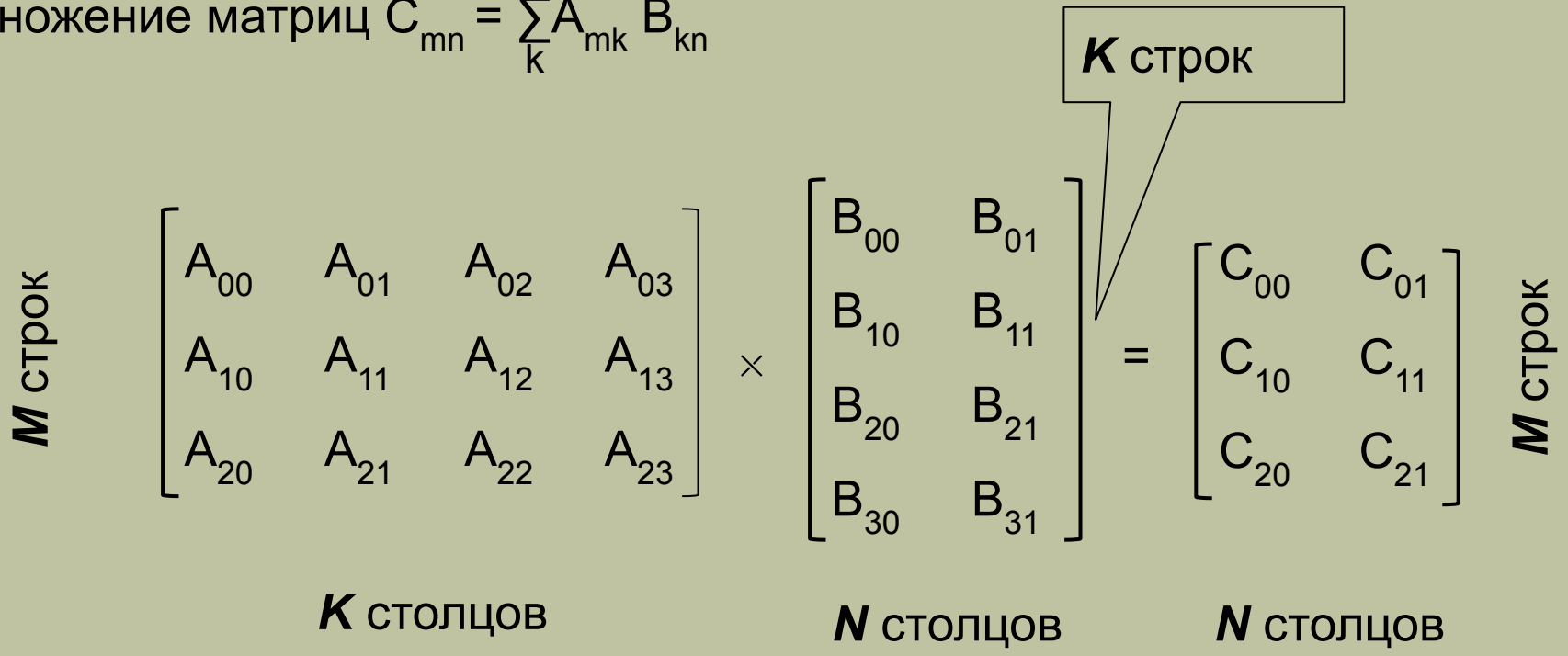
Лекция 8

Библиотека *cuBLAS*.

- Тензорные операции, произведение матриц.
- Реализация произведения матриц на основе CUDA API.
- Особенности использования библиотеки cuBLAS.
- Функции cublas<T>gemm().
- Тензорные процессоры.
- Вызовы cublas<T>gemm() с использованием тензорных процессоров.

