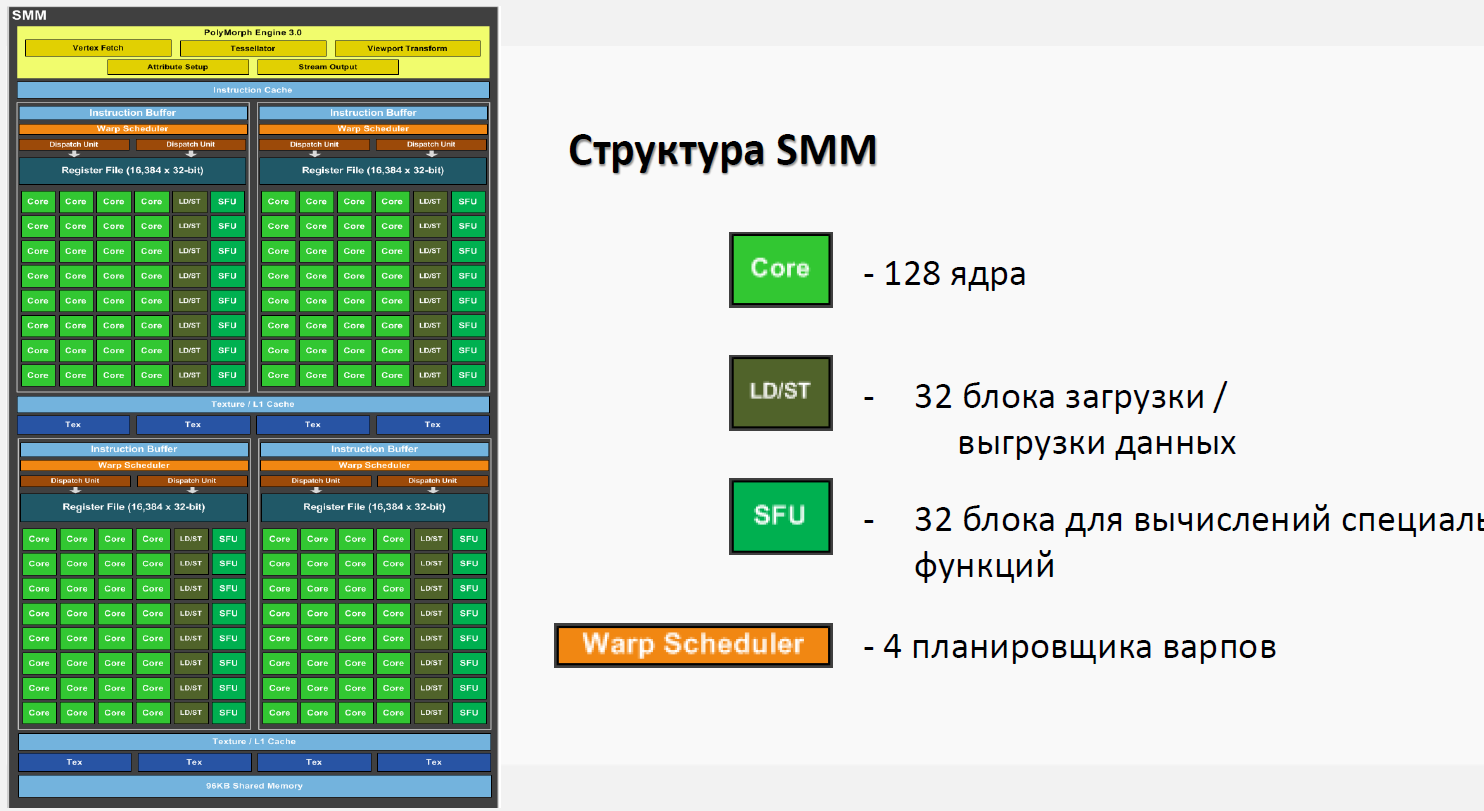
**1. Архитектура GPU. Основные отличия между GPU и CPU. Нити, блоки. Warp. Встроенные типы и переменные. (максимум 12 баллов)**

Видеокарта содержит Compute Units – процессорные ядра. Процессорные ядра могут выполнять несколько потоков за счет того, что в каждом содержится несколько (8-16) потоковых процессоров (Stream Cores или Stream Processor).

