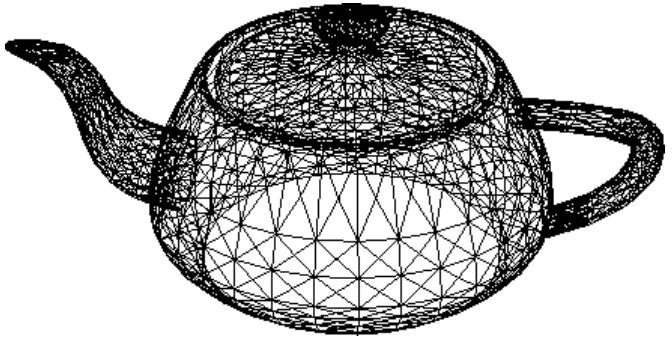
Текстуры и поверхности в CUDA

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

Что такое текстура?



+

* Способ доступа к данным

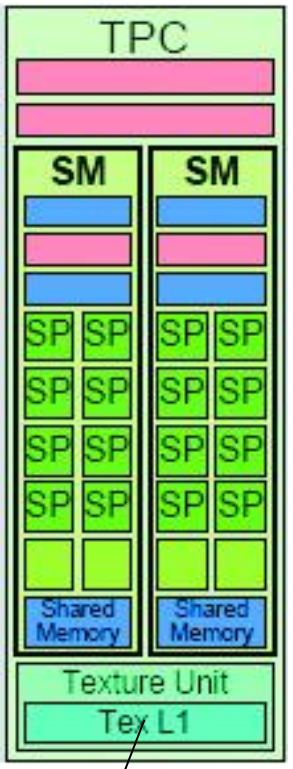
Особенности текстур

* Латентность больше, чем у прямого обращения в память
* Дополнительные стадии в конвейере:
  * Преобразование адресов
  * Фильтрация
  * Преобразование данных
* Но зато есть кэш
* Разумно использовать, если:
  * Объем данных не влезает в shared память
  * Паттерн доступа хаотичный
  * Данные переиспользуются разными потоками

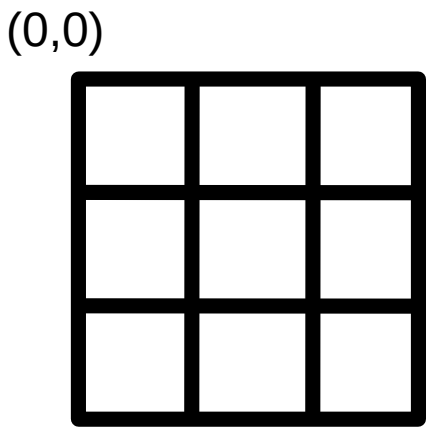
Свойства текстур

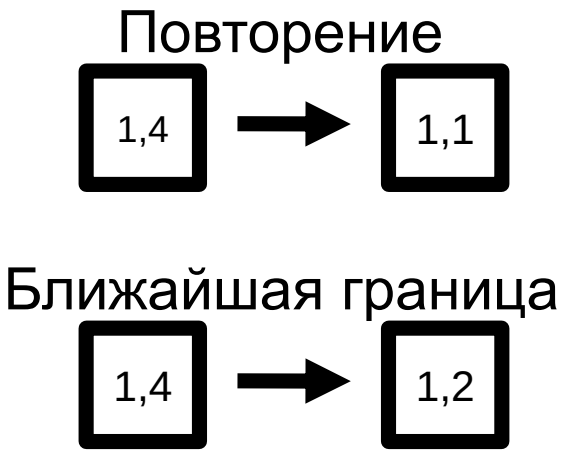
* Доступ к данным через кэш
  * Оптимизированы для доступа к данным которые расположены рядом в двумерном пространстве
* Фильтрация
* Свертывание (выход за границы)
  * Повторение/ближайшая граница
* Адресация в 1D, 2D и 3D
* Целые/нормализованные координаты

