CUDA Driver API

Романенко А.А.
arom@ccfit.nsu.ru
Новосибирский государственный университет

* Привязано ли ядро к исполняемому коду?
* Можно ли запускать ядро не используя расширения языка Си?
* Можно ли программировать на CUDA не на Си/Си++?

Объекты в CUDA driver API

* **Device** — CUDA-совместимое устройство
* **Context** — «эквивалент» процессу для CPU
* **Module** — «эквивалент» динамической библиотеки
* **Function** — ядро
* **Heap memory** — указатель на память устройства
* **CUDA Array** — контейнер для 1D или 2D массивов на устройстве, доступных через текстуру
* **Texture reference** — объект для описания данных в текстуре

Порядок работы

* Инициализация драйвера
* Выбор устройства (GPU)
* Создание контекста
* Работа в рамках контекста
  * Ядра или в формате PTX или бинарном формате
* Удаление контекста

Инициализация драйвера

* CUresult cuInit(unsigned int flag);
  * Flag = 0
* CUT_DEVICE_INIT_DRV(cuDevice, ARGC, ARGV)
* Без инициализации все функции будут возвращать CUDA_ERROR_NOT_INITIALIZED

Управление устройствами (1)

* CUresult **cuDeviceGetCount**(int *count)
* CUresult **cuDeviceGet**(CUdevice *device, int ordinal)
* CUresult **cuDeviceComputeCapability**(int *major, int *minor, CUdevice dev)
* CUresult **cuDeviceTotalMem**(unsigned int *bytes, CUdevice dev)
* CUresult **cuDeviceGetAttribute**(int *pi, CUdevice_attribute attrib, CUdevice dev)

Атрибуты устройства (1)

* CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK
* CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X
* CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y
* CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z
* CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X
* CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y
* CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z
* CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK
* CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY
* CU_DEVICE_ATTRIBUTE_WARP_SIZE
* CU_DEVICE_ATTRIBUTE_MAX_PITCH

Атрибуты устройства (2)

* CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK
* CU_DEVICE_ATTRIBUTE_CLOCK_RATE
* CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT
* CU_DEVICE_ATTRIBUTE_GPU_OVERLAP
* CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT
* CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT
* CU_DEVICE_ATTRIBUTE_INTEGRATED
* CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY
* CU_DEVICE_ATTRIBUTE_COMPUTE_MODE
  * CU_COMPUTEMODE_DEFAULT
  * CU_COMPUTEMODE_EXCLUSIVE
  * CU_COMPUTEMODE_PROHIBITED
* пр.

Управление устройствами (2)

* CUresult cuDeviceGetProperties (CUdevprop *prop, CUdevice dev)

```
typedef struct CUdevprop_st {
        int maxThreadsPerBlock;
        int maxThreadsDim[3];
        int maxGridSize[3];
        int sharedMemPerBlock;
        int totalConstantMemory;
        int SIMDWidth;
        int memPitch;
        int regsPerBlock;
        int clockRate;
        int textureAlign
} CUdevprop;
```

Контекст CUDA

* Контекст CUDA — аналог процесса для CPU
* В рамках потока может быть только один активный контекст CUDA
* При создании контекста (cuCtxCreate) счетчик использования равен 1
* **cuCtxAttach**()* увеличивает счетчик, **cuCtxDetatch**()* уменьшает счетчик на 1
* Контекст разрушается когда счетчик использования становиться равным 0 или явно вызывается **cuCtxDestroy**()
* Активный контекст может меняться. **cuCtxPopCurrent**(), **cuCtxPushCurrent**()

* - в CUDA 4.0 объявлены как depricated

Модули в CUDA (1)

* Модуль — динамически подгружаемый объект с ядрами (kernel). Аналог DLL файлов
* Модули собираются с помощью nvcc.
  Могут распространятся независимо.
  * make -keep
  * nvcc --keep
* CUmodule cuModule;
  cuModuleLoad(&cuModule, "module.cubin");
  CUfunction cuFunc;
  cuModuleGetFunction(&cuFunc, cuModule, "myKernel");

Модули в CUDA (2)

```
#define ERROR_BUFFER_SIZE 100
CUmodule cuModule;
CUjit_option options[3];
void* values[3];
char* PTXCode = "some PTX code";
options[0] = CU_ASM_ERROR_LOG_BUFFER;
values[0] = (void*)malloc(ERROR_BUFFER_SIZE);
options[1] = CU_ASM_ERROR_LOG_BUFFER_SIZE_BYTES;
values[1] = (void*)ERROR_BUFFER_SIZE;
options[2] = CU_ASM_TARGET_FROM_CUCONTEXT;
values[2] = 0;
cuModuleLoadDataEx(&cuModule, PTXCode, 3,
                        options, values);
for (int i = 0; i < values[1]; ++i) {
   // Parse error string here
}
```

Управление модулями

- **cuModuleLoad()** - загрузка модуля из cubin файла
- **cuModuleLoadData()** - загрузка модуля из PTX строки
- **cuModuleLoadDataEx()** - загрузка модуля из PTX строки с возвратом ошибок
- **cuModuleLoadFatBinary()** - загрузка модуля из «жирного» cubin файла
  - Стало доступно в CUDA 4.0
  - nvcc -fatbin
- **cuModuleUnload()** - выгрузка модуля

