Шаблоны для параллельных алгоритмов

Зачем нужны шаблоны?

- Количество потоков по трем измерениям.
- Количество потоков в группе по трем измерениям.
- Локальная память.
- Макс. размер изображений.
- Несколько видеокарт.
- Асинхронный запуск ядер и команд.
- ...

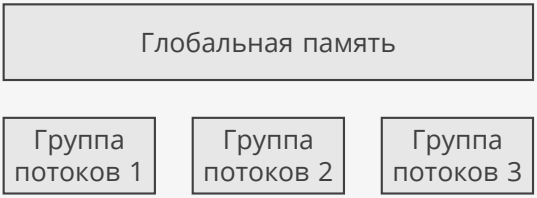
Раздел 1

Отображение

Отображение (map)

Последовательная версия:

```
for (int i=0; i<n; ++i) {
  result[i] = func(a[i],b[i]);
}
```



Параллельная версия:

- Разделить данные на независимые части.
- Отдать каждую часть отдельной группе потоков.
- Скопировать данные из глобальной памяти в локальную.
- Обработать и скопировать обратно.

Раздел 2

Редукция, свертка

Редукция (reduce), свертка (fold)

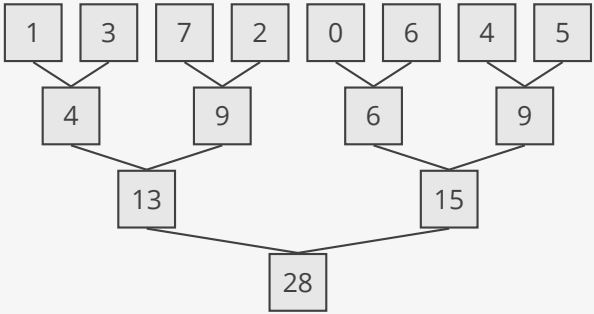
Последовательная версия (редукция):

```
float result = x[0];
for (int i=1; i<n; ++i) {
  result = func(result,x[i]);
}
```

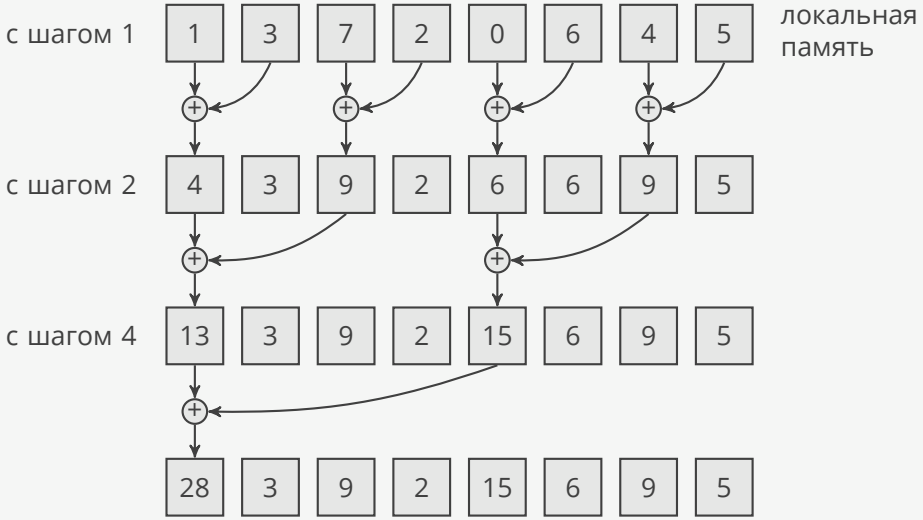
Последовательная версия (свертка):

```
float result = initial;
for (int i=0; i<n; ++i) {
  result = func(result,x[i]);
}
```