Свойства в картинках



Кэш текстуры





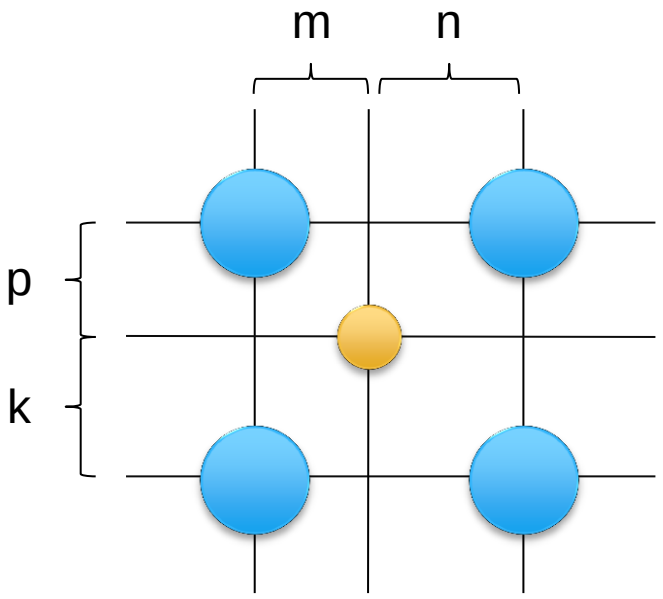
- tex1Dfetch(texRef, x)
- tex1D(texRef, x)
- tex2D(texRef, x, y)
- tex3D(texRef, x, y, z)

Свойства в картинках

* Нормализация координат

  [0…n] -> [0…1]

* Фильтрация



$$U = V_{11}*n*k + V_{12}*m*k + V_{21}*n*p + V_{22}*m*p$$

Свойства текстур

* Преобразование данных:
  * cudaReadModeNormalizedFloat :
    * Исходный массив содержит данные в integer,
    * возвращаемое значение во floating point представлении (доступный диапазон значений отображается в интервал [0, 1] или [-1,1])
  * cudaReadModeElementType
    * Возвращаемое значение то же, что и во внутреннем представлении

Типы/свойства текстур

* Привязанные к линейной памяти
  * Только 1D
  * Целочисленная адресация
  * Фильтры и свертывание отсутствуют
* Привязанные к массивам CUDA
  * 1D, 2D или 3D
  * целые/нормализованные координаты
  * Фильтрация
  * Свертывание

Работа с текстурами

* Host:
  * Выделить память (cudaMalloc/cudaMallocArray/...)
  * Объявить указатель на текстуру
  * Связать указатель на текстуру с областью памяти
  * После использования:
    * Отвязать текстуру, освободить память
* Device:
  * Чтение данных через указатель текстуры
  * Текстуры для линейной памяти: tex1Dfetch()
  * Текстуры на массивах: tex1D() or tex2D() or tex3D()

Работа с текстурами (Host)

```
texture<float, 2, cudaReadModeElementType> tex;
        ...
cudaChannelFormatDesc channelDesc =
                cudaCreateChannelDesc(32, 0, 0, 0, cudaChannelFormatKindFloat);
cudaArray* cu_arr;
cudaMallocArray(&cu_arr, &channelDesc, width, height );
cudaMemcpyToArray(cu_arr, 0, 0, h_dta, size, cudaMemcpyHostToDevice);
// set texture parameters
        tex.addressMode[0] = cudaAddressModeWrap;
        tex.addressMode[1] = cudaAddressModeWrap;
        tex.filterMode = cudaFilterModeLinear;
        tex.normalized = true;    // access with normalized texture coordinates
// Bind the array to the texture
        cudaBindTextureToArray(tex, cu_arr, channelDesc);
```

Работа с текстурами (Device)

```
__global__ void Kernel( float* g_odata, int width, int height, float theta) {
        // calculate normalized texture coordinates
        unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
        unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;

        float u = x / (float) width;
        float v = y / (float) height;

        // transform coordinates
        u -= 0.5f;
        v -= 0.5f;

        float tu = u*cosf(theta) - v*sinf(theta) + 0.5f;
        float tv = v*cosf(theta) + u*sinf(theta) + 0.5f;

        // read from texture and write to global memory
        g_odata[y*width + x] = tex2D(tex, tu, tv);
}
```

