Содержание

- Пиковая производительность GPU
- Разделяемая память
- Шаблон работы с разделяемой памятью
- Оптимизация работы с разделяемой памятью
- Пример – умножение плотных матриц.

Пиковая производительность GPU

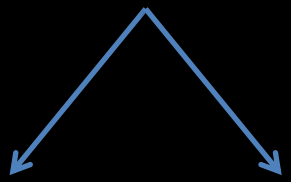
- Операция FMAD(a, b, c) == a = b * c + a
- Пиковая производительность:

  CUDA_CORES * FREQ * FLOPS/CLOCK

Пиковая производительность GPU

- Операция FMAD(a, b, c) == a = b * c + a
- Пиковая производительность:

CUDA_CORES * FREQ * FLOPS/CLOCK



SM_COUNT * CUDA_CORES / SM

Пиковая производительность GPU

- Операция FMAD(a, b, c) == a = b * c + a
- Пиковая производительность:

  CUDA_CORES * FREQ * FLOPS/CLOCK

GTX 580: 512 cores * 780 MHz * 2 = 0.8 TFLOPs (Fermi 2.0)

GTX 680: 1536 cores * 1000 MHz * 2 = 3.0 TFLOPs (Kepler 3.0)

GTX 780Ti: 2880 cores * 954 MHz * 2 = 5.4 TFLOPs (Kepler 3.5)

GTX 980Ti: 2816 cores * 1190 MHz * 2 = 6.7 TFPs (Maxwell 5.0)

GTX 1080Ti:3584 cores * 1500 MHz * 2 = 10.7 TFPs (Pascal 6.0)

Пиковая производительность GPU

- Операция FMAD(a, b, c) == a = b * c + a
- Пиковая производительность:

  CUDA_CORES * FREQ * FLOPS/CLOCK

GTX 1050: (128 * 5) cores * 1500 MHz * 2 = 2.0 TFLOPs

GTX 1060: (128 * 10) cores * 1700 MHz * 2 = 4.3 TFLOPs

GTX 1070: (128 * 15) cores * 1680 MHz * 2 = 6.4 TFLOPs

GTX 1080: (128 * 20) cores * 1610 MHz * 2 = 8.2 TFLOPs

GTX 1080Ti: (128 * 28) cores * 1500 MHz * 2 = 10.7 TFLOPs

Нативная реализация

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
    int bx = blockIdx.x;  // номер блока по x
    int by = blockIdx.y;  // номер блока по y
    int tx = threadIdx.x; // номер нити в блоке по x
    int ty = threadIdx.y; // номер нити в блоке по y

    dataT sum = 0;
    int ia = n * (BLOCK_SIZE * by + ty); // номер строки из A'
    int ib = BLOCK_SIZE * bx + tx;      // номер столбца из B'
    int ic = ia + ib;  // номер элемента из C'
    // вычисление элемента матрицы C
    for (int k = 0; k < n; k++)
        sum += a[ia + k] * b[ib + k * n];
    c[ic] = sum;
}
```

Реализация SMEM2

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
   int bx = blockIdx.x,     by = blockIdx.y;  // номер блока по x и y
   int tx = threadIdx.x,    ty = threadIdx.y; // номер нити в блоке по x и y
   int aBegin = n * BLOCK_SIZE * by;
   int aEnd = aBegin + n - 1,          bBegin = BLOCK_SIZE * bx;
   int  aStep = BLOCK_SIZE,            bStep = BLOCK_SIZE * n;

   dataT sum = 0;
   __shared__ dataT  as[BLOCK_SIZE][BLOCK_SIZE];
   __shared__ dataT  bs[BLOCK_SIZE][BLOCK_SIZE];

  for (int ia = aBegin, ib = bBegin;      ia <= aEnd;      ia += aStep, ib += bStep) {
      as[tx][ty] = a[ia + n * ty + tx];
      bs[tx][ty] = b[ib + n * ty + tx];
      __syncthreads();

      for (int k = 0; k < BLOCK_SIZE; k++)
            sum += as[k][ty] * bs[tx][k];
      __syncthreads();
   }
   c[aBegin + bBegin + ty * n + tx] = sum;
}
```

Реализация SMEM3

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
   int bx = blockIdx.x,     by = blockIdx.y;  // номер блока по x и y
   int tx = threadIdx.x,    ty = threadIdx.y; // номер нити в блоке по x и y
   int aBegin = n * BLOCK_SIZE * by;
   int aEnd = aBegin + n - 1,          bBegin = BLOCK_SIZE * bx;
   int  aStep = BLOCK_SIZE,            bStep = BLOCK_SIZE * n;

   dataT sum = 0;
   __shared__ dataT  as[BLOCK_SIZE][BLOCK_SIZE + 1];
   __shared__ dataT  bs[BLOCK_SIZE][BLOCK_SIZE + 1];