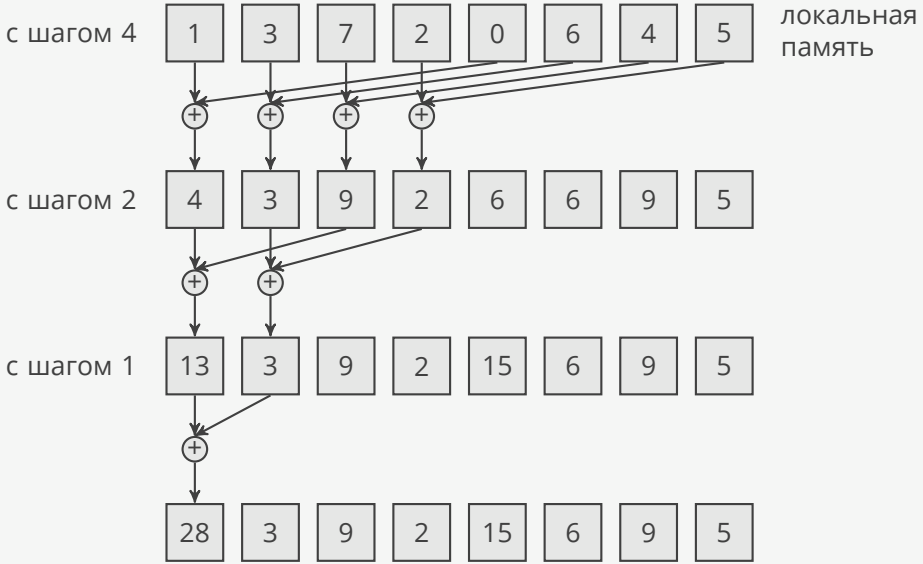
Параллельная версия (редукция, суммирование):



Параллельная редукция (версия 1)



Параллельная редукция (версия 2)



Редукция в OpenCL

Суммирование в локальной памяти:

```
int local_id = get_local_id(0); // номер потока в группе
int local_size = get_local_size(0); // кол-во потоков в группе
for (int offset=local_size/2; offset > 0; offset /= 2) {
  if (local_id < offset) {
    data[local_id] += data[local_id + offset];
  }
  barrier(CLK_LOCAL_MEM_FENCE);
}
```

Запись в глобальную память:

```
if (local_id == 0) {
  int group_id = get_group_id(0);
  sums[group_id] = data[0];
}
```

Редукция в OpenCL

Вариант 1 (канонический):
рекурсивно вызвать редукцию для массива `sums`.

Вариант 2 (и так сойдет):
завершить редукцию на процессоре.

```
std::vector<float> sums(ngroups);
float total_sum = 0;
for (float s : sums) { total_sum += s; }
```

Асимптотическая сложность:

$$\mathcal{O}\left(N/P + \log_2 N\right)$$

N количество элементов
P количество потоков

Редукция на видеокарте

- Два этапа: внутри блока и глобальная.
- Второй этап — редукция результирующего массива из первого этапа.
- Возможны оптимизации по размеру блока, количеству блоков и т.п.

Раздел 3

Сканирование

Сканирование (scan)

Включительное сканирование

$$y_i = \sum_{j=1}^{i} x_j, \qquad i = 1, ..., n$$

```
float sum = 0;
for (int i=0; i<n; ++i) {
  float x_i = x[i];
  y[i] = sum;
  sum += x_i;
}
```

Исключительное сканирование

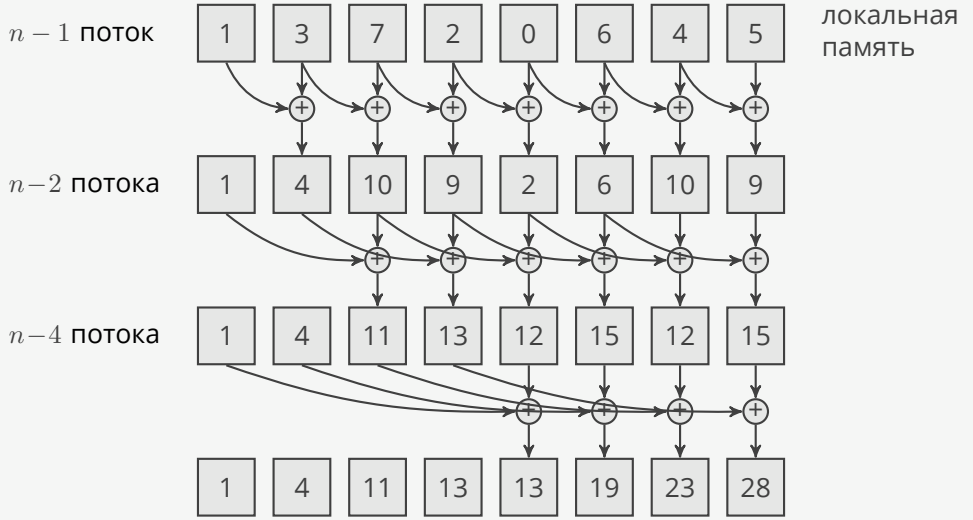
$$z_i = \sum_{j=1}^{i-1} x_j, \qquad i = 2, ..., n \qquad z_1 = 0$$

```
float sum = 0;
for (int i=0; i<n; ++i) {
  sum += x[i];
  z[i] = sum;
}
```

Другие названия: префиксная сумма, кумулятивная сумма.

