




Казанский  
федеральный  
университет

ВЫСШАЯ ШКОЛА  
информационных технологий  
и информационных систем


# Алгоритмы в CUDA

Эдуард Храмченков


# Параллельные алгоритмы

- ▶ Разработка алгоритмов для CUDA в целом соответствует базовым принципам создания параллельных алгоритмов
  - ▶ Важно учитывать особую структуру памяти GPU для достижения максимальной производительности
  - ▶ Как правило различные версии параллельного алгоритма для GPU могут отличаться по быстродействию на порядки
- 


# Параллельная редукция

- ▶ Дан массив  $a[n]$ , и операция  $op$
  - ▶ Редукция  $A = a[0] op a[1] op \dots op a[n-1]$
  - ▶ Рассмотрим редукцию массива данных по сумме
  - ▶ В данном случае редукция – сумма всех элементов массива
  - ▶ Последовательная реализация редукции тривиальна
  - ▶ Как распараллелить редукцию?
- 

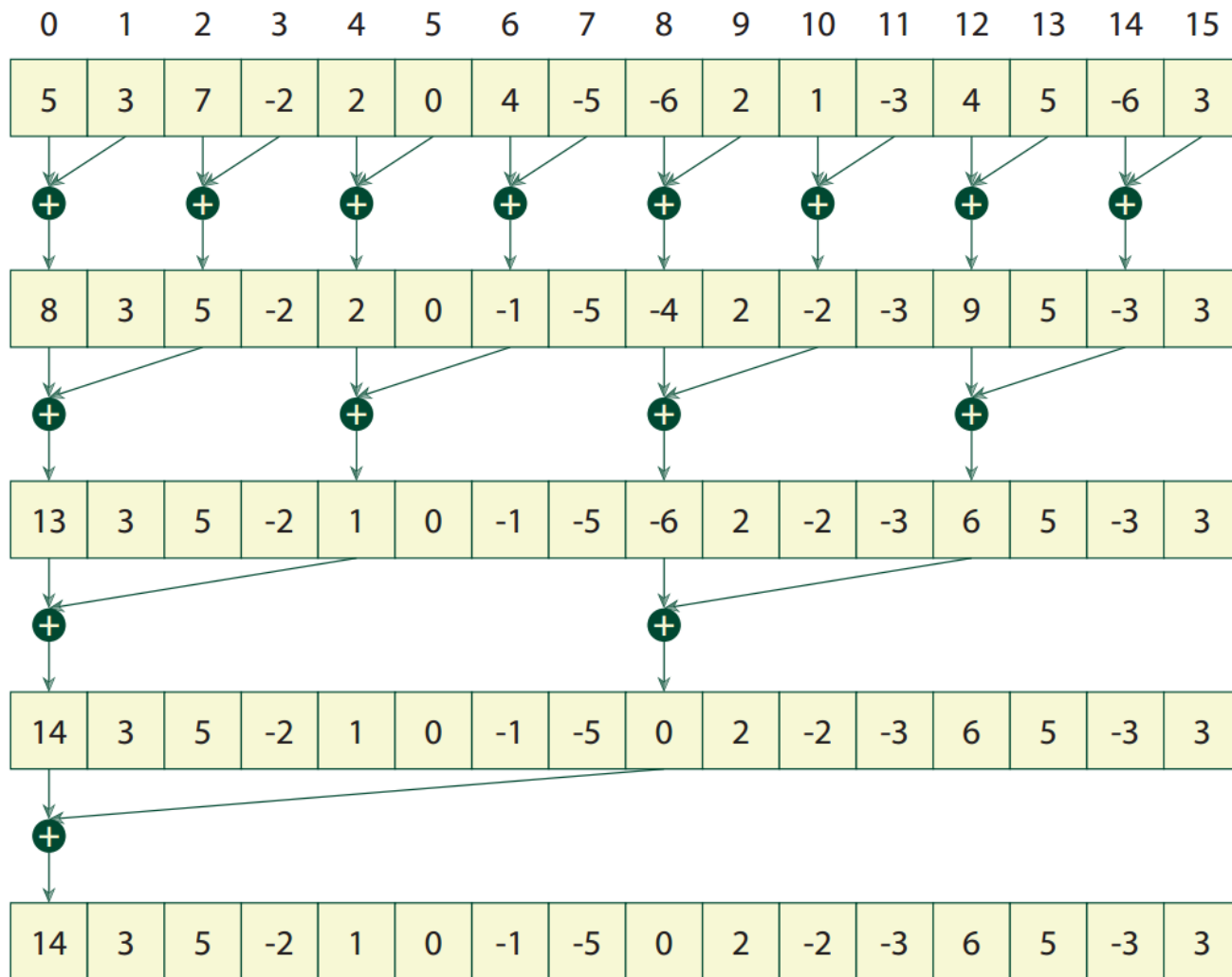
# Параллельная редукция

- ▶ Метод «разделяй и властвуй» – один из основных методов построения алгоритмов
  - ▶ Исходный массив делится на части и каждую часть обрабатывает свой блок
  - ▶ Независимое нахождение частичных сумм
  - ▶ Для увеличения числа нитей в блоке можно использовать 3-мерные блоки нитей
  - ▶ Это необходимо, если массив очень большой – увеличение диапазона индексации
- 


# Параллельная редукция

- ▶ Внутри каждого блока также введем параллелизацию по отдельным нитям
  - ▶ Сначала суммируются попарно соседние элементы, потом их частичные суммы, и т.д.
  - ▶ На каждом этапе суммирования число шагов уменьшается 2
  - ▶ Количество шагов, количество операций - ?
  - ▶  $O(\log_2 m)$ ;  $O(m)$
- 


# Параллельная редукция



# Пример 0

- ▶ Элементарная редукция
  - ▶ Вопрос – почему такая разница между фактическим и замеренным временем выполнения?
  - ▶ Накладные расходы на неявное копирование памяти
- 

# Пример 1

- ▶ В случае если *ор* – простая операция, производительность ограничена быстродействием памяти
  - ▶ Оптимизация – создать буфер в разделяемой памяти для части массива обрабатываемой блоком
  - ▶ Переписать выделение памяти на GPU для гарантированного отсутствия неявного копирования
- 

