

# Разработка параллельных программ для GPU

Обзор CUDA API

Виды CUDA APIs и возможности CUDA-устройств

# **ОСОБЕННОСТИ CUDA APIs**

# Виды CUDA APIs

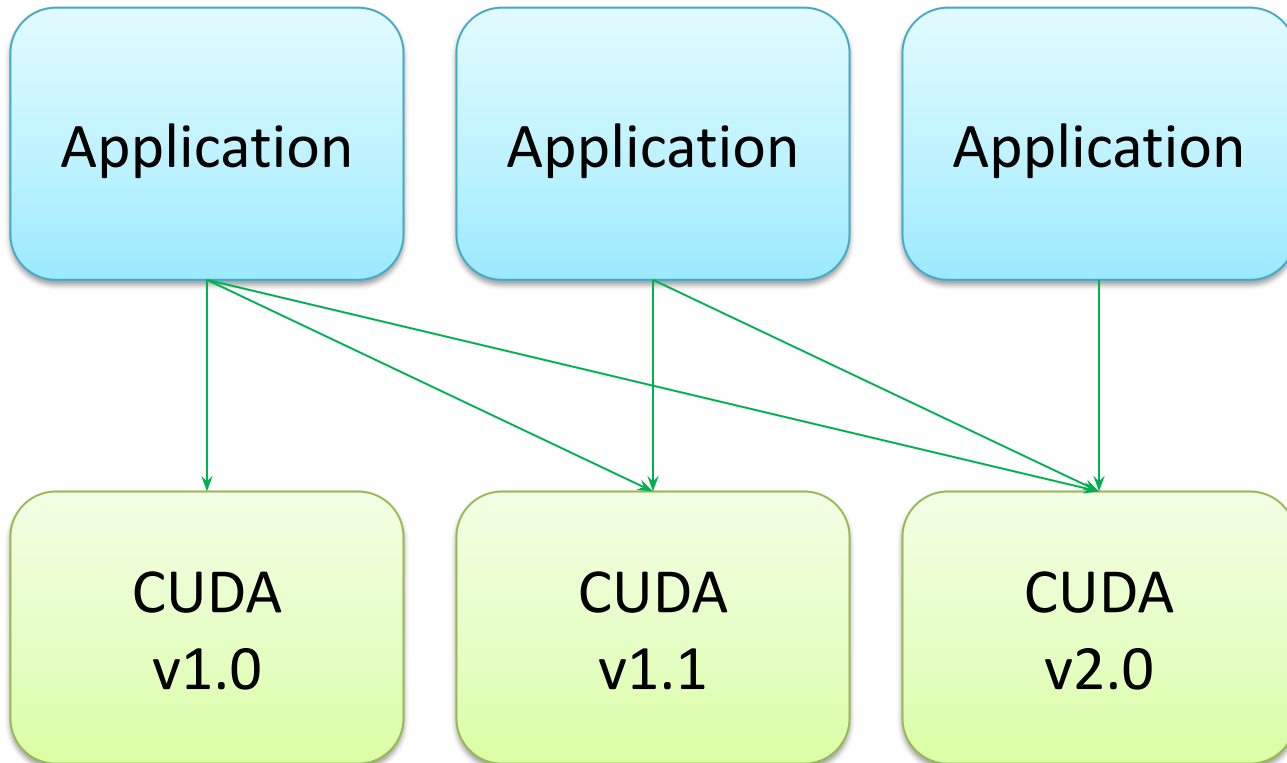
- CUDA Driver API
  - Ручная инициализация контекста GPU
  - Отсутствуют CUDA-расширения для C++
  - Код CPU может компилироваться без nvcc
- CUDA Runtime API
  - Автоматическая инициализация контекста GPU
  - Наличие CUDA-расширений для C++

# Выбор CUDA API

- CUDA Driver API
  - Больше гибкости ( + )
  - Сложность кода ( - )
- **CUDA Runtime API**
  - Меньше гибкости ( - )
  - Простота кода ( + )

# Совместимость CUDA API

- Имеется обратная совместимость версий



# Вычислительные возможности GPU

- **Capability** – это версия архитектуры CUDA GPU, которая указывает на его вычислительные возможности и особенности

GPU	ВЫЧИСЛИТЕЛЬНЫЕ ВОЗМОЖНОСТИ
Tesla C2050/C2070	2.0
Tesla C1060	1.3
Tesla C870	1.0
Tesla D870	1.0