По сути — суммирование с записью промежуточных результатов.

Параллельное сканирование



Сканирование в OpenCL

```
int local_id = get_local_id(0); // номер потока в группе
int local_size = get_local_size(0); // кол-во потоков в группе
float sum = 0;
for (int offset=1; offset<local_size; offset *= 2) {
  if (local_id >= offset) {
    sum += data[local_id - offset];
  }
  barrier(CLK_LOCAL_MEM_FENCE);
  data[local_id] = sum;
  barrier(CLK_LOCAL_MEM_FENCE);
}
int global_id = get_global_id(0);
result[global_id] = data[local_id];
```

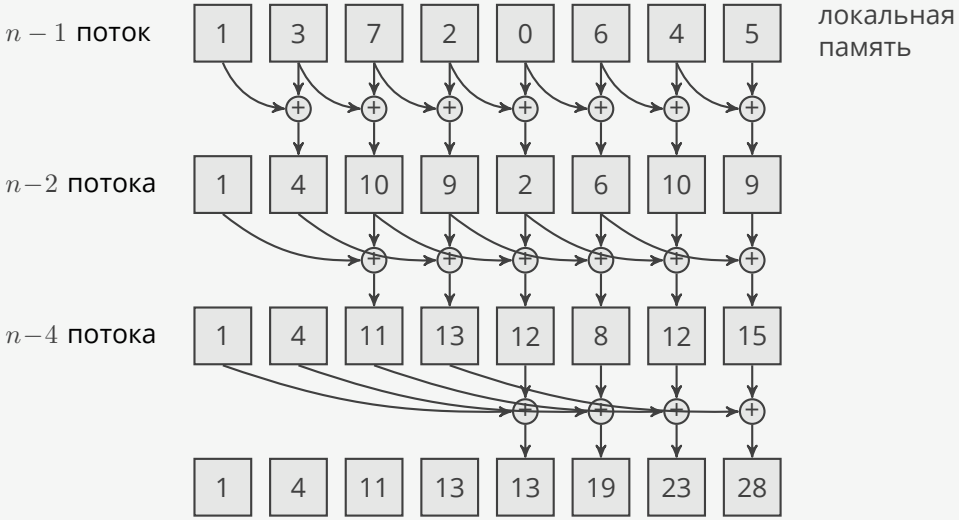
Сканирование на видеокарте

- Опять два этапа.
- Второй этап опять выполняется рекурсивно.
- Те же оптимизации, что и в редукции.

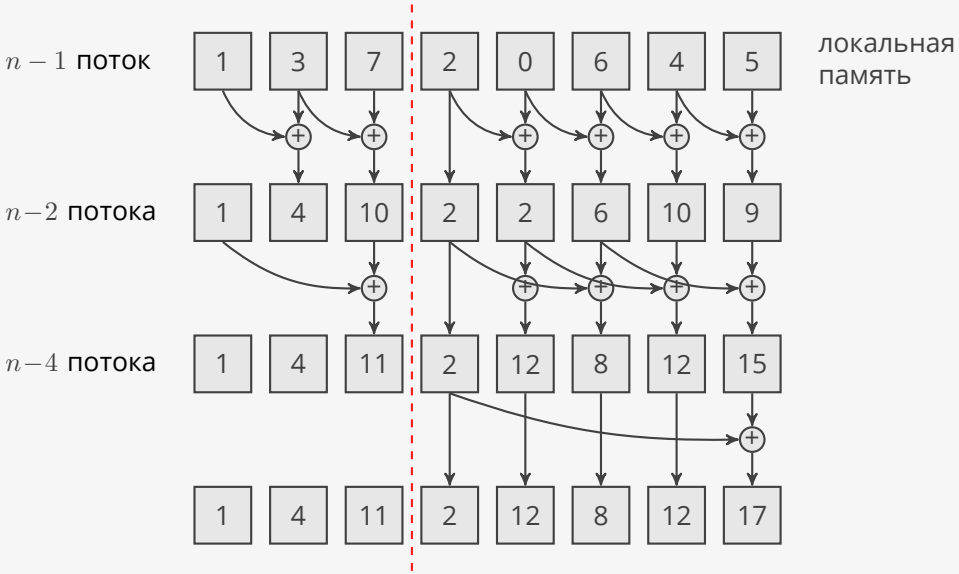
Раздел 4

Сканирование по сегментам

Сканирование по сегментам (segscan)



