Парвицкий Никон task02 Theory

Теоретическое задание: параллелизуемость/code divergence/memory coalesced access

Задача 1

Пусть на вход дан сигнал x[n], а на выход нужно дать два сигнала y1[n] и y2[n]:

```
y1[n] = x[n - 1] + x[n] + x[n + 1]
y2[n] = y2[n - 2] + y2[n - 1] + x[n]
```

Какой из двух сигналов будет проще и быстрее реализовать в модели массового параллелизма на GPU и почему?

Ответ:

Сигнал y1 будет проще и быстрее реализовать в модели массвого параллелизма так как мы можем посчитать одновременно все значения y1 запустив n потоков параллельно и каждый поток будет вычислять свое значения для y1, в отличии от y2, где следующее значение зависит от предыдущего и мы вынуждены ждать

Задача 2

Предположим что размер warp/wavefront равен 32 и рабочая группа делится на warp/wavefront-ы таким образом что внутри warp/wavefront номер WorkItem по оси x меняется чаще всего, затем по оси y и затем по оси z.

Напоминание: инструкция исполняется (пусть и отмаскированно) в каждом потоке warp/wavefront если хотя бы один поток выполняет эту инструкцию неотмаскированно. Если не все потоки выполняют эту инструкцию неотмаскированно - происходит т.н. code divergence.

Пусть размер рабочей группы (32, 32, 1)

```
int idx = get_local_id(1) + get_local_size(1) * get_local_id(0);
if (idx % 32 < 16)
    foo();
else
    bar();
```

Произойдет ли code divergence? Почему?

Ответ:

Для начала напишем как выглядит положение WorkItem-ов по Warp-ам исходя из условия

```
Warp 0:  (x = 0, y = 0),  (x = 1, y = 0)  ... (x = 31, y = 0)
Warp 1:  (x = 0, y = 1),  (x = 1, y = 1)  ... (x = 31, y = 1)
...
Warp 31: (x = 0, y = 31), (x = 1, y = 31) ... (x = 31, y = 31)
```

Теперь напишем как выглядит idx для каждого Warp-а

```
int idx = get_local_id(1) + get_local_size(1) * get_local_id(0);

get_local_id(1)   - координата y
get_local_size(1) - размер по y
get_local_id(0)   - координата x
тогда
Warp 0 где (x=0...31, y=0):
idx = 0 + 32 * 0 = 0
idx = 0 + 32 * 1 = 32
...
idx = 0 + 32 * 31 = 992

Warp 1 где (x=0...31, y=1):
idx = 1 + 32 * 0 = 1
idx = 1 + 32 * 1 = 33
...
idx = 1 + 32 * 31 = 993

...

Warp 31 где (x=0...31, y=31):
idx = 31 + 32 * 0 = 31
idx = 31 + 32 * 1 = 63
...
idx = 31 + 32 * 31 = 1023
```

Посмотрим как выглядят idx-ы для Warp-ов по модулю 32

```
Warp 0:
idx % 32 = 0 % 32 = 0
idx % 32 = 32 % 32 = 0
...
idx % 32 = 992 % 32 = 0

Warp 1:
idx % 32 = 1 % 32 = 1
idx % 32 = 33 % 32 = 1
...
idx % 32 = 993 % 32 = 1

...

Warp 31:
idx % 32 = 31 % 32 = 31
idx % 32 = 63 % 32 = 31
...
idx % 32 = 1023 % 32 = 31
```

Тогда для Warp-ов 0...15 будет для всех WorkItem-ов вызываться foo() для Warp-ов 16...31 будет для всех WorkItem-ов вызываться bar()

Итого: code divergence НЕ произойдет так как в рамках одного Warp-а все WorkItem-ы будут исполнять один и тот же код и никто никого не будет ждать

Задача 3

Как и в прошлом задании предположим что размер warp/wavefront равен 32 и рабочая группа делится на warp/wavefront-ы таким образом что внутри warp/wavefront номер WorkItem по оси x меняется чаще всего, затем по оси y и затем по оси z.

Пусть размер рабочей группы (32, 32, 1). Пусть data - указатель на массив float-данных в глобальной видеопамяти идеально выравненный (выравнен по 128 байтам, т.е. data % 128 == 0). И пусть размер кеш линии - 128 байт.

(a)

```
data[get_local_id(0) + get_local_size(0) * get_local_id(1)] = 1.0f;
```

Будет ли данное обращение к памяти coalesced? Сколько кеш линий записей произойдет в одной рабочей группе?

(b)

```
data[get_local_id(1) + get_local_size(1) * get_local_id(0)] = 1.0f;
```

Будет ли данное обращение к памяти coalesced? Сколько кеш линий записей произойдет в одной рабочей группе?

(c)

```
data[1 + get_local_id(0) + get_local_size(0) * get_local_id(1)] = 1.0f;
```

Будет ли данное обращение к памяти coalesced? Сколько кеш линий записей произойдет в одной рабочей группе?

Ответ

(a) как и в Задаче 2 распишем по Warp-ам как выглядят индексы

```
idx = get_local_id(0) + get_local_size(0) * get_local_id(1)

Warp 0:
idx = 0 + 32 * 0 = 0
idx = 1 + 32 * 0 = 1
...
idx = 31 + 32 * 0 = 31

Warp 1:
idx = 0 + 32 * 1 = 32
idx = 1 + 32 * 1 = 33
...
idx = 31 + 32 * 1 = 63

...

Warp 31:
idx = 0 + 32 * 31 = 992
idx = 1 + 32 * 31 = 993
...
idx = 31 + 32 * 31 = 1023
```

Итого: паттерн обращения к памяти coalesced и для каждого Warp-а загрузится единственная кэш-линяя, то есть 32 кэш-линии на всю ворк группу

(b) как и в Задаче 2 распишем по Warp-ам как выглядят индексы

```
idx = get_local_id(1) + get_local_size(1) * get_local_id(0)

Warp 0:
idx = 0 + 32 * 0 = 0
idx = 0 + 32 * 1 = 32
...
idx = 0 + 32 * 31 = 992

Warp 1:
idx = 1 + 32 * 0 = 1
idx = 1 + 32 * 1 = 33
...
idx = 1 + 32 * 31 = 993

...

Warp 31:
idx = 31 + 32 * 0 = 31
idx = 31 + 32 * 1 = 63
...
idx = 31 + 32 * 31 = 1023
```

Итого: паттерн обращения к памяти НЕ coalesced и для каждого Warp-а загрузится 32 кэш-линии, по одной на каждый WorkItem, то есть 1024 кэш-линии на всю ворк группу

(c) как и в Задаче 2 распишем по Warp-ам как выглядят индексы

```
idx = 1 + get_local_id(0) + get_local_size(0) * get_local_id(1)

Warp 0:
idx = 1 + 0 + 32 * 0 = 1
idx = 1 + 1 + 32 * 0 = 2
...
idx = 1 + 31 + 32 * 0 = 32

Warp 1:
idx = 1 + 0 + 32 * 0 = 1
idx = 1 + 1 + 32 * 0 = 2
...
idx = 1 + 31 + 32 * 0 = 32

...

Warp 31:
idx = 1 + 0 + 32 * 31 = 993
idx = 1 + 1 + 32 * 31 = 994
...
idx = 1 + 31 + 32 * 31 = 1024
```

Итого: паттерн обращения к памяти coalesced, НО для каждого Warp-а загрузится 2 кэш-линии, по две на каждый WorkItem (для крайнего элемента нужна своя линия), то есть 64 кэш-линии на всю ворк группу