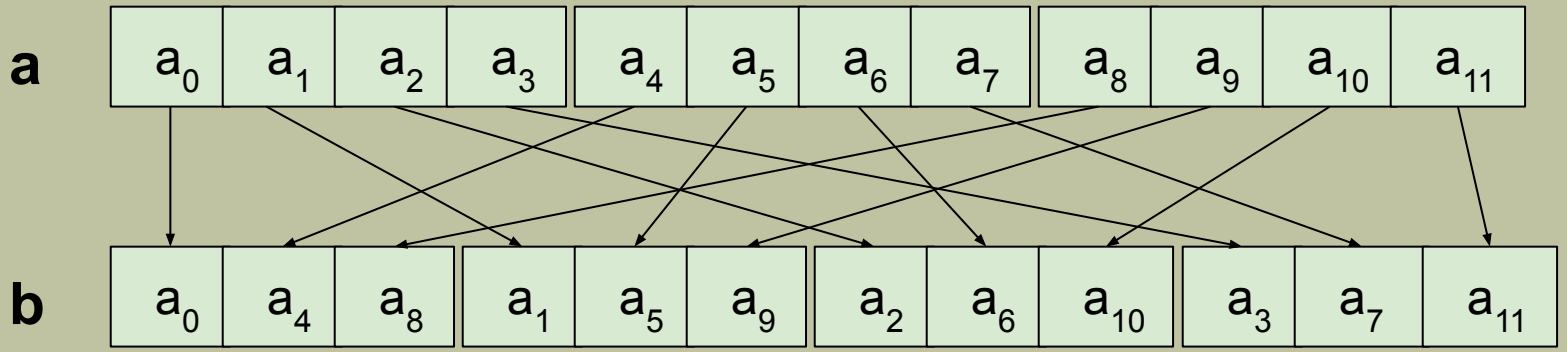
Лекция 6

- Совместный доступ к глобальной памяти (*coalescing*).
- Разделяемая память (*shared memory*).

Лабораторная 4

Провести копирование массива *a*, включающего *N* векторов длины *K* по образцу, приведенному на диаграмме:



a_0	a_1	a_2	a_3
a_4	a_5	a_6	a_7
a_8	a_9	a_{10}	a_{11}

Транспонирование

a_0	a_4	a_8
a_1	a_5	a_9
a_2	a_6	a_{10}
a_3	a_7	a_{11}

n=i/K, k=i%N - двумерная индексация, матрица.
a[k+n*K] => a[n+k*N]

```
__global__ void gCoalescingTest1(int* a, int N, int K){
 int i=threadIdx.x+blockIdx.x*blockDim.x;
 int n=i/K;       //i=k+n*K
 int k=i%K;
 a[k+n*K]=i;
}

__global__ void gCoalescingTest2(int* a, int N, int K){
 int i=threadIdx.x+blockIdx.x*blockDim.x;
 int n=i/K;       //i=k+n*K
 int k=i%K;
 a[n+k*N]=i;
}
```