`cudaGetDeviceProperties()`

Способы оценки эффективности приложений CUDA

# **ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ**

# Время выполнения

- Общее время вычислений на GPU
- Время выполнения участка кода GPU



# Таймеры CPU

- Таймеры CPU позволяют замерять общее время выполнения вычислений на GPU

```
LARGE_INTEGER frequency;  
LARGE_INTEGER start, finish;
```

```
QueryPerformanceFrequency(&frequency);  
QueryPerformanceCounter(&start);
```

```
// Обращение к GPU...
```

```
cudaThreadSynchronize()
```

```
QueryPerformanceCounter(&finish);
```

```
double seconds = ((finish.QuadPart - start.QuadPart) / (double)(frequency.QuadPart));
```

# Таймеры CUDA

```
float time;
cudaEvent_t start, stop;

cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start, 0);

// Обращение к GPU...

cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);

cudaEventElapsedTime(&time, start, stop);

cudaEventDestroy(start);
cudaEventDestroy(stop);
```

- Таймеры CUDA позволяют замерять время выполнения участка кода GPU

# Скорость передачи данных

- Теоретическая пропускная способность

$$F_{\text{DDRAM}} * (R_{\text{DDRAM}}/8) * \text{sizeof(float)},$$

где  $F_{\text{DDRAM}}$  – частота,  $R_{\text{DDRAM}}$  – разрядность шины

- Эффективная пропускная способность

$$(B_R + B_W) / \text{time},$$

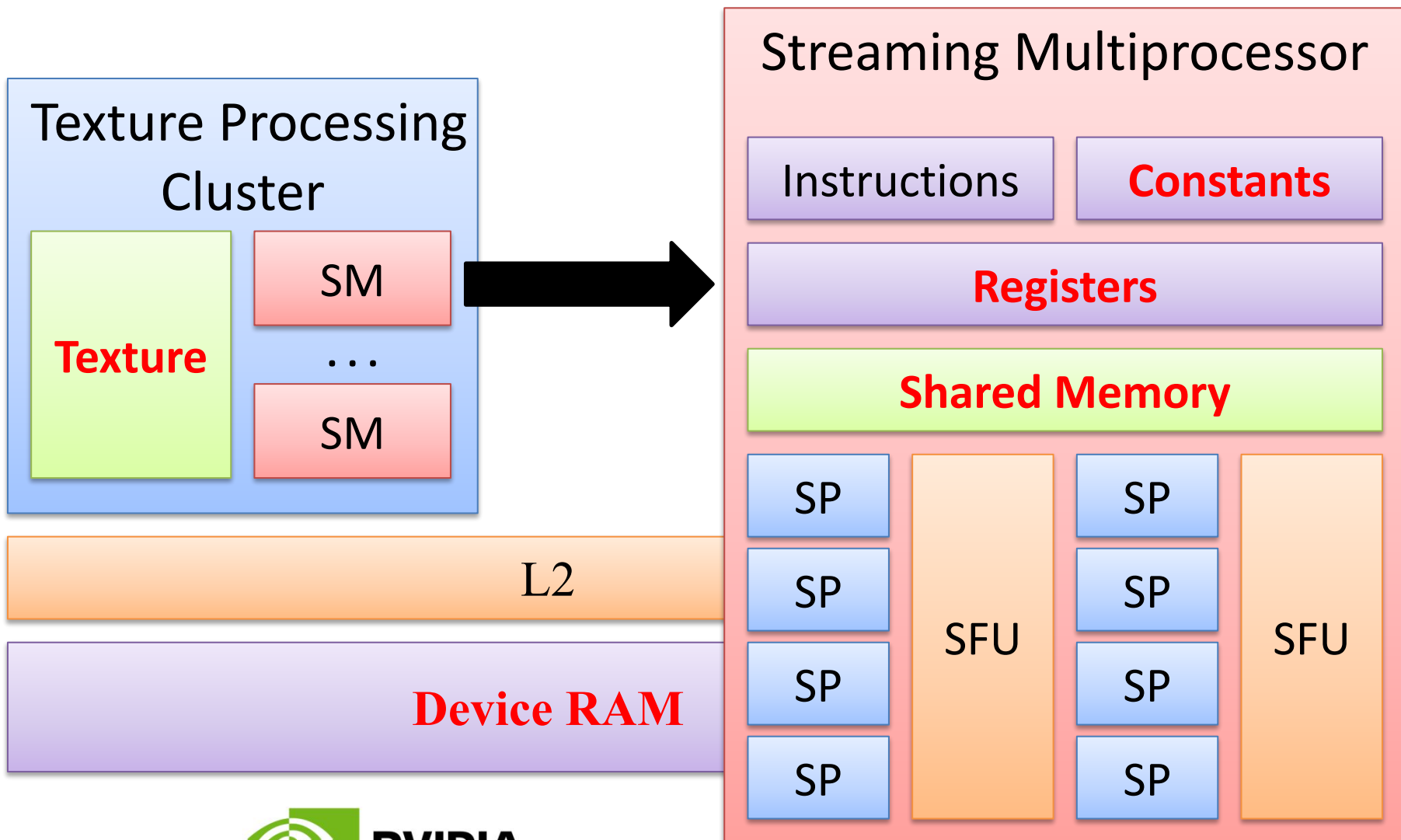
где  $B_R$  и  $B_W$  – объем прочитанной/записанной информации

