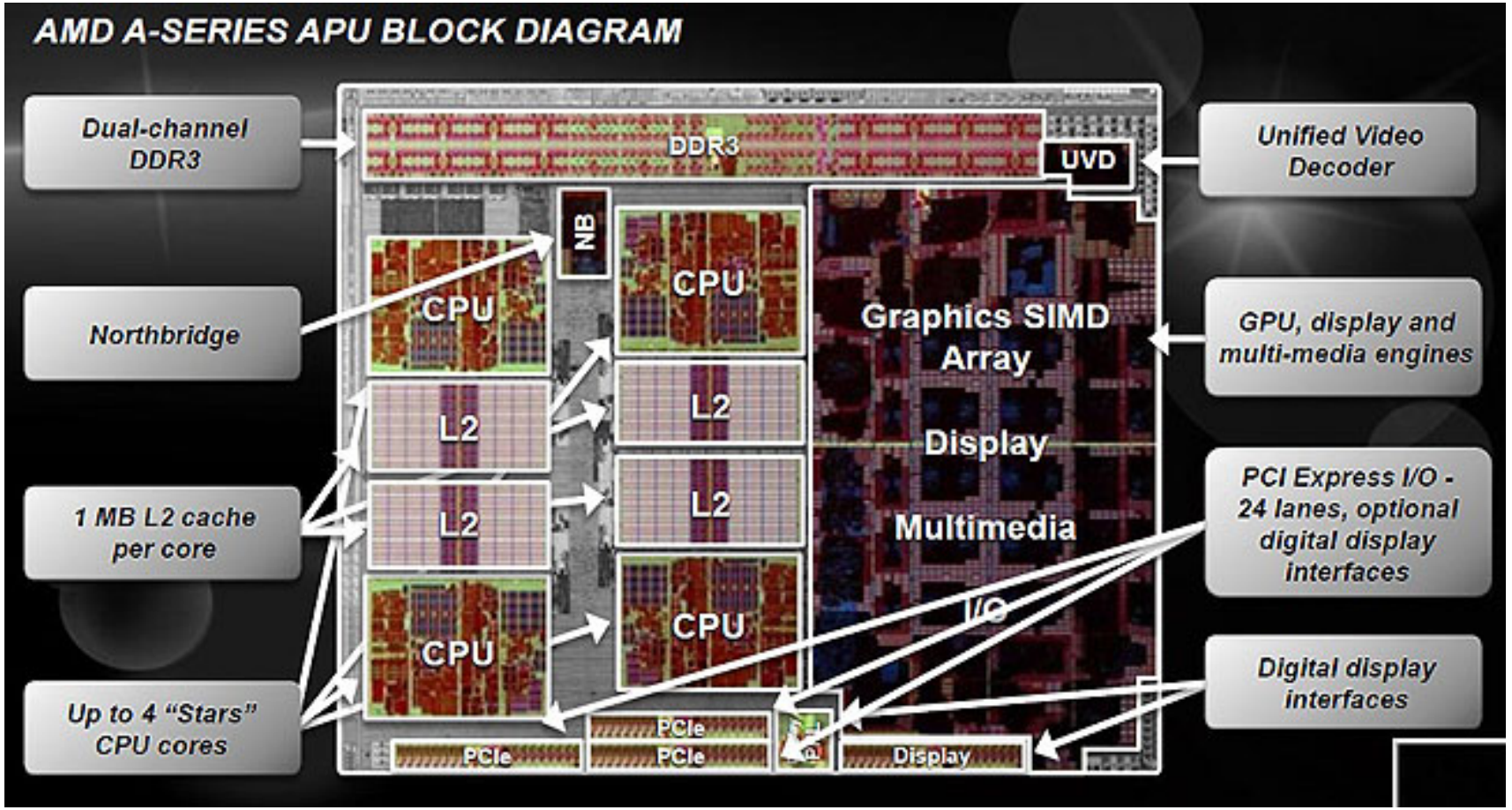
Оно уже есть: Nvidia GPU

	GTX 1080	GTX 980 Ti	GTX 980	GTX 780
CUDA Cores	2560	2816	2048	2304
Texture Units	160	176	128	192
ROPs	64	96	64	48
Core Clock	1607 MHz	1000 MHz	1126 MHz	863 MHz
Boost Clock	1733 MHz	1075 MHz	1216 MHz	900 MHz
Mem Clock	10Gbs GDDR5X	7Gbps GDDR5	7Gbps GDDR5	6Gbps GDDR5
Mem Bus Width	256-bit	256-bit	256-bit	256-bit
VRAM	8GB	6GB	4GB	3GB
TDP	180W	250W	165W	250W
GPU	GP104	GM200	GM204	GK110
Manufct. Process	TSMC 16nm	TSMC 28nm	TSMC 28nm	TSMC 28nm
Transistors	7.2 Bn	8 Bn	5.2 Bn	7.1 Bn