Отличия GPU (SIMD архитектура):

* очень высокая степень параллелизма
* основная часть чипа занята логикой, а не кэшем
* ограничение функциональности (для достижения высоких скоростей может выполнить лишь часть операций, которые может выполнить центральный процессор)
* Меньшая тактовая частота, гораздо большее количество ядер



****

**Спецификаторы функций**

|  |  |  |
| --- | --- | --- |
| **Спецификатор** | **Функция выполняется на** | **Функция может вызываться из** |
| \_\_device\_\_ | device | device |
| \_\_global\_\_ | device | host, device |
| \_\_host\_\_ | host | host |

**Спецификаторы переменных**

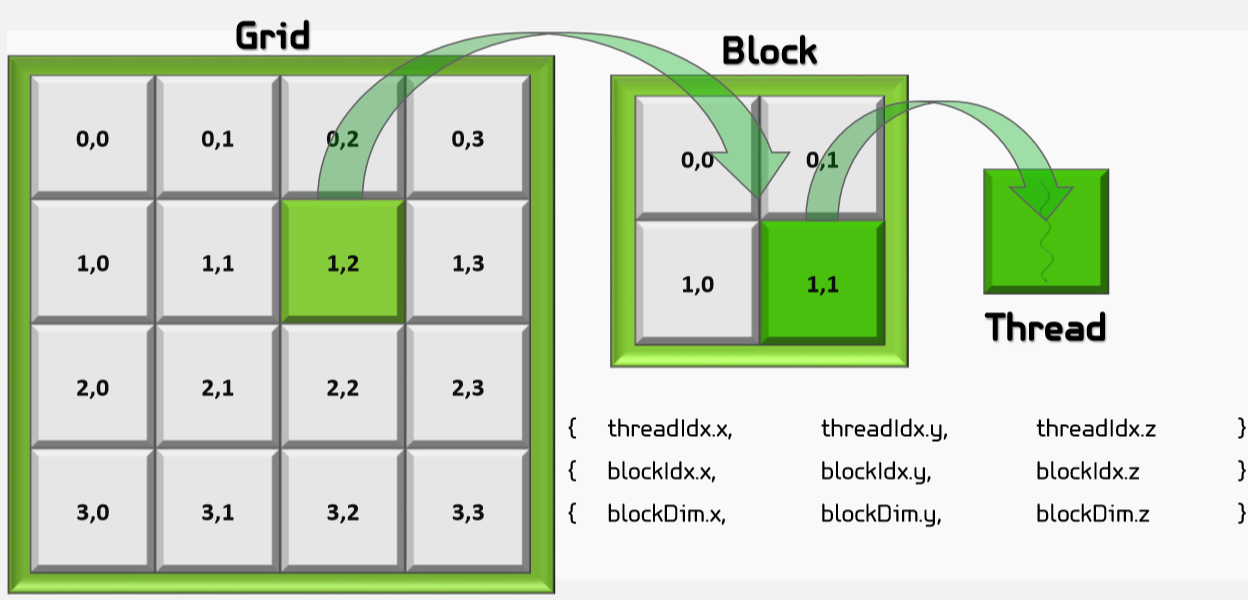
|  |  |  |  |
| --- | --- | --- | --- |
| **Спецификатор** | **Находится** | **Доступна** | **Вид доступа** |
| \_\_device\_\_ | device | device/host | R/W |
| \_\_constant\_\_ | device | device/host | R/o |
| \_\_shared\_\_ | device | block | R/W |

blockIdx - номер блока, в котором находится нить

blockDim - размер блока

**Линейный индекс нити:**

tid = threadIdx.x + blockIdx.x \* blockDim.x



**Thread** – единица выполнения программы. Имеет свой уникальный идентификатор внутри Block.

**Warp** – 32 последовательно идущих Thread

**Block** – Объединение потоков, которое выполняется целиком на одном SM. Имеет свой уникальный идентификатор внутри Grid.