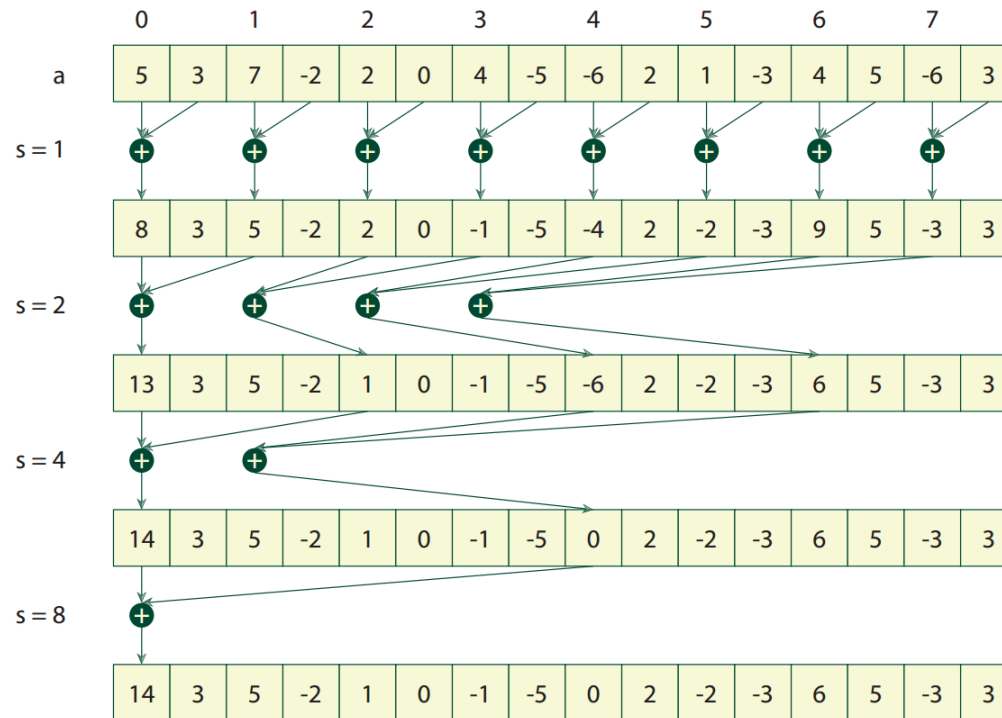


# Пример 2

- ▶ Недостаток варианта №1 – ветвление нитей
- ▶ Выход – перераспределение операций

