

Анатолий Свириденков
(codedgers.com)

Блог: http://bit.ly/cuda_blog



Проблематика

Где нужна вычислительная мощность:

- Ускорение вычислений
- Переход в реальное время
- Разгрузка **CPU**
- Улучшение качества

HD TV



Чего ожидать от параллелизма

Закон Амдала (ускорение от параллелизма):

$$Sp = 1 / (a + (1 - a) / p)$$

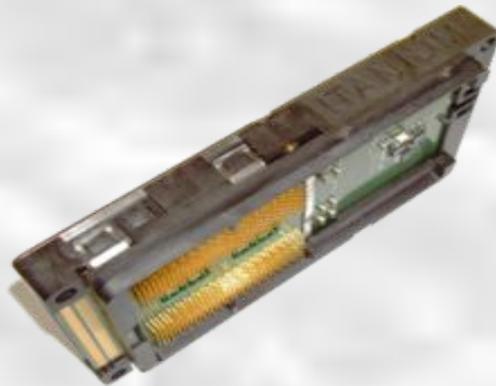
p – количество потоков

a – доля последовательных вычислений

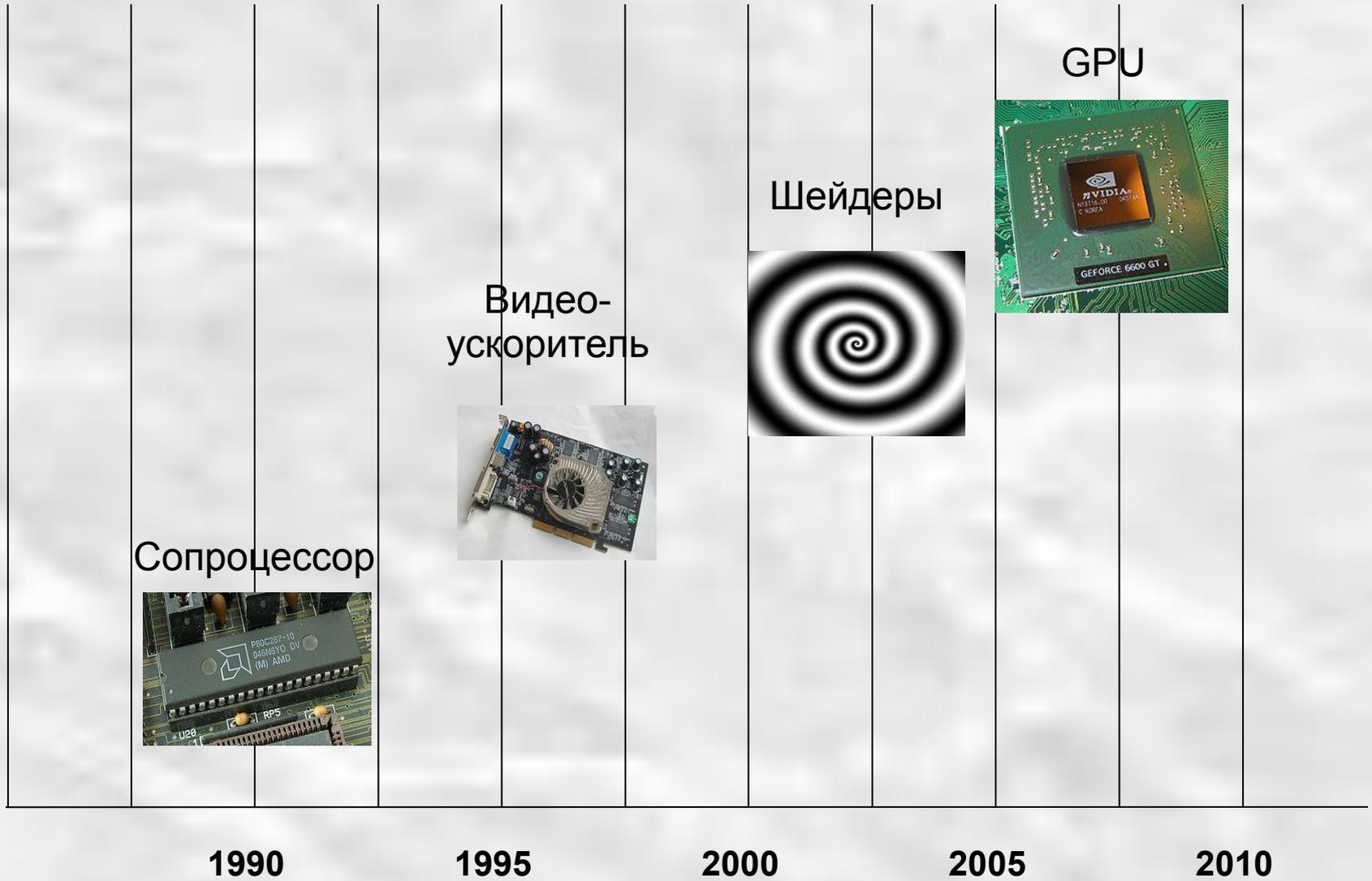
| | $p = 2$ | $p = 8$ | $p = 1000$ |
|-----------|---------|---------|------------|
| $a = 0,9$ | 1,05 | 1,1 | 1,11 |
| $a = 0,5$ | 1,33 | 1,77 | 2,0 |
| $a = 0,1$ | 1,81 | 4,71 | 9,91 |