**Grid** – Объединение блоков, которые выполняются на одном устройстве

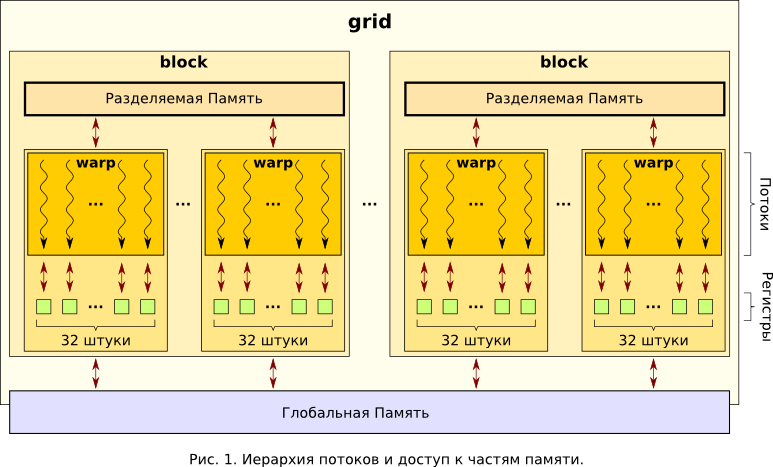
**Kernel** – Параллельная часть алгоритма, выполняемая на Grid

**Исполнение потоков**

* На каждом мультипроцессоре может обрабатываться несколько блоков потоков
* На каждом такте мультипроцессор исполняет одну и ту же инструкцию над группой потоков, называемой варп (**warp**)
* Количество потоков в варпе – warp size (32 потока)
* В случае условного ветвления, сначала исполняется одна ветвь над одной частью варпа, затем другая над оставшейся if( threadIdx.x < 8 ) { do smth } else { do smth else }

**Планировщик варпов**

* На каждом такте планировщик выбирает группу потоков и над каждым потоком из группы исполняется одна и та же команда.
* Планирование варпов осуществляется таким образом, чтобы максимально скрыть время ожидания данных из глобальной памяти: пока одни группы потоков ожидают данные, над другими исполняются инструкции.
* **Роль планировщика:** скрывать время ожидания данных одними потоками из глобальной памяти за счёт работы других потоков в это время.



* Ядро – это функция, которая исполняется над сеткой блоков потоков
* Каждый поток и блок имеет собственные координаты
* Блок потоков - ­ набор потоков, способных общаться между собой посредством
  + Разделяемой памяти
  + Точек синхронизации (барьеры)

**Вызов функции-ядра**

KernelName <<< nBlock, nThread, nShMem, nStream >>> ( args);

KernelName - название функции-ядра

nBlock - число блоков сети ( grid )

nThread - число нитей в блоке

nShMem - количество дополнительной разделяемой памяти, выделяемой на блок

nStream - номер потока из которого запускается функция-ядро

**2. Модели параллельных вычислений. (максимум 12 баллов)**

Часто, возникают конфликты между степенью распараллеливания и объемом коммуникаций, то есть чем между большим числом процессоров распределена задача, тем больший объем данных необходимо пересылать между ними.

***? Процесс/канал*** (*Process/Channel*)

В этой модели программы состоят из одного или более процессов, распределенных по процессорам. Процессы выполняются одновременно, их число может измениться в течение времени выполнения программы. Процессы обмениваются данными через каналы, которые представляют собой однонаправленные коммуникационные линии, соединяющие только два процесса. Каналы можно создавать и удалять.

***Обмен сообщениями*** (*Message Passing*)

В этой модели программы, возможно различные, написанные на традиционном последовательном языке исполняются процессорами компьютера. Каждая программа имеют доступ к памяти исполняющего е§ процессора. Программы обмениваются между собой данными, используя подпрограммы приема/передачи данных некоторой коммуникационной системы. Программы, использующие обмен сообщениями, могут выполняться только на MIMD компьютерах.

***Общей памяти*** (*Shared Memory*)

В этой модели все процессы совместно используют общее адресное пространство. Процессы асинхронно обращаются к общей памяти как с запросами на чтение, так и с запросами на запись, что создает проблемы при выборе момента, когда можно будет поместить данные в память, когда можно будет удалить их. Для управления доступом к общей памяти используются стандартные механизмы синхронизации - семафоры и блокировки процессов.