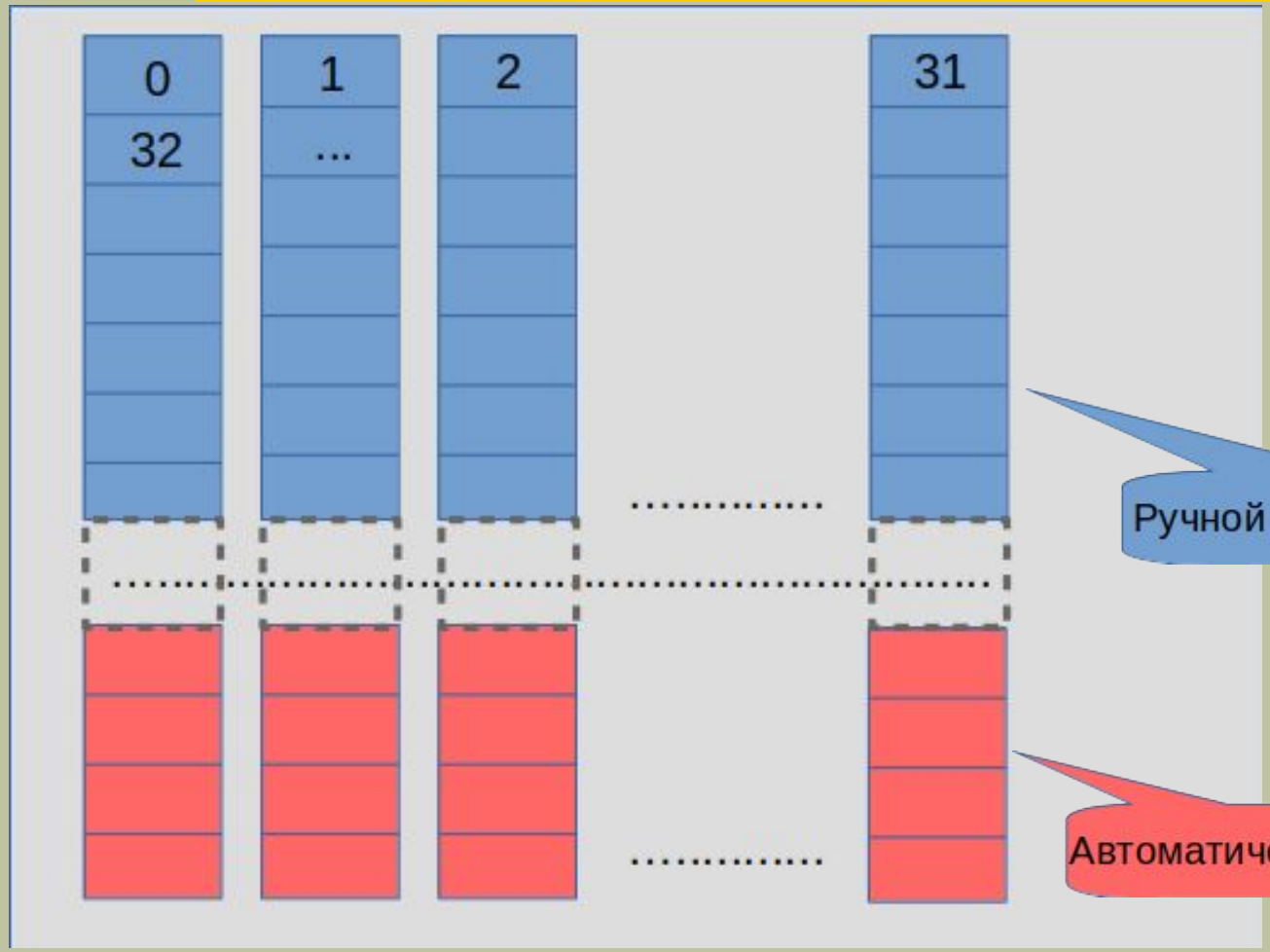
```
……
68.49%  108.42us    1  108.42us  108.42us  108.42us
                                             gCoalescingTest2(int*, int, int)
31.51%  49.888us    1  49.888us  49.888us  49.888us
                                             gCoalescingTest1(int*, int, int)
……
```

Запросы на чтение и запись к глобальной памяти нитями одного варпа (а *warp*) объединяются в транзакции, количество которых равно количеству необходимых для выполнения запросов блоков данных (*cache lines*) L1 кэша размером в 128 байт. COMPUTE CAPABILITIES 2.* !

```
__global__ void gTranspositionTest1(int* a, int* b, int N, int K){
 int k=threadIdx.x+blockIdx.x*blockDim.x;
 int n=threadIdx.y+blockIdx.y*blockDim.y;

 b[k+n*K]=a[n+k*N];
}

__global__ void gTranspositionTest2(int* a, int*b, int N, int K){
 int k=threadIdx.x+blockIdx.x*blockDim.x;
 int n=threadIdx.y+blockIdx.y*blockDim.y;

 b[n+k*N]=a[k+n*K];
}
```