Матричное произведение тензоров
Произведение матриц

Умножение матриц $C_{mn} = \sum_k A_{mk} B_{kn}$



```
#include <malloc.h>
#include <ctime>
#define M 1024
#define K 1024
#define N 1024

void hMultMats(double** A, double** B, double** C){
  double acc;
  for(int m=0; m<M;m++)
    for(int n=0; n<N;n++){
      acc=0.0;
      for(int k=0; k<K;k++)
        acc+=A[m][k]*B[k][n];
      C[m][n]=acc;
    }
}
```

```
void hInitMats(double** A, double** B){
  for(int m=0; m<M;m++)
     for(int k=0; k<K; k++)
        A[m][k]=(double)((k+m*K)*1.0E-5);

  for(int k=0; k<K;k++)
     for(int n=0; n<N;n++)
        B[k][n]=1.0;
}
```

```
Int main(){
 double **A, **B, **C;
  A=(double**)calloc(M, sizeof(double*));
  for(int m=0; m<M; m++)
    A[m]=(double*)calloc(K, sizeof(double));
……
hInitMats(A, B);
clock_t start=clock();
  hMultMats(A, B, C);
clock_t finish=clock();
……..
  hMatOut(A, M, K);
  hMatOut(B, K, N);
  hMatOut(C, M, N);
……
}
```

```
/Lecture8/Lab8> ./lab8cpu
0       0.00128 0.00256 0.00384 0.00512 0.0064  0.00768 0.00896
1.31072 1.312   1.31328 1.31456 1.31584 1.31712 1.3184  1.31968
……..
9.17504 9.17632 9.1776  9.17888 9.18016 9.18144 9.18272 9.184
////////////////////////////////////////////////////////////
1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1
……..
////////////////////////////////////////////////////////////
5.23776 5.23776 5.23776 5.23776 5.23776 5.23776 5.23776 5.23776
1347.42 1347.42 1347.42 1347.42 1347.42 1347.42 1347.42 1347.42
2689.59 2689.59 2689.59 2689.59 2689.59 2689.59 2689.59 2689.59
……..
9400.48 9400.48 9400.48 9400.48 9400.48 9400.48 9400.48 9400.48
Elapsed time: 4173.59 ms
```

```
/Lecture8/Lab8> g++  lab8cpu.cpp  -pg -o lab8cpu
/Lecture8/Lab8> ./lab8cpu
/Lecture8/Lab8> ls -ltr
итого 720
-rw-r--r-- 1 malkov users   1528 Mar 16 18:27 lab8cpu.cpp
-rwxr-xr-x 1 malkov users  20496 Mar 16 19:22 lab8cpu
-rw-r--r-- 1 malkov users   1023 Mar 16 19:22 gmon.out
/Lecture8/Lab8> gprof lab8cpu gmon.out > lab8cpu.prof
```

```
/Lecture8/Lab8> vim lab8cpu.prof
Flat profile:

Each sample counts as 0.01 seconds.
 %   cumulative  self    self    total
time      secs      secs  calls   s/call    s/call            name
100.40  3.98     3.98      1       3.98     3.98  hMultMats(double**, double**, double**)
 0.25     3.99      0.01    1        0.01     0.01  hInitMats(double**, double**)
.....
```

```
#include <malloc.h>
#define M 1024
#define K 1024
#define N 1024
#define BLOCK_DIM 32

__global__ void gMultMats(float* A, float* B, float* C){
  int n=threadIdx.x + blockIdx.x*blockDim.x;
  int m=threadIdx.y + blockIdx.y*blockDim.y;

  float acc=0.0;
  for(int k=0; k<K; k++)
    acc+=A[k+m*K]*B[n+k*N];
  C[n+m*N]=acc;
}
```

```
__global__ void gInit(float* D, int s){
  int j=threadIdx.x + blockIdx.x*blockDim.x;
  int i=threadIdx.y + blockIdx.y*blockDim.y;
  int J=blockDim.x*gridDim.x;

  D[j+i*J]=s*(float)((j+i*J)*1.0E-5)+(1-s)*1.0f;
}
```

```
int main(){
  float *A, *B, *C;
  cudaMalloc((void**)&A, M*K*sizeof(float));
……
  gInit<<<dim3(K/32, M/32),dim3(32,32)>>>(A,1);
  cudaDeviceSynchronize();
  gInit<<<dim3(N/32, K/32),dim3(32,32)>>>(B,0);
  cudaDeviceSynchronize();
  cudaMemset(C, 0, M*N*sizeof(REAL));

  gMultMats<<<dim3(N/BLOCK_DIM, M/BLOCK_DIM),
                    dim3(BLOCK_DIM, BLOCK_DIM)>>>(A,B,C);
  cudaDeviceSynchronize();

  hMatOut(A, M, K);
……..
```

```
/Lecture8/Lab8> ./lab8gpu
0       0.00128 0.00256 0.00384 0.00512 0.0064  0.00768 0.00896
1.31072 1.312   1.31328 1.31456 1.31584 1.31712 1.3184  1.31968
……
9.17504 9.17632 9.1776  9.17888 9.18016 9.18144 9.18272 9.184
////////////////////////////////////////////////////////////
1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1
……..
////////////////////////////////////////////////////////////
5.23776 5.23776 5.23776 5.23776 5.23776 5.23776 5.23776 5.23776
1347.42 1347.42 1347.42 1347.42 1347.42 1347.42 1347.42 1347.42
2689.59 2689.59 2689.59 2689.59 2689.59 2689.59 2689.59 2689.59
……..
9400.48 9400.48 9400.48 9400.48 9400.48 9400.48 9400.48 9400.48
```

```
/Lecture8/Lab8> nvprof ./lab8gpu

    Type
  Time(%)  Time     Calls   Avg         Min         Max              Name
GPU activities:
63.25%  3.9720ms   1  3.9720ms  3.9720ms  3.9720ms  gMultMats(float*, float*, float*)
 1.90%   119.46us  2  59.728us  59.648us  59.808us  gInit(float*, int)
```