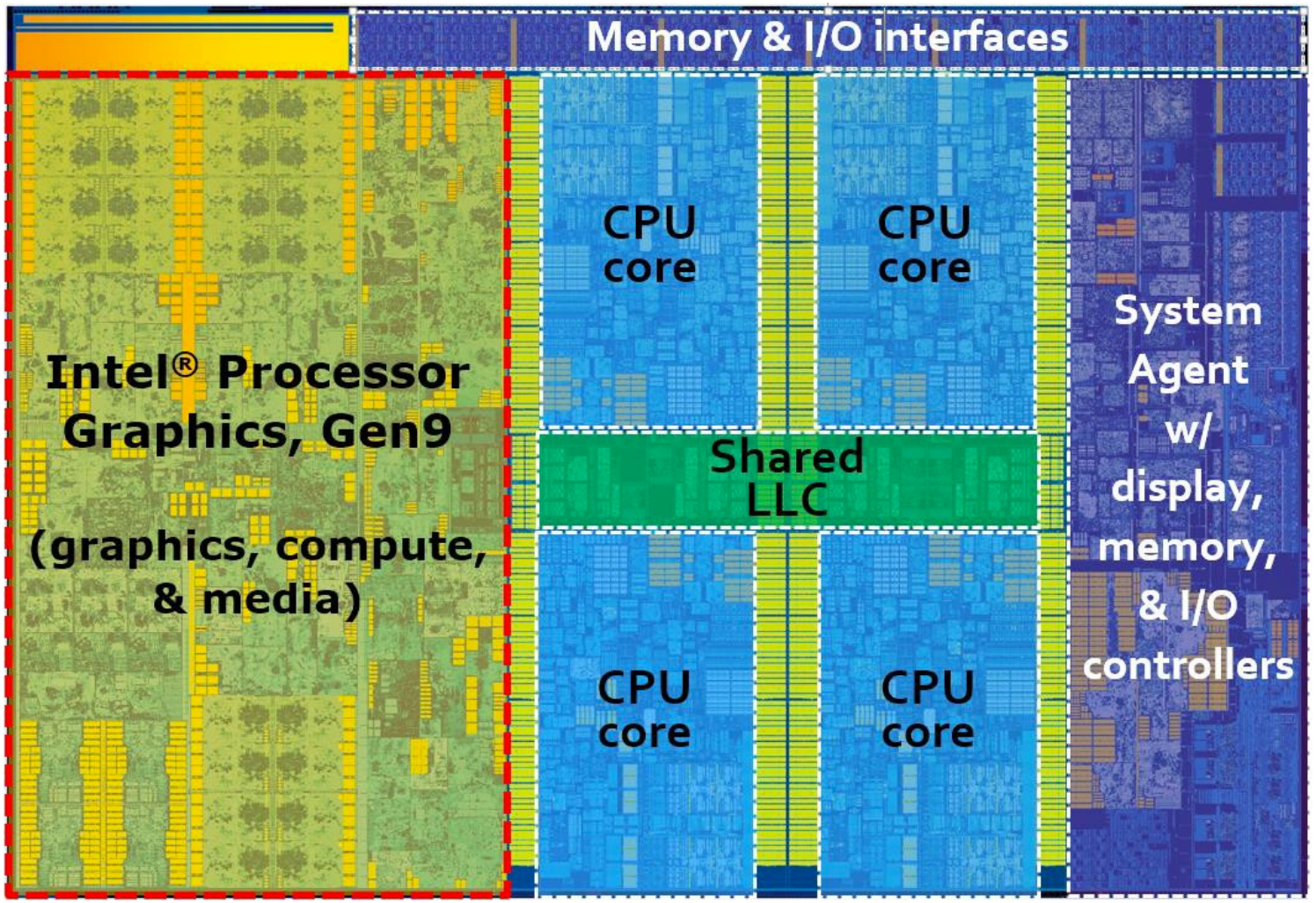
Оно уже есть: ATI Radeon

	Radeon RX 480	Radeon RX 470	Radeon RX 460
GCN Architecture	4th Gen	4th Gen	4th Gen
Compute Units	36	32	16
Stream Processors	2304	?2048?	?1024?
Texture Units	144	128	64
Clock Speeds (Boost / Base)	1266 MHz / 1120 MHz	N/A	N/A
Peak Performance	Up to 5.8 TFLOPS	Up to 4 TFLOPS	Up to 2 TFLOPS
Memory Size	4/8 GB	4 GB	2GB
Memory Bandwidth	224 GB/s or higher	N/A	N/A
Memory Interface	256-bit	256-bit	128-bit
Memory Type	GDDR5	GDDR5	GDDR5
Board Power	150W	?150W?	N/A
Power connector	1x6-pin	1x6-pin	none
AMD FreeSync	Yes	Yes	Yes
DirectX 12	Yes	Yes	Yes
Vulkan	Yes	Yes	Yes
VR Premium	Yes	N/A	N/A
DisplayPort Version	1.3 / 1.4 HDR	1.3 / 1.4 HDR	1.3 / 1.4 HDR