35.79%	5.1200us	1	**5.1200us**	5.1200us	5.1200us	gTranspositionTest2(int*, int*, int, int)
27.96%	4.0000us	1	**4.0000us**	4.0000us	4.0000us	gTranspositionTest1(int*, int*, int, int)
16.11%	2.3040us	1	**2.3040us**	2.3040us	2.3040us	gCopy(int*, int*, int, int)

L1 кэш память



32 банка памяти

1 строка =32*4 байта = 128 байт

Размер кэша L1:
512 строки * 128 байт =64 K

Регулируемое соотношение:
16K/48K или 48K/16K

cudaDeviceSetCacheConfig(cudaFuncCachePreferL1);

```
__global__ void gTest(...){
    ……..
    return;
}

int main(){
    cudaFuncSetCacheConfig(gTest, cudaFuncCachePreferL1);
    ……..
    gTest<<<num_th, num_bl>>>(...);

    return 0;
}
```

Разделяемая память CUDA – память с низкой латентностью и высокой пропускной способностью.

Высокая пропускная способность обеспечивается параллельным выполнением запросов, благодаря разделению памяти на отдельные модули, банки памяти.

Если более одной нити варпа обращаются к одному и тому же банку, то происходит конфликт, который разрешается сериализацией выполнения запроса.

Разделяемая память выделяется (статически или динамически) только на устройстве. Область видимости – нити одного блока. Время жизни – время выполнения ядра.

Статическое выделение разделяемой памяти

```
#define N 3
#define M 512
__global__ void gTest1(){
__shared__ float s[N][M];
.....................................
}
```

Динамическое выделение разделяемой памяти

```
__global__ void gTest2(){
  extern __shared__ float s[];
  float* a=(float*)s;
  float* b=(float*)&s[512];
  float* c=(float*)&s[1024];
  ..........................................
}
```

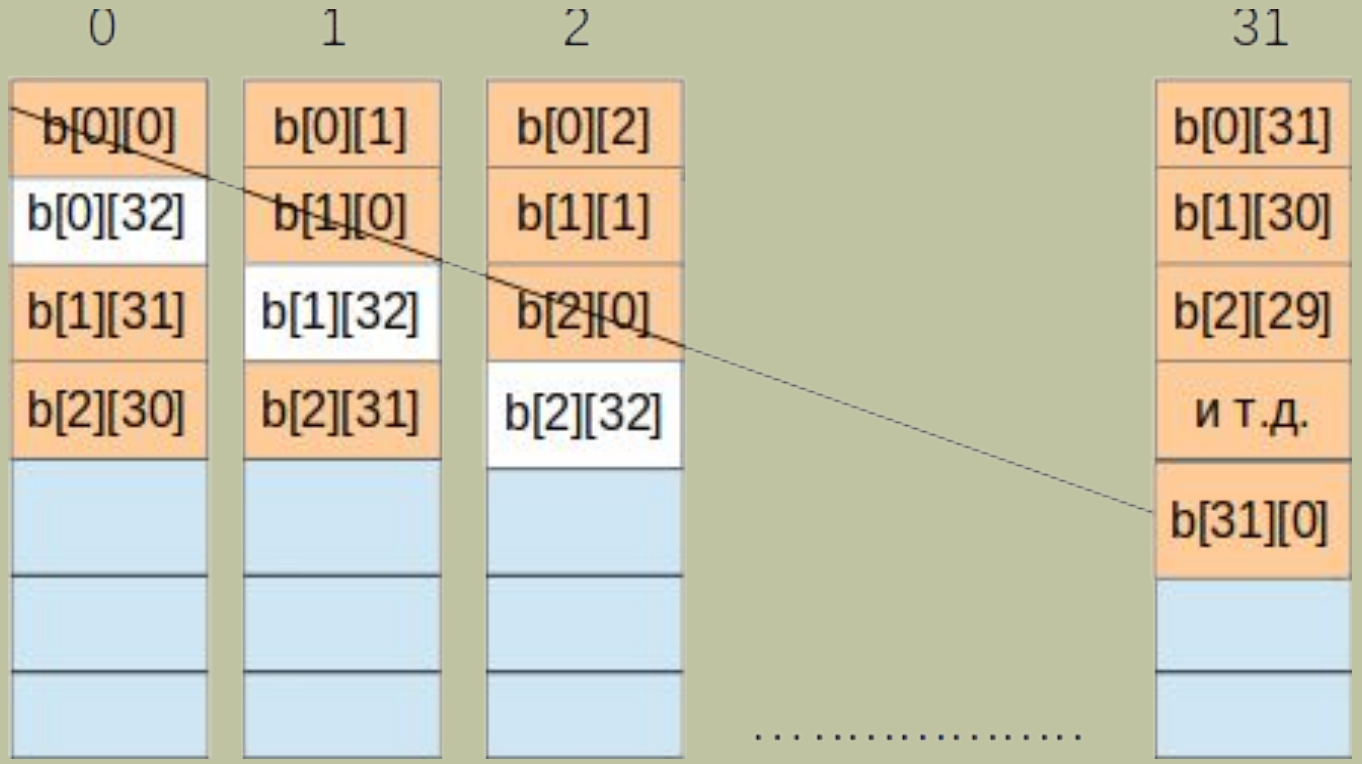
```
gTest2<<<100,32,N*M*sizeof(float)>>>();
```