# Пример 2

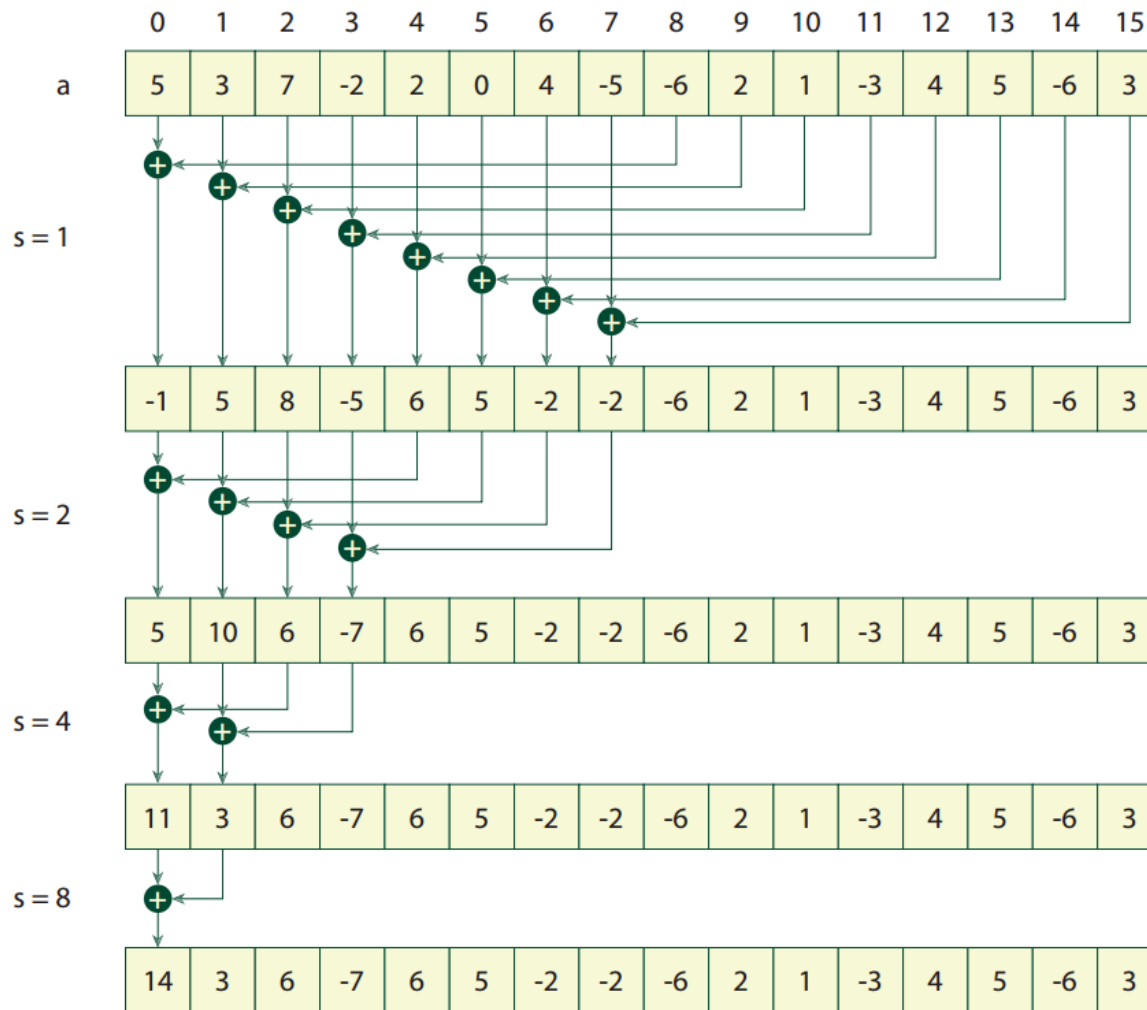
- ▶ Недостаток – ветвление нитей
- ▶ Выход – перераспределение операций