- Реальная пропускная способность

Способы оптимизации работы с памятью CUDA GPU

# **ОПТИМИЗАЦИЯ РАБОТЫ С ПАМЯТЬЮ**

# Архитектура CUDA GPU



# Типы памяти устройства

- Streaming Multiprocessor
  - Регистровая память
  - Разделяемая память
  - Память констант
- Texture Processing Cluster
  - Память текстур
- DDRAM
  - Локальная память
  - Глобальная память

# Передача данных Host/Device

- Является дорогостоящей операцией
- Возможна асинхронная передача

`cudaMemcpy()`

`cudaMemcpyAsync()`

# Асинхронная передача данных

- Копирование данных и выполнение ядра можно осуществлять параллельно

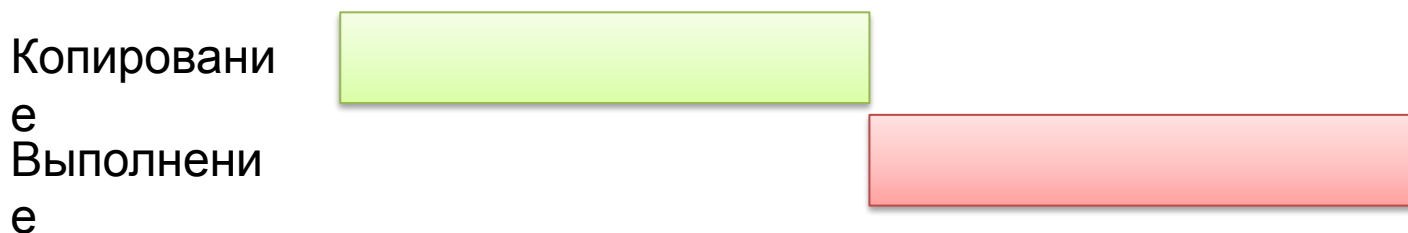
```
for (int i = 0; i < countStreams; ++i)
{
    ...
    cudaMemcpyAsync(..., stream[i]);
}
```

```
for (int i = 0; i < countStreams; ++i)
{
    ...
    kernel<<<N / (countThreads * countStreams), countThreads, 0, stream[i]>>>(...);
}
```

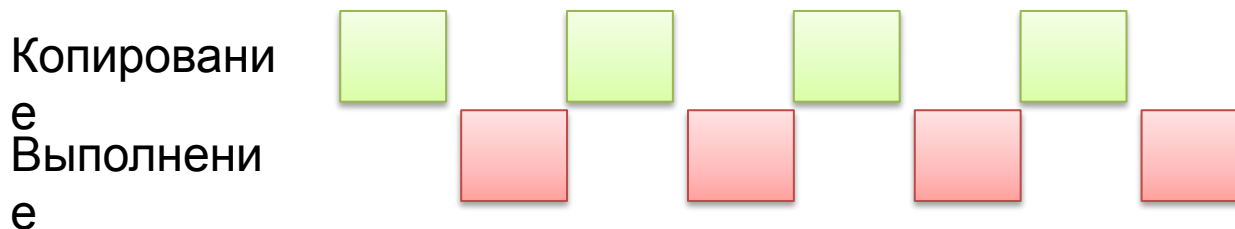


# Возможная оптимизация

- Синхронная передача данных в GPU



- Асинхронная передача данных в GPU



# Нулевое копирование (Zero Copy)

- Прямое обращение к памяти Host'a
  - **Встроенные видеокарты**
    - Использование кэша CPU

# Объединенное чтение DDRAM

- Выравнивание исходных данных по границе слова
- Потоки warp'а должны осуществлять одновременное чтение DDRAM

# Разделяемая память и конфликты

- Общая для всех потоков блока
- Распределяется между блоками
- Разбивается на банки (32-битные слова)

