Технология CUDA для высокопроизводительных вычислений на кластерах с графическими процессорами

Колганов Александр
alexander.k.s@mail.ru

часть 7

Иерархия памяти

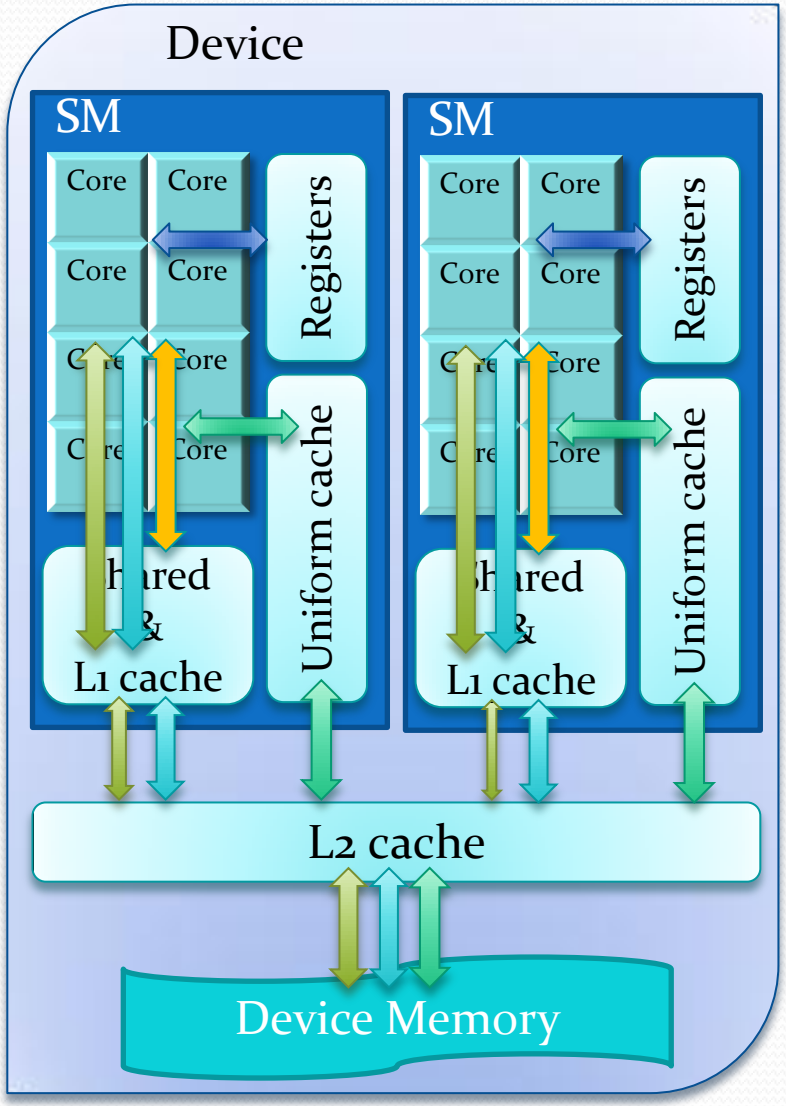
Глобальная

Общая

Константная

Локальная

Регистровая



Ответьте на вопросы

- Какова длина кеш-линии L1?
- В чем проблема разреженного доступа к глобальной памяти?
- Зачем нужная локальная память?
- Когда неявно используется локальная память?
- Что такое «однородное»(uniform) обращение в память?
- В чем особенность константной памяти?
- Когда неявно используется константная память?
- Что такое банк-конфликт порядка n?
- Зачем нужна синхронизация внутри блока при работе с общей памятью?
- Факторы, влияющие на Occupancy

время выполнения задачи =
время работы ядра + обмен данными
между CPU и GPU

- Как сократить накладные расходы на обмен данными между CPU и GPU?
  - Ускорение копирований
  - Выполнение копирований параллельно с вычислениями

Pinned-память

DMA & zero-copy

- “Zero-copy”- копирование памяти без участия центрального процессора
  - Копирование выполняется спец. контроллером, процессор переключается на другие задачи
- DMA (Direct memory access) – прямой доступ к оперативной памяти, без участия ЦП
  - Реализуется через zero-copy операции
  - Скорость передачи увеличивается, так как данные не пересылаются в ЦП и обратно

DMA и виртуальная память

- Виртуальная память организованна в страницы, отображаемые на физические страницы ОП

- Виртуальные страницы могут быть
  - Перемещены в оперативной памяти
  - Отгружены на диск (*swapping*)

В таких условиях реализовать DMA очень сложно!