# Пример 3

- ▶ Недостаток варианта №2 – при  $s = 2$  все обращения в разделяемую память идут по четным банкам памяти
- ▶ Решение конфликта по банкам памяти - суммирование наиболее удаленных элементов

# Пример 3



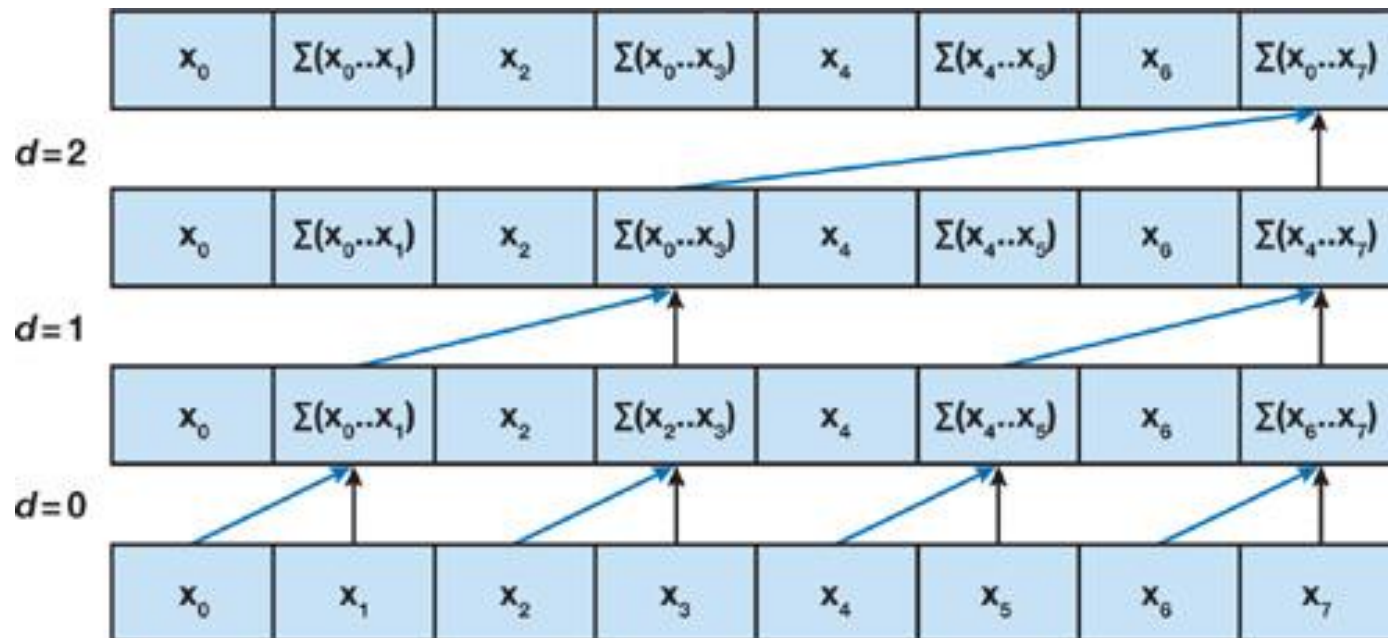
# Пример 4

- ▶ На первом шаге работает половина нитей блока
- ▶ Уменьшив число блоков вдвое, а в каждом блоке будем обрабатывать в 2 раза больше элементов