# Регистровое давление

- Регистры жестко распределяются между потоками мультипроцессора
- При большом количестве потоков возникает конфликт доступа к регистрам

Методы оценки топологии вычислений CUDA

# **ВЫБОР ОПТИМАЛЬНОЙ ТОПОЛОГИИ**

# Степень покрытия

- Степень покрытия мультипроцессора – это отношение числа активных warp'ов к максимально возможному числу активных warp'ов
  - По количеству используемых регистров
    - С учетом топологии вычислений
    - Без учета топологии вычислений
  - По размеру используемой разделяемой памяти

# Определение степени покрытия

- CUDA GPU:
  - 8192 регистра
  - 768 потоков на мультипроцессор
- Топология:
  - 12 регистров на ядро
  - 128 потоков в блоке

$T_{\max} = 8192 \text{ регистров} / 12 \text{ регистров} = 682 \text{ потока}$

$T_{\text{real}} = \text{int}(682 / 128) * 128 = 640 \text{ потоков}$

$C = T_{\text{real}} / T_{\max} = 83\%$



Оптимизация инструкций CUDA

# ОПТИМИЗАЦИЯ КОДА

# Регистровая зависимость

- Инструкция использует регистр, значение которого было получено при выполнении предыдущей инструкции

```
register = instruction1();  
instruction2(register);
```

# Float vs Double

- Арифметические операции с float-числами осуществляются быстрее, чем с double-числами
  - Рекомендуется использовать суффикс «f» при объявлении числовых констант, например, 3.14f

# Деление чисел

- При делении чисел на **степень двойки** рекомендуется использовать оператор сдвига

$$\begin{aligned} X / N &\rightarrow X \gg \log_2(N) \\ X \% N &\rightarrow X \& (N-1) \end{aligned}$$

# Степень числа

- Для известных целых значений степеней рекомендуется использовать явное умножение вместо вызова `pow()`

`pow(X, 2)` → `X * X`  
`pow(X, 3)` → `X * X * X`

# Часто используемые функции

- Обратный квадратный корень
  - `rsqrtf()` / `rsqrt()`
- Прочие арифметические операции
  - `expf2()` / `exp2()` – экспонента во **2-й степени**
  - `expf10()` / `exp10()` – экспонента в **10-й степени**
  - `cbrtf()` / `cart()` – экспонента в степени **1/3**
  - `rcbrtf()` / `rebut()` – экспонента в степени **-1/3**

# Точность vs Скорость

- Аппаратные аналоги функций

\_\_sinf() / sinf()

\_\_cosf() / cosf()

\_\_expf() / expf()

- Совмещенные функции

sincosf() / sincos()

Общие рекомендации по написанию кода

# **УПРАВЛЕНИЕ ПОТОКОМ КОМАНД**



# Операторы ветвления

- Инструкции управления потоком команд (if, switch, for, while, do-while) отрицательно сказываются на производительности
  - В идеале все потоки warp'a должны идти по одному пути, иначе увеличивается количество выполняемых инструкций и возможно последовательное выполнение

# Предикативная запись

```
if (a >= b)
{
    c = a;
}
else
{
    c = b;
}
```



```
c = (a >= b) & a + (a < b) & b;
```

Отладка и профилирование приложений CUDA

# **ОТЛАДКА И ПРОФИЛИРОВАНИЕ**

# Существующие утилиты

- Linux
  - **CUDA-GDB**
  - <http://developer.nvidia.com/cuda-gdb>
- Windows Vista & Windows 7
  - **NVIDIA Parallel Nsight**
  - <http://developer.nvidia.com/nvidia-parallel-nsight>

Краткий обзор архитектурных особенностей GPU

# **АППАРАТНЫЕ ОСОБЕННОСТИ GPU**

# Причины рассогласования

- Основные причины рассогласования результатов вычислений на GPU и CPU
  - Усечение double чисел до float при отсутствии аппаратной поддержки double
  - Неассоциативность арифметических операций с дробными числами
  - Небольшие отклонения от стандарта IEEE 754
  - Особенности архитектуры процессоров x86

# Литература

- NVIDIA Developer Zone
  - <http://developer.nvidia.com/cuda>
- NVIDIA Parallel Nsight
  - <http://developer.nvidia.com/cuda-gdb>
- CUDA C Best Practices Guide
  - [http://developer.download.nvidia.com/compute/cuda/4\\_0/toolkit/docs/CUDA\\_C\\_Best\\_Practices\\_Guide.pdf](http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf)

**ВОПРОСЫ?**