DMA и виртуальная память

- Запретим перемещение страниц по ОП и их выгрузку на диск
  - Привяжем страницы виртуальной памяти к страницам физической
  - Эти физические страницы теперь недоступны ОС для размещения новых виртуальных страниц (paging)

page-able → page-locked

Для page-locked памяти DMA реализуемо без существенных накладных расходов

Page-locked память & CUDA

- «Pinned» - синоним, «прикрепленный»
- В CUDA можно напрямую выделить page-locked (pinned) память на хосте или сделать таковой память, выделенную ранее
- Операции копирования Host ←→ Device с ней происходят быстрее и могут выполняться параллельно с работой ядер

cudaHostRegister

- “Залочить” память, предварительно выделенную обычным способом:

```
float *ptr = malloc(n * sizeof(float))
cudaHostRegister(ptr, n * sizeof(float),0);
cudaMemcpy(devPtr,prt, n * sizeof(float),
                        cudaMemcpyHostToDevice)
…
cudaHostUnregister(ptr);
```

Mapped pinned-память

- Pinned-память можно отобразить в виртуальное адресное пространство GPU
- Нити смогут обращаться к ней напрямую, без необходимости копирования в память GPU
  - Необходимые копирования будут неявно выполняться асинхронно, параллельно с работой ядра
- С хоста память будет так же доступна

Mapped pinned-память

- Pinned-память можно отобразить в виртуальное адресное пространство GPU

```
cudaHostRegister(ptr,n * sizeof(float),
                 cudaHostRegisterMapped);
float *ptrForDevice = NULL;
cudaHostGetDevicePointer(&ptrForDevice, ptr, 0);
// не нужно выделять память на GPU и копировать в
// неё входные данные
kernel<<<…>>>(ptrForDevice,…);
```

Mapped pinned-память

- Для активации возможности маппирования pinned-памяти:
  - До первого вызова функции из cuda-runtime (т.е. до инициализации устройства) установить флаг инициализации `cudaDeviceMapHost`:

```
cudaSetDeviceFlags(cudaDeviceMapHost);
cudaSetDevice(0);// инициализируется с флагами
```

  - Проверить свойство устройства `canMapHostMemory`:

```
cudaDeviceProp deviceProp;
cudaGetDeviceProperties(0, &deviceProp);
if (deviceProp.canMapHostMemory) {
    …
}
```

Portable pinned-память

- По-умолчанию, преимущества pinned-памяти доступны только для устройства, которое было активным в момент выделения
- Чтобы блок мог pinned-памяти одинаково использоваться всеми устройствами, нужно передавать флаг `cudaHostRegisterPortable`

```
cudaHostRegister(ptr,n * sizeof(float),
                 cudaHostRegisterPortable |
                 cudaHostRegisterMapped );
```

Прямое выделение pinned-памяти

- Самое простое:

```
float *ptr = NULL;
cudaMallocHost(&ptr, n * sizeof(float));
```

- С флагами:

```
cudaHostAlloc(&ptr, n * sizeof(float),
                    cudaHostAllocDefault);
```

Возможные флаги:

- `cudaHostAllocDefault`: эмулирование `cudaMallocHost`().
- `cudaHostAllocPortable`: аналогично `cudaHostRegisterPortable`
- `cudaHostAllocMapped`: аналогично `cudaHostRegisterMapped`
- `cudaHostAllocWriteCombined`: см. далее

Write-combined память

- При указании флага `cudaHostAllocWriteCombined` выделится **write-combined pinned** память:
  - Записи в такую память производятся не сразу, а складываются в спец. буфер (write-combine buffer)
  - В некоторый момент, записи в буфере «комбинируются» и в итоге выполняются с минимальным числом обращений в память

Write-combined память

- В мультипроцессорных системах кэши следят за обычной памятью через шину (snooping), выполняя протокол обеспечения когерентности
- Write-combined память **не кешируется**
  - => Кэши не следят за этой памятью через шину

Write-combined память

- Освобождаются ресурсы кэшей L1 и L2 ЦП
- Кэши не следят за такой памятью через шину
  - Копирование такой памяти через шину выполняется на 40% быстрее
- Медленное чтение с хоста

Если аппаратура позволяет выделять такую память, то стоит использовать её для буферов, записываемых на хосте и пересылаемых на GPU (или читаемых напрямую через mapped память)

cudaMemcpy* и pinned-память

- Копировании обычной (pageable) памяти с хоста на GPU происходит через DMA к промежуточному буферу в pinned-памяти
- Управление хосту возвращается после выполнения копирований в этот буфер, но необязательно до завершения DMA

cudaMemcpy* и pinned-память

- Копировании обычной (pageable) памяти с хоста на GPU происходит через DMA к промежуточному буферу в pinned-памяти
  - Поэтому копирование сразу из pinned-памяти быстрее – не нужно выделять память под буфер и копировать в него данные

Тест

```
float *hostPtr = (float *)malloc(numberOfBytes);
cudaEventRecord(startPageable, 0);
cudaMemcpy(devicePtr, hostPtr, numberOfBytes,
                              cudaMemcpyHostToDevice);
cudaEventRecord(stopPageable, 0);

cudaHostRegister(hostPtr, numberOfBytes, 0);

cudaEventRecord(startPinned, 0);
cudaMemcpy(devicePtr, hostPtr, numberOfBytes,
                              cudaMemcpyHostToDevice);
cudaEventRecord(stopPinned, 0);
cudaDeviceSynchronize();
```