Примеры параллелизма

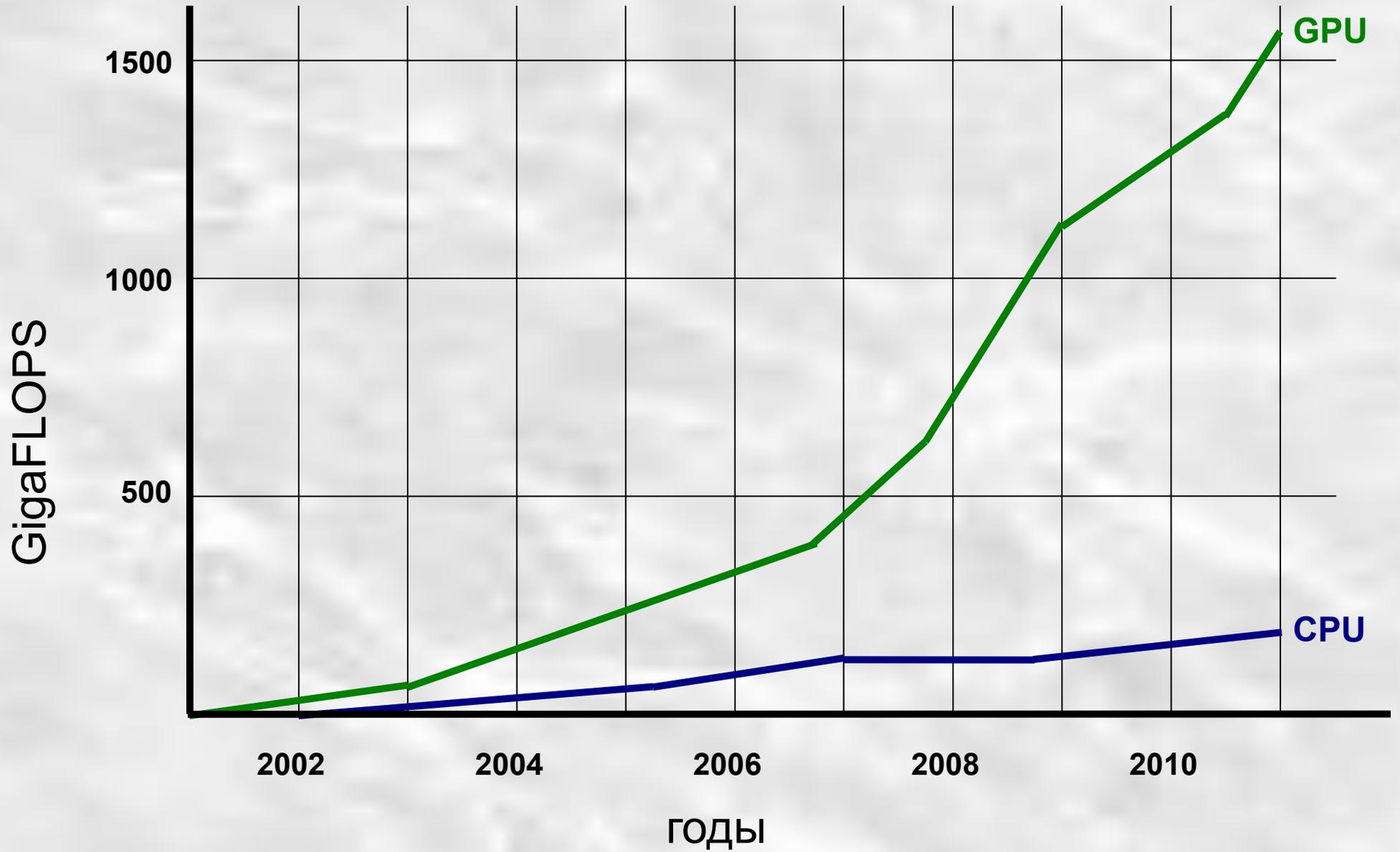
- Параллелизм данных, DPL: MMX, SSE, и т. д.
- Параллелизм кода, IPL: спекулятивные вычисления и конвейер, VLIW
- Квази многопоточность, многоядерность, hyper threading
- Кластеры