3-й параметр – размер разделяемой памяти.

```
#define SH_DIM  32
__global__ void gTranspose(int* a, int* b){
  __shared__ int cache[SH_DIM][SH_DIM];
  int k=threadIdx.x+blockIdx.x*blockDim.x;
  int n=threadIdx.y+blockIdx.y*blockDim.y;
  int N=blockDim.x*gridDim.x;

  cache[threadIdx.y][threadIdx.x]=a[k+n*N];
  __syncthreads();

  k=threadIdx.x+blockIdx.y*blockDim.x;
  n=threadIdx.y+blockIdx.x*blockDim.y;

  b[k+n*N]=cache[threadIdx.x][threadIdx.y];
}
```

Размещение массива cache в разделяемой памяти (shared memory):

__shared__ float buffer[SH_DIM][SH_DIM+1];



```
__global__ void gTransposeWC(int* a, int* b){
   __shared__ int cache[SH_DIM][SH_DIM+1];
.....................................................................
}
```

Lab5> nvprof ./lab5d1g 128 128

```
.....................................................................................
20.15%  4.2880us   1  4.2880us  4.2880us  4.2880us  gTranspose(int*, int*)
12.48%  2.6560us   1  2.6560us  2.6560us  2.6560us  gTransposeWC(int*, int*)
.....................................................................................
```

0	16	32	48	64	80	96	112
2048	2064	2080	2096	2112	2128	2144	2160
4096	4112	4128	4144	4160	4176	4192	4208
6144	6160	6176	6192	6208	6224	6240	6256
8192	8208	8224	8240	8256	8272	8288	8304
10240	10256	10272	10288	10304	10320	10336	10352
12288	12304	12320	12336	12352	12368	12384	12400
14336	14352	14368	14384	14400	14416	14432	14448

b

0	2048	4096	6144	8192	10240	12288	14336
16	2064	4112	6160	8208	10256	12304	14352
32	2080	4128	6176	8224	10272	12320	14368
48	2096	4144	6192	8240	10288	12336	14384
64	2112	4160	6208	8256	10304	12352	14400
80	2128	4176	6224	8272	10320	12368	14416
96	2144	4192	6240	8288	10336	12384	14432
112	2160	4208	6256	8304	10352	12400	14448

Лабораторная 5

Реализуйте транспонирование матрицы размерностью N*K без использования разделяемой памяти, с разделяемой памятью без разрешения конфликта банков и с разрешением конфликта банков. Сравните время выполнения соответствующих ядер на GPU. Для всех трёх случаев определите эффективность использования разделяемой памяти с помощью метрик *nvprof* или *ncu.*