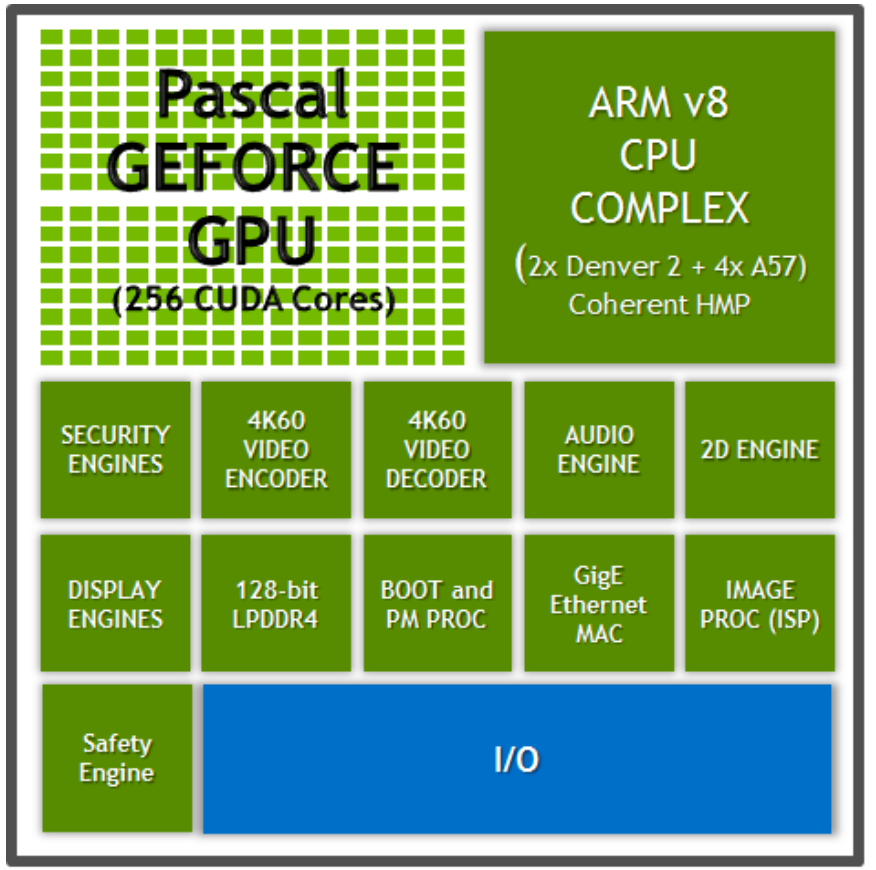
Оно уже есть: AMD APU



Оно уже есть: Intel Skylake

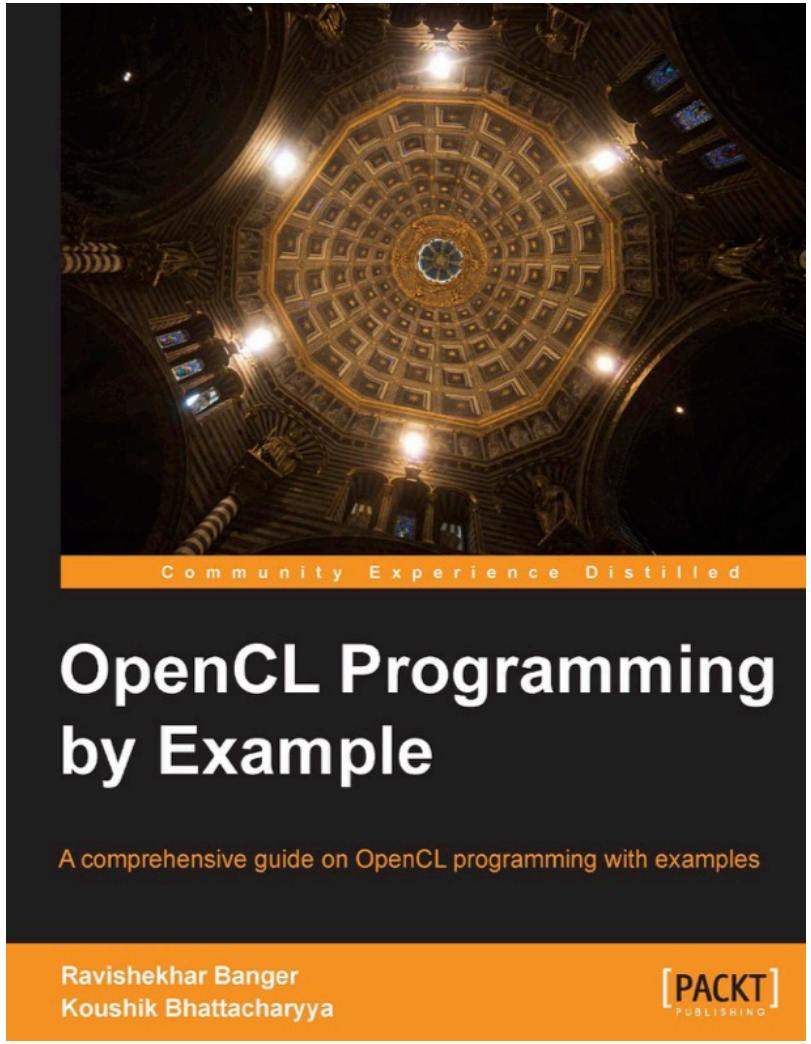


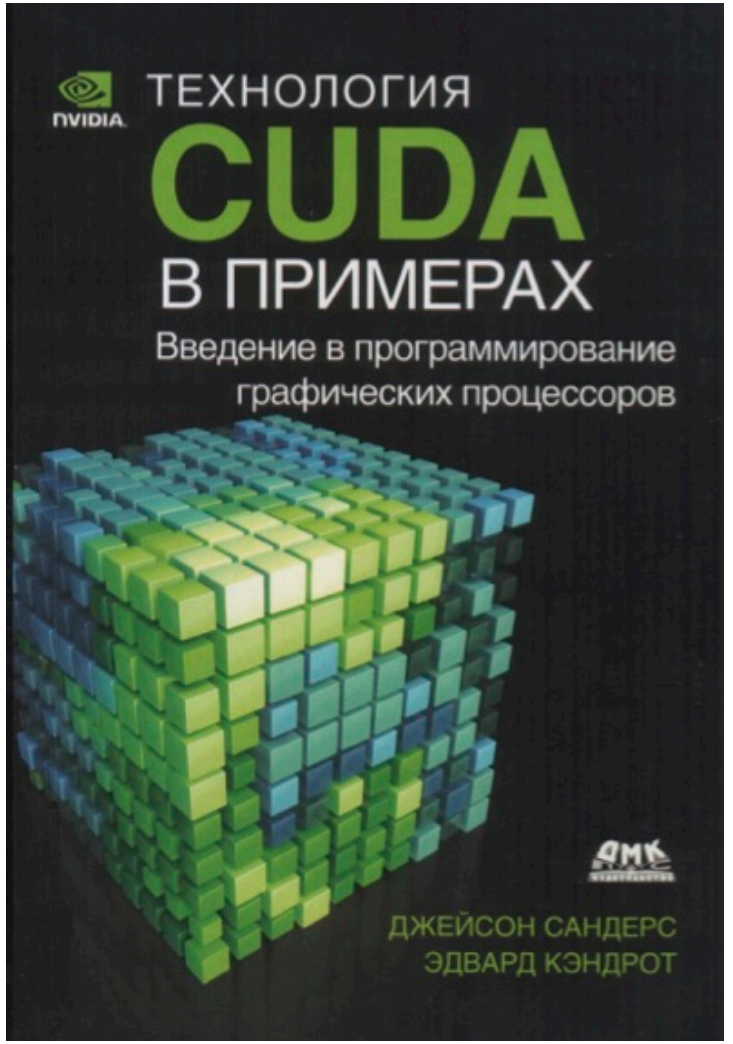
Оно уже есть: Nvidia Tegra Parker



Parker's rich feature set

Но сначала почитайте:





Берите и пользуйтесь!

Берите и пользуйтесь!
Если задача подходящая.

… и это сложно!
Вас предупредили!

Но вы станете круче всех!


ПАПА НЕ МАЙНИ

Спасибки! ☺