3.98s*1000/3.9720=**1002.0141**

Ускорение в тысячу раз!😲

Оптимизация “сырого” кода

```
__global__ void gMultMats(REAL* A, REAL* B, REAL* C){
  int nb = blockIdx.x * blockDim.x;
  int mb = blockIdx.y * blockDim.y;
  int nt = threadIdx.x;
  int mt = threadIdx.y;

  float acc = 0.f;
  __shared__ float sA[BLOCK_DIM][BLOCK_DIM];
  __shared__ float sB[BLOCK_DIM][BLOCK_DIM];

  for (int k = 0; k < K; k += BLOCK_DIM){
    sA[mt][nt] = A[(nt + k) + (mb + mt) * K];
    sB[mt][nt] = B[(nb + nt) + k * N];
    __syncthreads();
```

```
    for (int l = 0; l < BLOCK_DIM; l++)
      acc += sA[mt][l] * sB[l][nt];
  }
  C[(mb + mt) * N + (nb + nt)] = acc;
}
```

```
/Lecture8/Lab8opt> nvprof ./lab8o2
58.69%  2.9319ms   1  2.9319ms  2.9319ms  2.9319ms  gMultMats(float*, float*, float*)
2.39%   119.20us   2  59.599us  59.456us  59.743us  gInit(float*, int)
……
```

3.9720/2.9319=1.3548

Ускорение по отношению к последовательному алгоритму равно 1357.5287!

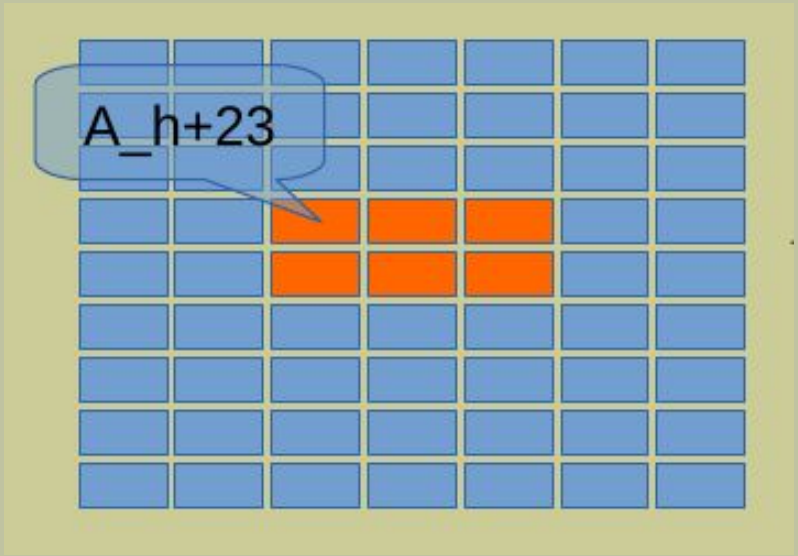
Библиотека cuBLAS (***B**asic **L**inear **A**lgebra **S**ubroutines*)

- Хранение по столбцам (column-major storage), совместимость с Фортраном, для копирования и инициализации матриц следует использовать специальный API.
- Линейная индексация массивов;
- Имена функций образуются по схеме: cublas<T><function>. Например, ***cublasSgemm***, тип данных *float*, *generic/general matrix-matrix* умножение плюс сложение: $C=\alpha AB+\beta C$