Тип double и текстуры

* Тип double не поддерживается.
* Представить double как два значения типа int
  * texture<int2,1> my_texture;

```
static __inline__ __device__
    double fetch_double(texture<int2, 1> t, int i) {
                int2 v = tex1Dfetch(t,i);
                return __hiloint2double(v.y, v.x);
}
```

Пример

```
__global__ void kern(double *o){
        unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
        if(x<32){
              o[x] = fetch_double(my_texture, x)*2.0;
        }
}
int main(int argc, char *argv[]){
        double hbuf[32];    double *dob;    double *dbuf;
        size_t ii;
        cudaMalloc((void**)&dbuf, sizeof(double)*32);
        cudaMalloc((void**)&dob, sizeof(double)*32);
        cudaBindTexture(&ii, my_texture, dbuf,
            cudaCreateChannelDesc(32,32,0,0, cudaChannelFormatKindSigned));
        for(i = 0 ; i < 32 ; i++)   hbuf[i]=1.0/3.0*i;
        cudaMemcpy(dbuf, hbuf, 32*sizeof(double), cudaMemcpyHostToDevice);
        kern<<<1, 32>>>(dob);
        cudaMemcpy(hbuf, dob, 32*sizeof(double), cudaMemcpyDeviceToHost);
        for(i = 0 ; i < 32 ; i++)   printf("%lf\t", hbuf[i]);
        printf("\n");
        return 0;
}
```

Поверхности (Surface)

* Появились в CUDA 3.2
* Из поверхностей можно как читать, так и писать в них.
* Объявление
  * surface<void, Dim> surface_ref;
* Привязка к массивам
  * surface <void, 2> surfRef;
  * cudaBindSurfaceToArray(surfRef, cuArray);

Поверхности. Адресация

* Побайтовая адресация
* При указатели на float
  * Текстуры tex1d(texRef1D, x)
  * Поверхности surf1Dread(surfRef1D, 4*x)
  * Текстуры tex2d(texRef2D, x, y)
  * Поверхности surf2Dread(surfRef2D, 4*x, y)

Пример

```
// 2D surfaces
surface<void, 2> inputSurfRef;
surface<void, 2> outputSurfRef;

// Simple copy kernel
__global__ void copyKernel(int width, int height) {
  // Calculate surface coordinates
  unsigned int x = blockIdx.x * blockDim.x + threadIdx.x;
  unsigned int y = blockIdx.y * blockDim.y + threadIdx.y;
  if (x < width && y < height) {
    uchar4 data;
    // Read from input surface
    surf2Dread(&data, inputSurfRef, x * 4, y);
    // Write to output surface
    surf2Dwrite(data, outputSurfRef, x * 4, y);
  }
}
```

Пример. Продолжение

```
int main() {
    cudaChannelFormatDesc channelDesc =
    cudaCreateChannelDesc(8,8,8,8,
                          cudaChannelFormatKindUnsigned);
    cudaArray* cuInputArray; cudaArray* cuOutputArray;
    cudaMallocArray(&cuInputArray, &channelDesc, width,
                        height, cudaArraySurfaceLoadStore);
    cudaMallocArray(&cuOutputArray, &channelDesc, width,
                        height, cudaArraySurfaceLoadStore);
    cudaMemcpyToArray(cuInputArray, 0, 0, h_data, size,
                                    cudaMemcpyHostToDevice);
    cudaBindSurfaceToArray(inputSurfRef, cuInputArray);
    cudaBindSurfaceToArray(outputSurfRef, cuOutputArray);
    // Invoke kernel
    dim3 dimB(16, 16);
    dim3 dimG((width + dimB.x - 1)/dimB.x,
              (height + dimB.y - 1)/ dimB.y);

    copyKernel<<<dimGrid, dimBlock>>>(width, height);
    cudaFreeArray(cuInputArray); cudaFreeArray(cuOutputArray);
}
```

Прочие типы текстур и поверхностей

* Layered texture/surface
  * Только на базе CUDA array (**cudaMalloc3DArray()**)
* Cubemap texture/surface
  * Только на базе CUDA array (**cudaMalloc3DArray()**)
* Cubmap layered texture/surface
  * Только на базе CUDA array (**cudaMalloc3DArray()**)
* Texture gather