Предыстория к GPGPU



Рост производительности



Терминология

- host - **CPU**
 - device - **GPU**
 - **ядро** — код запускаемого на GPU из основного приложения
 - **поток** — часть вычислений исполняемых параллельно
 - **сетка (grid)** — все множество потоков для одного ядра
 - **блок** — набор потоков исполняемых на одном SM
 - **warp** — набор потоков физически исполняемых параллельно
-
-

Программная модель памяти

| Тип | Доступ | Расположение | Латентность |
|-------------|--------------------------|---------------|---------------------------|
| Регистры | Поток GPU (R\W) | SM | 2-4 такта |
| Локальная | Поток GPU (R\W) | DRAM | ~500 тактов |
| Разделяемая | Блок потоков GPU (R\W) | SM | 2-4 такта |
| Глобальная | CPU(R\W), GPU(R\W) | DRAM | ~500 тактов |
| Константная | CPU(R\W), GPU(Read-only) | DRAM + КЕШ SM | ~500 к DRAM 2-4 к кешу |
| Текстурная | CPU(R\W), GPU(Read-only) | DRAM + КЕШ SM | ~500 к DRAM 2-4 к кешу |

Программная модель потоков

Grid

| | | | | |
|------------|------------|------------|------------|------------|
| B (0:0) | B (1:0) | B (2:0) | B (3:0) | B (4:0) |
| B (0:1) | B (1:1) | B (2:1) | B (3:1) | B (4:1) |
| B (0:2) | B (1:2) | B (2:2) | B (3:2) | B (4:2) |
| B (0:3) | B (1:3) | B (2:3) | B (3:3) | B (4:3) |

Block

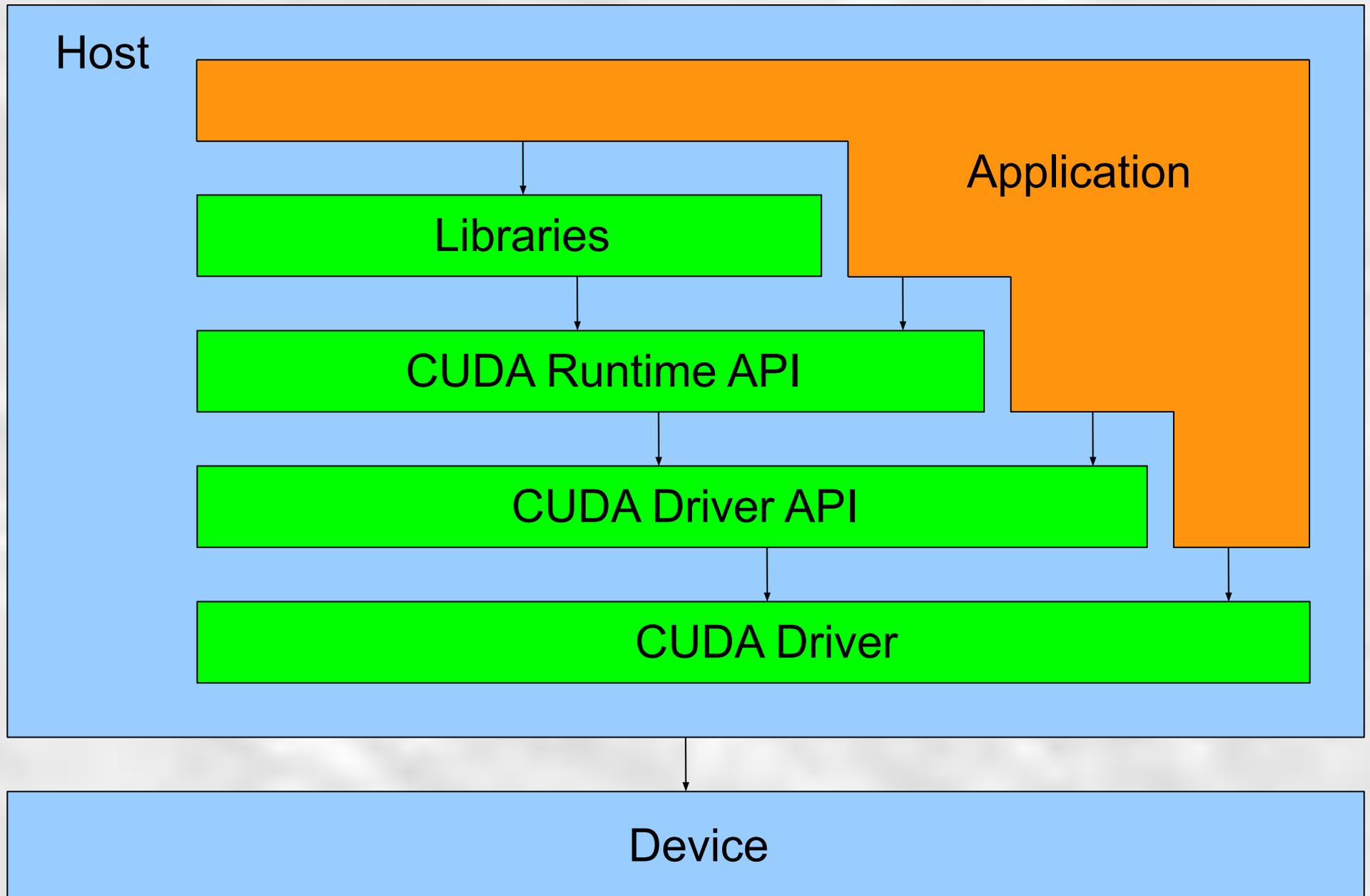
| | | |
|--------------|--------------|--------------|
| T (0:0:0) | T (1:0:0) | T (2:0:0) |
| T (0:1:0) | T (1:1:0) | T (2:1:0) |
| T (0:2:0) | T (1:2:0) | T (2:2:0) |