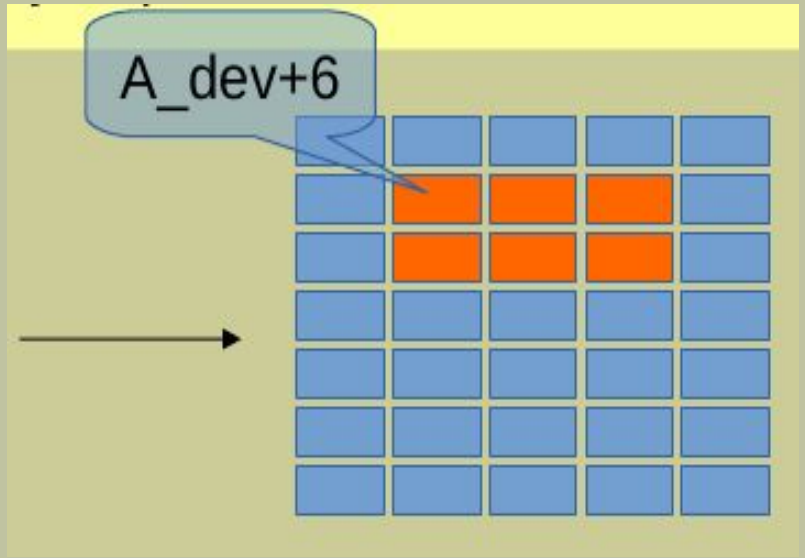
Документация: ***https://docs.nvidia.com/cuda/cublas/#***

```
int main(){
// Инициализация библиотеки CUBLAS
 cublasHandle_t cublas_handle;
 cublasCreate(&cublas_handle);
……..
//Копирование матрицы с числом строк num_rows и числом
столбцов //num_cols с хоста на устройство
  cublasSetMatrix(num_rows, num_cols, elem_size, A_h, ldah, A_dev, ldad);
  <вызов функции cuBLAS API>
  cublasGetMatrix(num_rows, num_cols, elem_size, A_dev, num_rows, A_h,
                                                          num_rows);
……..
  cublasDestroy(cublas_handle);
}
```

leading dimension



leading dimension равно 9



leading dimension равно 7

```
void hMultMatsBlas(REAL* A, REAL* B, REAL* C){
  cublasHandle_t cublas_handle;
  cublasCreate(&cublas_handle);

  const float alpha=1.0;
  const float beta=0.0;
//  cublasSetMathMode(cublas_handle, CUBLAS_TENSOR_OP_MATH);
  cublasSgemm(cublas_handle, CUBLAS_OP_T, CUBLAS_OP_N,
                        M, N, K,
                        &alpha,
                        A, M,
                        B, K,
                        &beta,
                        C, M);
  cublasDestroy(cublas_handle);
}
```

```
void hMatOutBlas(REAL* D, int I, int J){
  REAL* Dh=(REAL*)calloc(I*J, sizeof(REAL));
  cublasGetMatrix(I, J, sizeof(REAL), D, I, Dh, I);
……
}
```

“NVIDIA GeForce RTX 2060”

~/Lecture8/Lab8blas> nvprof ./lab8blas

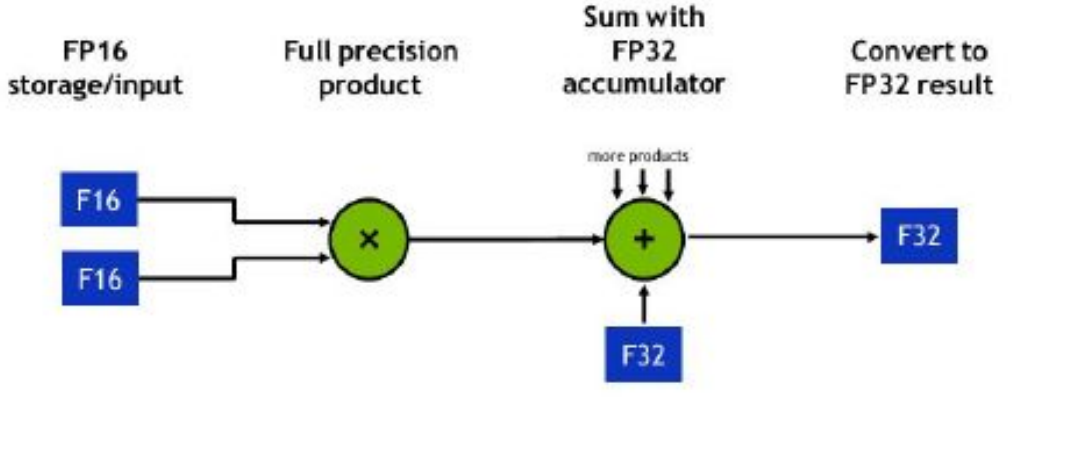
16.51% 516.96us 1 **516.96us** 516.96us 516.96us
volta_sgemm_128x64_tn
3.81% 119.39us 2 **59.695us** 59.552us 59.839us
gInit(float*, int)