Сканирование по сегментам (segscan)



Сканирование по сегментам в OpenCL

```
int i = get_local_id(0); // номер потока в группе
int local_size = get_local_size(0); // кол-во потоков в группе
float sum = 0;
for (int offset=1; offset<local_size; offset *= 2) {
  if (i >= offset) {
    data[i] = flags[i] ? data[i] : (data[i] + data[i-offset]);
    flags[i] = flags[i] | flags[i-offset];
  }
  barrier(CLK_LOCAL_MEM_FENCE);
}
int global_id = get_global_id(0);
result[global_id] = data[i];
```

Все примитивы

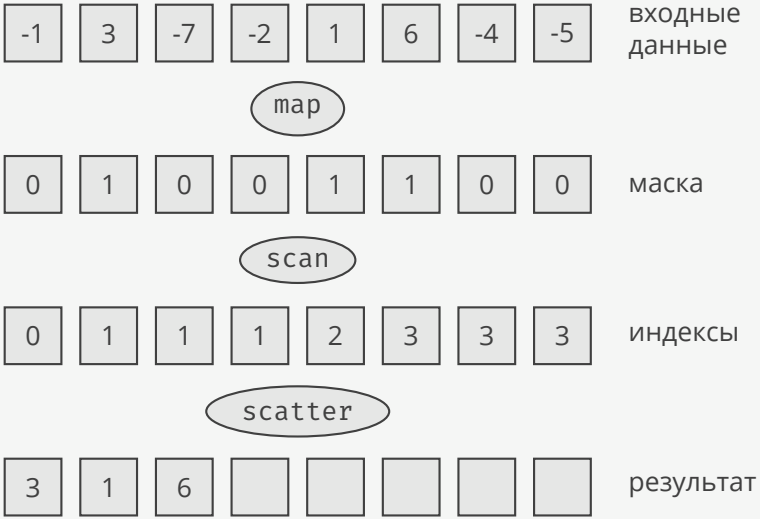
Операция	Название	Тип
отображение	`map`	$N \to N$
сканирование	`scan`	$N \to N$
сканирование по сегментам	`segscan`	$N \to M$
редукция	`reduce`	$N \to 1$

- Всего три шаблона: `map`, `reduce`, `scan`, `segscan` из функциональных языков.
- Сканирование не распространено в последовательных алгоритмах.

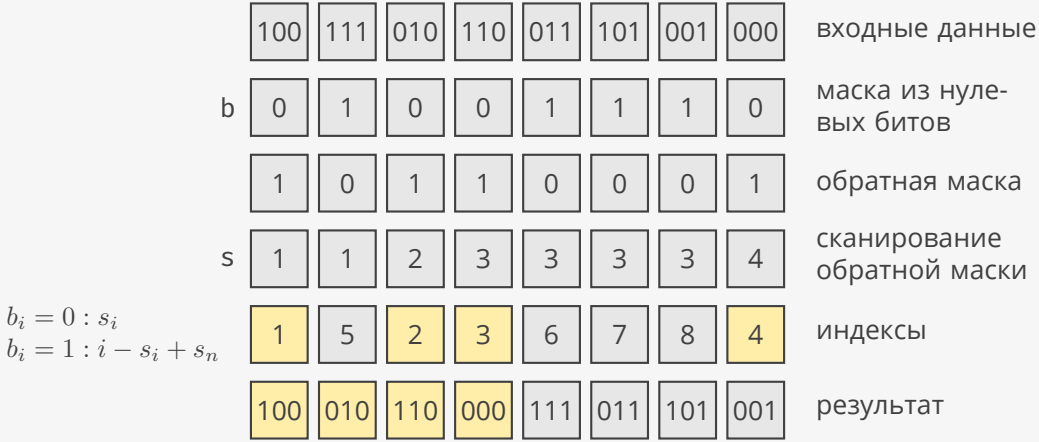
Раздел 5

Примеры применения шаблонов

Упаковка потока данных (`filter`)



Сортировка по разрядам



Изображения:

- «Aerial photo of mountain at daytime», Mamun Srizon.