Управление исполнением

* Задание конфигурации потокового блока
* Задание конфигурации сети
* Передача параметров функций
* Задание размеров разделяемой памяти

* Запуск ядра

Управление исполнением

* cuFuncSetBlockShape()
* cuFuncSetSharedSize()
* cuLaunch()
* cuLaunchGrid()
* cuLaunchGridAsync()

Управление исполнением

* cuFuncSetBlockShape()
* cuFuncSetSharedSize()
* cuLaunch()
* cuLaunchGrid()
* cuLaunchGridAsync()

CUDA 3.2 и более ранние версии

CUDA 4.0

```
cuLaunchKernel (
        function,
        gDimX, gDimY, gDimZ,
        bDimX, bDimY, bDimZ,
        sharedMemBytes,
        Stream,
        kernelParams, extra
    )
```

Передача параметров

* cuParamSetf ()
* cuParamSeti()
* cuParamSetSize ()
* cuParamSetTexRef()
* cuParamSetv()

Передача параметров

* cuParamSetf ()
* cuParamSeti()
* cuParamSetSize ()
* cuParamSetTexRef()
* cuParamSetv()

CUDA 3.2 и более ранние версии

CUDA 4.0

Передача через последние 2 параметра cuLaunchKernel (... kernelParams, extra)

kernelParams — массив указателей на параметры

extra — параметры, упакованные в один массив

Передача параметров

* std::vector< void* > kernelParams;
  float *dev_in1; float * dev_in2; float *dev_out;

  kernelParams.push_back(&dev_in1);
  kernelParams.push_back(&dev_in2); kernelParams.push_back(&dev_out);
  kernelParams.push_back(const_cast< int*>(&VEC_SIZE));

* const size_t sharedMemSize = 0;
  const CUstream stream = 0;
* *// equivalent to*
  // vecSum<<<GS, BS, 0, 0>>>(dev_in1, dev_in2, dev_out, VEC_SIZE);
* status = cuLaunchKernel(function,
  GS.x, GS.y, GS.z, BS.x, BS.y, BS.z,
* sharedMemSize, stream, &kernelParams[0], 0);

Передача параметров

```
#define ALIGN_UP(offset, alignment) (offset) = ((offset)+(alignment) - 1) & ~((alignment) - 1)
char paramBuffer[1024];
size_t paramBufferSize = 0;
#define ADD_TO_PARAM_BUFFER(value, alignment) { \
                paramBufferSize = ALIGN_UP(paramBufferSize, alignment); \
                memcpy(paramBuffer + paramBufferSize, &(value), sizeof(value)); \
                paramBufferSize += sizeof(value); }
CUdeviceptr dev_in1, dev_in2, dev_out;
ADD_TO_PARAM_BUFFER(dev_in1, __alignof(dev_in1));
ADD_TO_PARAM_BUFFER(dev_in2, __alignof(dev_in2));
ADD_TO_PARAM_BUFFER(dev_out, __alignof(dev_out));
ADD_TO_PARAM_BUFFER(VEC_SIZE, __alignof(VEC_SIZE));
void *config[] = {
    CU_LAUNCH_PARAM_BUFFER_POINTER, paramBuffer,
    CU_LAUNCH_PARAM_BUFFER_SIZE,    &paramBufferSize,
    CU_LAUNCH_PARAM_END
};
status = cuLaunchKernel(f, gx, gy, gz, bx, by, bz, sh, s, NULL, config);
```

Управление памятью

* CUresult **cuMemAlloc** (CUdeviceptr *dptr, unsigned int size)
* CUresult **cuMemAllocHost** (void **pp, unsigned int bytesize)
* CUresult **cuMemAllocPitch** (CUdeviceptr *dptr, unsigned int *pPitch, unsigned int WidthInBytes, unsigned int Height, unsigned int ElementSizeBytes)
* CUresult **cuMemFree** (CUdeviceptr dptr)
* CUresult **cuMemFreeHost** (void *p)
* CUresult **cuMemcpy*** - функции копирования между массивами, памятью GPU и CPU

Управление текстурами

* cuTexRefCreate()
* cuTexRefDestroy()
* cuModuleGetTexRef()
* cuTexRefSetAddress()
* cuTexRefSetArray()
* cuTexRefSetFilterMode()
* cuTexRefSetAddressMode()
  * CU_TR_ADDRESS_MODE_WRAP,
  * CU_TR_ADDRESS_MODE_CLAMP,
  * CU_TR_ADDRESS_MODE_MIRROR,
  * CU_TR_ADDRESS_MODE_BORDER

} Нет документации

* cuTexRefSetFlags()

CUDA driver API vs. runtime API

* Runtime API основано на driver API
* Runtime API работает в рамках контекста, созданного через driver API. Если контекста нет, то он создается неявно перед первым вызовом функции из runtime API.
* Driver API предоставляет большую гибкость
  * Получение дополнительной информации об устройстве, например, объем свободной памяти (cuMemGetInfo)
  * пр.
* Используя driver API
  * не происходит явная интеграция кода ядер в программу
  * нет возможности пускать программу в режиме эмуляции
  * усложняется процесс написания программы и отладки