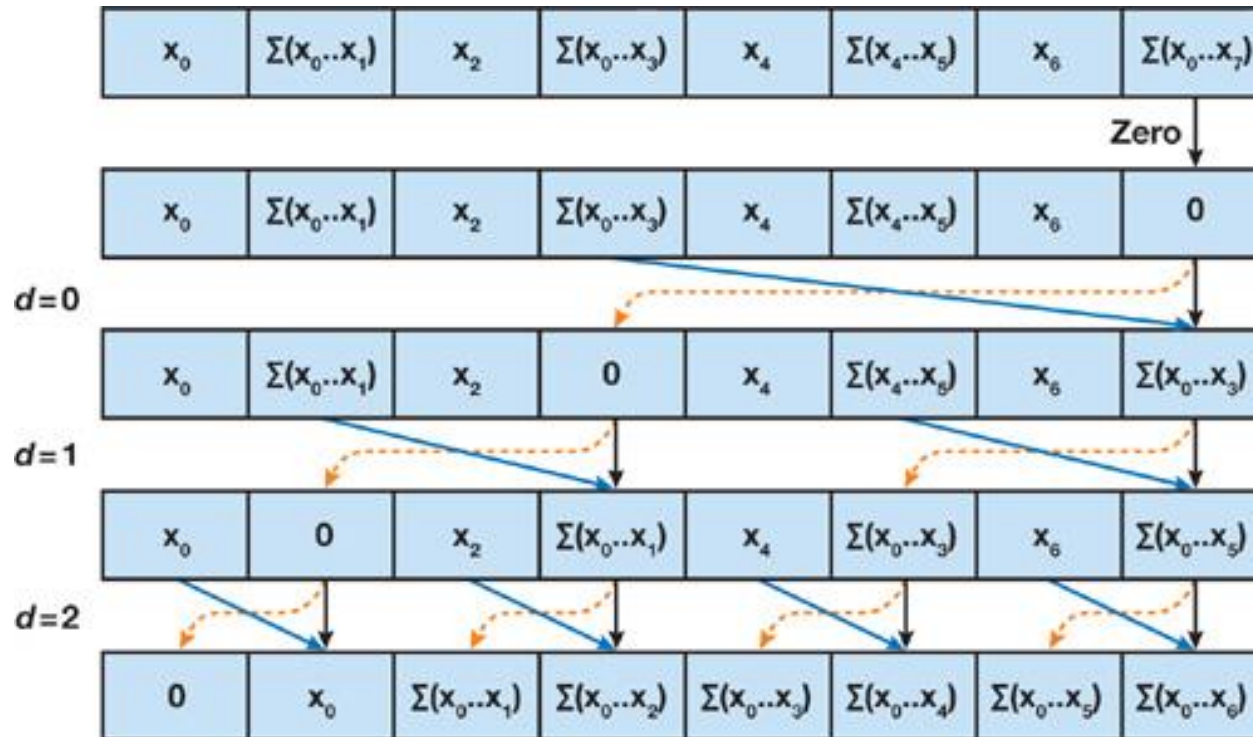
# Префиксная сумма

- Для массива  $\{a_0, a_1, \dots, a_{n-1}\}$  префиксная сумма есть массив  $\{0, a_0, a_0 + a_1, a_0 + a_1 + a_2, \dots, a_0 + \dots + a_{n-2}\}$

# Префиксная сумма: 1й этап



# Префиксная сумма: 2й этап





# Пример 6

- ▶ Префиксная сумма в пределах одного блока
- ▶ Работает только на небольших массивах

# Пример 7

- ▶ Префиксная сумма на любых массивах кратных 2



Казанский  
федеральный  
университет

ВЫСШАЯ ШКОЛА  
информационных технологий  
и информационных систем

# Вопросы

[ekhramch@kpfu.ru](mailto:ekhramch@kpfu.ru)