Тест

```
float elapsedPinned, elapsedPageable;
cudaEventElapsedTime(&elapsedPageable, startPageable,
                                stopPageable);
cudaEventElapsedTime(&elapsedPinned, startPinned,
                                stopPinned);
printf("Copy from pageable %f\n", elapsedPageable);
printf("Copy from pinned %f\n", elapsedPinned);
```

```
$./a.out
Copy from pageable 947.509155
Copy from pinned 339.375580
```

Замечания

- Выделение pinned-памяти занимает больше времени, чем обычный malloc
- Доступный для выделения объем сильно ограничен
- Чрезмерное использование page-locked памяти деградирует систему
  - Для освобождения использовать `cudaFreeHost()`, `cudaHostUnregister()`

UVA

Unified Virtual Address (UVA)

- На 64-битной архитектуре, начиная с поколения Fermi (cc 2.0), используется единое виртуальное адресное пространство для памяти хоста и всех устройств
  - Unified Virtual Address space, UVA

- Если UVA включено, то

```
cudaDeviceProp::unifiedAddressing == 1
```

Unified Virtual Address (UVA)

- Без UVA для каждого указателя хранятся метаданные о том где реально расположена память, на которую он указывает
- С UVA эта информация «вшита» в значение указателя
  - Диапазоны адресов всех GPU и CPU не пересекаются

Unified Virtual Address (UVA)

- Чтобы узнать где реально расположена память:

```
float *ptr;
cudaPointerAttributes pointerAttributes;
cudaPointerGetAttribute
                        (&pointerAttributes, ptr)
```

Unified Virtual Address (UVA)

```
struct cudaPointerAttributes {
    enum cudaMemoryType memoryType;
    int device;
    void *devicePointer;
    void *hostPointer;
}
```

- **memoryType** - cudaMemoryTypeHost | cudaMemoryTypeDevice
- **device** - устройство, на котором расположена память
- **devicePointer** - NULL, если не доступна с текущего устройства
- **hostPointer** – NULL, если не доступна с хоста

Pinned-память и UVA

- С UVA память, выделенная через `cudaHostAlloc()`
  - Автоматически является **portable**
  - Автоматически является **mapped**
  - Доступна с хоста и с любого GPU по одному и тому же указателю (т.к. адресное пространство единое)
    - Не нужно использовать `cudaHostGetDevicePointer()`
    - Исключение – `cudaHostAllocWriteCombined`

Pinned-память и UVA

- Для памяти, залоченной через `cudaHostRegister` и для **write-combined** памяти указатели для хоста и для устройства являются разными
  - Нужен `cudaHostGetDevicePointer()`

Пример

- Без UVA и mapped памяти:

```
float *ptr = NULL;
cudaHostAlloc(&ptr, 1024, 0);
float *ptrForDevice = NULL;
cudaMalloc(&ptrForDevice, 1024);
cudaMemcpy(ptrForDevice, ptr, 1024,
                    cudaMemcpyHostToDevice)
kernel<<<…>>>(ptrForDevice,…);
```

Пример

- С mapped памятью:

```
cudaSetDeviceFlags(cudaDeviceMapHost);
cudaDeviceProp deviceProp;
cudaGetDeviceProperties(device, &deviceProp);
if (deviceProp.canMapHostMemory ) {
   float *ptr = NULL;
   cudaHostAlloc(&ptr, 1024, cudaHostAllocMapped)
   float *ptrForDevice = NULL;
   cudaHostGetDevicePointer(&ptrForDevice, ptr, 0);
   kernel<<<…>>>(ptrForDevice,…);
}
```

Пример

- С mapped памятью и UVA:

```
cudaSetDeviceFlags(cudaDeviceMapHost)
cudaDeviceProp deviceProp;
cudaGetDeviceProperties(device, &deviceProp);
if (deviceProp.unifiedAddressing ) {
   float *ptr = NULL;
   cudaHostAlloc(&ptr, 1024, cudaHostAllocMapped)
   kernel<<<…>>>(ptr,…);
}
```

Пример

```
float *ptrForDevice = NULL;
if (deviceProp.unifiedAddressing ) {
   ptrForDevice = ptr
} else if (deviceProp.canMapHostMemory ) {
   cudaHostGetDevicePointer(&ptrForDevice, ptr, 0);
} else {
   cudaMalloc(&ptrForDevice, 1024);
   cudaMemcpy(ptrForDevice, ptr, 1024,
                  cudaMemcpyHostToDevice)
}
kernel<<<…>>>(ptrForDevice,…);
```

cudaMemcpy* и UVA

- С UVA система в состоянии сама определить где находится память
  - Можно указывать `cudaMemcpyDefault` в `cudaMemcpyKind`:

```
float *dstPtr, *srcPtr;
cudaMemcpy(dstPtr, srcPtr,
           n*sizeof(float), cudaMemcpyDefault)
```

Выводы

Выводы

время выполнения задачи =

время работы ядра + обмен данными между CPU и GPU

- page-locked (pinned) память позволяет
  1. Уменьшить время обмена данными
  2. Упростить хост-код при использовании mapped pinned памяти и доступе к ней напрямую из ядер
     - Не нужно возиться с пересылкой данных на GPU и обратно
     - С UVA обращаемся к памяти с хоста и с устройства по одному указателю