Приведен пример сетки из 20 блоков (5x4), в каждом блоке 18 (3x3x2) потоков. Всего в сетке 360 потоков.

Особенности программирования

- **функция ядро** возвращает только **void**
 - **память** — узкое место в вычислениях и требует особого внимания
 - **шина PCI-Express** — узкое место в вычислениях
 - **ветвления** внутри warp **снижают** быстродействие
-
-

Программный стек CUDA



CUDA Toolkit

Последняя версия CUDA Toolkit 4.0 RC2

- <https://nvdeveloper.nvidia.com>

Состав:

- Драйвер для разработчиков
- GPU Computing SDK

GPU Computing SDK:

- Компилятор
 - Набор утилит
 - Документация
 - Библиотеки (CUBLAS, CUSPARSE)
 - Примеры
-
-

Компилятор NVCC

Самый важный параметр:

--help (-help) — печатает справку

Основные выходные форматы (и ключи компиляции):

--cubin (-cubin) — компилирует в виртуальный формат cubin

--ptx (-ptx) — компиляция в ассемблер для gpi

--gpi (-gpi) — компиляция в бинарный формат

NVIDIA Parallel nSight специально разработан для работы в Visual Studio

Типы функций

| Обозначение | Где расположена | Кто может вызывать |
|-------------------------|-----------------|--------------------|
| <code>__global__</code> | GPU | CPU |
| <code>__device__</code> | GPU | GPU |
| <code>__host__</code> | CPU | CPU |

- по умолчанию все функции `__host__`
- `__host__` и `__device__` **совместимы**, компилятор создаст две версии: для **CPU** и **GPU**

```
__global__ void sum(float *c, float *a, float b);
```

```
__host__ __device__ float add(float a, float b);
```

Hello World! Сложение массивов.

```
#define N 1024
```

```
// GPU
```

```
__global__ void sum(float *c, float *a, float *b)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    c[index] = a[index] + b[index];
}
```

```
// CPU
```

```
void sum(float *c, float *a, float *b)
{
    for(int i = 0; i < N; i++)
    {
        c[i] = a[i] + b[i];
    }
}
```

Встроенные константы:

blockIdx — номер блока у текущего потока;

blockDim — количество блоков;

threadIdx — номер потока в блоке.