  for (int ia = aBegin, ib = bBegin;      ia <= aEnd;      ia += aStep, ib += bStep) {
      as[tx][ty] = a[ia + n * ty + tx];
      bs[tx][ty] = b[ib + n * ty + tx];
      __syncthreads();

      for (int k = 0; k < BLOCK_SIZE; k++)
            sum += as[k][ty] * bs[tx][k];
      __syncthreads();
   }
   c[aBegin + bBegin + ty * n + tx] = sum;
}
```

Устранение банк конфликтов

Реализация SMEM3 v2

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
   int bx = blockIdx.x,     by = blockIdx.y;  // номер блока по x и y
   int tx = threadIdx.x,    ty = threadIdx.y; // номер нити в блоке по x и y
   int aBegin = n * BLOCK_SIZE * by;
   int aEnd = aBegin + n - 1,          bBegin = BLOCK_SIZE * bx;
   int  aStep = BLOCK_SIZE,            bStep = BLOCK_SIZE * n;

   dataT sum = 0;
   __shared__ dataT  as[BLOCK_SIZE][BLOCK_SIZE];
   __shared__ dataT  bs[BLOCK_SIZE][BLOCK_SIZE];

  for (int ia = aBegin, ib = bBegin;      ia <= aEnd;      ia += aStep, ib += bStep) {
      as[ ty ][ tx ] = a[ia + n * ty + tx];
      bs[ ty ][ tx ] = b[ib + n * ty + tx];
      __syncthreads();

      for (int k = 0; k < BLOCK_SIZE; k++)
            sum += as[ ty ][ k ] * bs[ k ][ tx ];
      __syncthreads();
   }
   c[aBegin + bBegin + ty * n + tx] = sum;
}
```

Реализация SMEM6

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
   int bx = blockIdx.x,     by = blockIdx.y;  // номер блока по x и y
   int tx = threadIdx.x,    ty = threadIdx.y; // номер нити в блоке по x и y
   int aBegin = n * BLOCK_SIZE * by,      aEnd = aBegin + n - 1,        bBegin = BLOCK_SIZE * bx;
   int  aStep = BLOCK_SIZE,                bStep = BLOCK_SIZE * n,    offset = BLOCK_SIZE / ST;

   dataT sum1 = 0, sum2 = 0;
   __shared__ dataT  as[BLOCK_SIZE][BLOCK_SIZE];
   __shared__ dataT  bs[BLOCK_SIZE][BLOCK_SIZE];

  for (int ia = aBegin, ib = bBegin;      ia <= aEnd;       ia += aStep, ib += bStep) {
      as[ty][tx] = a[ia + n * ty + tx];          as[ty + offset][tx] = a[ia + n * (ty + offset) + tx];
      bs[ty][tx] = b[ib + n * ty + tx];          bs[ty + offset][tx] = b[ib + n * (ty + offset) + tx];
      __syncthreads();

      for (int k = 0; k < BLOCK_SIZE; k++) {
            sum1 += as[ty][k] * bs[k][tx];
            sum2 += as[ty + offset][k] * bs[k][tx];
       }
      __syncthreads();
    }
   c[aBegin + bBegin + ty * n + tx] = sum1;        c[aBegin + bBegin + (ty + offset) * n + tx] = sum2;
}
```

Реализация SMEM10

```
template <typename dataT>
__global__ void kernel(const dataT * __restrict a, const dataT * __restrict  b, int n, dataT *c)
{
   int bx = blockIdx.x,     by = blockIdx.y;  // номер блока по x и y
   int tx = threadIdx.x,    ty = threadIdx.y; // номер нити в блоке по x и y
   int aBegin = n * BLOCK_SIZE * by,      aEnd = aBegin + n - 1,        bBegin = BLOCK_SIZE * bx;
   int  aStep = BLOCK_SIZE,                    bStep = BLOCK_SIZE * n,    offset = BLOCK_SIZE / ST;

   const int numEl = 8;  dataT sum[numEl ] = {0, 0, 0, 0, 0, 0, 0, 0};
   __shared__ dataT  as[BLOCK_SIZE][BLOCK_SIZE];
   __shared__ dataT  bs[BLOCK_SIZE][BLOCK_SIZE];

  for (int ia = aBegin, ib = bBegin;      ia <= aEnd;      ia += aStep, ib += bStep) {
     for (int z = 0;  z < numEl; ++z) {
          as[ty + z * 4][tx] = a[ia + n * (ty + z * 4) + tx];
          bs[ty + z * 4][tx] = b[ib + n * (ty + z * 4) + tx];
     }
      __syncthreads();
      for (int k = 0; k < BLOCK_SIZE; k++)
           for (int z = 0;  z < numEl; ++z)
                sum[z] += as[ty + z * 4][k] * bs[k][tx];
      __syncthreads();
   }
   for (int z = 0; k < numEl; ++z)
        c[aBegin + bBegin + (ty + z * 4) * n + tx] = sum[z];
}
```

CUBLAS PERFORMANCE on GTX Titan (Kepler)