***Параллелизм данных*** (*Data Parallel*)

В этой модели единственная программа задает распределение данных между всеми процессорами компьютера и операции над ними. Распределяемыми данными обычно являются массивы. Как правило, языки программирования, поддерживающие данную модель, допускают операции над массивами, позволяют использовать в выражениях целые массивы, вырезки из массивов. Распараллеливание операций над массивами, циклов обработки массивов позволяет увеличить производительность программы. Компилятор отвечает за генерацию кода, осуществляющего распределение элементов массивов и вычислений между процессорами. Каждый процессор отвечает за то подмножество элементов массива, которое расположено в его локальной памяти. Программы с параллелизмом данных могут быть оттранслированы и исполнены как на MIMD, так и на SIMD компьютерах.

***Threads Model***

A thread is defined as a short sequence of instructions, within a process. Different threads can be executed on same processor or on different process. If the threads are executed on same processor, then the processor switches between the threads in a random fashion. If the threads are executed on different processors, they are executed simultaneously.

The threads communicate through global memory.

**Advantages:**

Programmer need not have to worry about parallel processing.

**Disadvantages:**

Care must be taken that no two threeds update the shared resources simultaneously.

Либо: -------------------------------------------------------

**– *Гранулярность*** - это мера отношения объема вычислений, выполненных в параллельной задаче, к объему коммуникаций (для обмена сообщениями).

Степень гранулярности варьируется от мелкозернистой до крупнозернистой.

**– *Крупнозернистый параллелизм (coarse grained)*:** каждое параллельное вычисление достаточно независимо от остальных, причем требуется относительно редкий обмен информацией между отдельными вычислениями. Единицами распараллеливания являются большие и независимые программы, включающие тысячи команд. Этот уровень параллелизма обеспечивается операционной системой.

Фактически, разные процессоры асинхронно выполняют различные программы.

**– *Среднезернистый параллелизм (medium grained)*:** единицами распараллеливания являются вызываемые процедуры, включающие в себя сотни команд. Обычно организуется как программистом, так и компилятором.

**– *Мелкозернистый параллелизм (fine grained)*:** каждое параллельное вычисление достаточно мало и элементарно, составляется из десятков команд. Обычно распараллеливаемыми единицами являются элементы выражения или отдельные итерации цикла, имеющие небольшие зависимости по данным. Сам термин «мелкозернистый параллелизм» говорит о простоте и быстроте любого вычислительного действия. Характерная особенность мелкозернистого параллелизма заключается в приблизительном равенстве интенсивности вычислений и обмена данными. Этот уровень параллелизма часто используется распараллеливающим (векторизирующим) компилятором.

Эффективное параллельное исполнение требует искусного баланса между степенью гранулярности программ и величиной коммуникационной задержки, возникающей между разными гранулами. В частности, если коммуникационная задержка минимальна, то наилучшую производительность обещает мелкоструктурное разбиение программы. Это тот случай, когда действует параллелизм данных. Если коммуникационная задержка велика, предпочтительней крупнозернистое разбиение программ.

Оно же: ----------------------------------------------------

**Уровни параллелизма**

Методы и средства реализации параллелизма зависят от того, на каком уровне он Должен обеспечиваться. Обычно различают следующие уровни параллелизма:

■ **Уровень заданий**. Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Этот уровень реализуется на ВС с множеством процессоров в многозадачном режиме.

■ **Уровень программ**. Части одной задачи выполняются на множестве процессоров. Данный уровень достигается на параллельных ВС.

■ **Уровень команд**. Выполнение команды разделяется на фазы, а фазы нескольких последовательных команд могут быть перекрыты за счет конвейеризации. Уровень достижим на ВС с одним процессором.

■ **Уровень битов** (арифметический уровень). Биты слова обрабатываются один за другим, это называется бит - последовательной операцией. Если биты слова обрабатываются одновременно, говорят о бит - параллельной операции. Данный уровень реализуется в обычных и суперскалярных процессорах.

**3. Параллельное вычисление алгоритма обратного распространения ошибки на GPU при обучении ИНН. (максимум 6 баллов)**