Hello World! CPU инициализация

```
int main(int argc, char **argv)
{
    float *a, *b, *c;
    float *A, *B, *C;
    a = (float*) malloc(N * sizeof(float));
    b = (float*) malloc(N * sizeof(float));
    c = (float*) malloc(N * sizeof(float));

    cudaMalloc((void **)&A, N * sizeof(float));
    cudaMalloc((void **)&B, N * sizeof(float));
    cudaMalloc((void **)&C, N * sizeof(float));

    for(int i = 0; i < N; i++)
    {
        a[i] = RandFloat(0.0f, 1.0f);
        b[i] = RandFloat(0.0f, 1.0f);
    }
}
```

Hello World! CPU вызов ядра

```
cudaMemcpy(A, a, N * sizeof(float), cudaMemcpyHostToDevice);  
cudaMemcpy(B, b, N * sizeof(float), cudaMemcpyHostToDevice);
```

```
sum<<<N/256, 256>>>(C, A, B);
```

```
cudaMemcpy(c, C, N * sizeof(float), cudaMemcpyDeviceToHost);
```

```
cudaFree(A);
```

```
cudaFree(B);
```

```
cudaFree(C);
```

```
free(a);
```

```
free(b);
```

```
free(c);
```

```
}
```

Hello World! CPU вызов ядра

```
cudaMemcpy(A, a, N * sizeof(float), cudaMemcpyHostToDevice);  
cudaMemcpy(B, b, N * sizeof(float), cudaMemcpyHostToDevice);
```

```
sum<<<N/256, 256>>>(C, A, B);
```

```
cudaMemcpy(c, C, N * sizeof(float), cudaMemcpyDeviceToHost);
```

```
cudaFree(A);
```

```
cudaFree(B);
```

```
cudaFree(C);
```

```
free(a);
```

```
free(b);
```

```
free(c);
```

```
}
```

Hello World! CPU вызов ядра

```
cudaMemcpy(A, a, N * sizeof(float), cudaMemcpyHostToDevice);  
cudaMemcpy(B, b, N * sizeof(float), cudaMemcpyHostToDevice);
```

```
sum<<<N/256, 256>>>(C, A, B);
```

```
cudaMemcpy(c, C, N * sizeof(float), cudaMemcpyDeviceToHost);
```

```
cudaFree(A);
```

```
cudaFree(B);
```

```
cudaFree(C);
```

```
free(a);
```

```
free(b);
```

```
free(c);
```

```
}
```

Hello World! CPU вызов ядра

```
cudaMemcpy(A, a, N * sizeof(float), cudaMemcpyHostToDevice);  
cudaMemcpy(B, b, N * sizeof(float), cudaMemcpyHostToDevice);
```

```
sum<<<N/256, 256>>>(C, A, B);
```

```
cudaMemcpy(c, C, N * sizeof(float), cudaMemcpyDeviceToHost);
```

```
cudaFree(A);
```

```
cudaFree(B);
```

```
cudaFree(C);
```

```
free(a);
```

```
free(b);
```

```
free(c);
```

```
}
```

GPGPU прочее

DirectCompute — библиотека от Microsoft. Часть DirectX;

OpenCL — кроссплатформенная библиотека;

Готовые библиотеки с поддержкой GPGPU:

- **OpenCV** — обработка изображения и компьютерное зрение
- **CUBLAS** — математические вычисления
- **CUFFT** — быстрые преобразования фурье
- **CUSPARSE** — библиотека линейной алгебры

Пакеты ПО со встроенной поддержкой **GPU**, например **Matlab**

OpenCV

```
#include <iostream>
#include "opencv2/opencv.hpp"
#include "opencv2/gpu/gpu.hpp"

int main (int argc, char* argv[])
{
    cv::gpu::GpuMat dst, src = cv::imread("file.png",
                                         CV_LOAD_IMAGE_GRAYSCALE);

    cv::gpu::threshold(src, dst, 128.0, 255.0, CV_THRESH_BINARY);

    cv::imshow("Result", dst);
    cv::waitKey();

    return 0;
}
```

Ссылки:

- http://www.nvidia.ru/object/cuda_home_new_ru.html - о CUDA
- http://www.nvidia.ru/object/cuda_opengl_new_ru.html - OpenGL
- http://www.nvidia.ru/object/directcompute_ru.html - DirectCompute
- <http://gpgpu.org/> - подборка информации по GPGPU
- <http://www.gpgpu.ru> - GPGPU по-русски

Контакты:

- skype: [sviridenkov.anatoliy](https://www.skype.com/ru/people/sviridenkov.anatoliy)
 - e-mail: Anatoliy.Sviridenkov@gmail.com
 - блог http://bit.ly/cuda_blog
-
-