“GeForce GTX 1050”

~/WORKSHOP/PGP-2023> nvprof ./lab8blas
…….
41.49% 1.3640ms 1 **1.3640ms** 1.3640ms 1.3640ms
sgemm_128x128x8_NT_vec
9.28% 305.00us 2 **152.50us** 152.36us 152.65us
gInit(float*, int)

Нейронные процессоры:

- специализированные микросхемы, “заточенные” на ускорение алгоритмов машинного обучения;
- аппаратная реализация GEMM ($C=\alpha AB+\beta C$);
- вычисления со смешанной точностью (FP16, FP32, FP64);
- перемножение матриц размерности 4x4, 8x8, 16x16 за один такт;



GETTING STARTED WITH TENSOR CORES IN HPC
Vishal Mehta, NVIDIA
Super Computing, 2019

Google TPU,
Huawei Ascend 310 / Ascend 910,
.....
NVIDIA Tensor Cores.

V100: 64 GEMM за такт, архитектура Volta (Nvidia Tesla V100), sm_70;
TU100-104: архитектура Turing (Geforce RTX 20*), sm_75;
A100: 256 GEMM за такт, архитектура Ampere (Geforce RTX 30*), sm_80.

WMMA, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#wmma;
cuBLAS, https://docs.nvidia.com/cuda/cublas/;
CUTLASS, https://nvidia.github.io/cutlass/;
cuTENSOR, https://docs.nvidia.com/cuda/cutensor/;
cuDNN, https://developer.nvidia.com/cudnn;
TensorFlow, https://www.tensorflow.org/,
https://docs.nvidia.com/deeplearning/frameworks/tensorflow-user-guide/;
PyTorch, https://pytorch.org/.

1. CUTENSOR A CUDA Library for High-Performance Tensor Primitives, Paul Springer, November 20th 2019.
2. GETTING STARTED WITH TENSOR CORES IN HPC, Vishal Mehta, NVIDIASuper Computing, 2019
3. CUTLASS:CUDA TEMPLATELIBRARY FOR DENSE LINEAR ALGEBRA AT ALL LEVELS AND SCALES, Jeng Bai-Cheng(Ryan), 21 Nov.
4. VOLTA TENSOR CORE TRAINING ORNL, August 2019.
5. Stefano Markidis et al. NVIDIA Tensor Core Programmability, Performance & Precision, arXiv:1803.04014 [cs.DC], (2018). https://doi.org/10.48550/arXiv.1803.04014.

Тензорные ядра не активны.

~/Lecture8/Lab8blas> ncu --replay-mode application --metrics **sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active** lab8blas

volta_sgemm_128x64_tn, 2023-Mar-20 13:56:45, Context 1, Stream 7

Section: Command line profiler metrics

sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active	%	0

~/Lecture8/Lab8blas> nvprof ./lab8blas

……

11.63% 308.32us 1 **308.32us** 308.32us 308.32us

volta_s884gemm_128x128_ldg8_f2f_tn

4.44% 117.79us 2 58.895us 58.784us 59.007us

gInit(float*, int)

……

Тензорные ядра активны.

~/Lecture8/Lab8blas> ncu --replay-mode application --metrics

sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active lab8blas

volta_s884gemm_128x128_ldg8_f2f_tn, 2023-Mar-20 12:29:27, Context 1, Stream 7

Section: Command line profiler metrics

---------- ----------

sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active % **37.90**

---------- ----------

Ускорение по отношению к оптимизированному алгоритму на основе *CUDA API* на *GPU* 9.51.

Потеря точности.

5.23777 5.23777 ……..5.23777
1347.41 1347.41……1347.41
2689.56 2689.56 ……2689.56
……..……
9400.73 9400.73 ……..9400.73

```
void hMultMatsBlas(half* A, half* B, half* C){
 ……
  const half alpha=1.0;
  const half beta=0.0;

  cublasSetMathMode(cublas_handle, CUBLAS_TENSOR_OP_MATH);
  cublasHgemm(cublas_handle, CUBLAS_OP_T, CUBLAS_OP_N,
                        M, N, K,
                        &alpha,
                        A, M,
                        B, K,
                        &beta,
                        C, M);
  cublasDestroy(cublas_handle);
}
```

```
__global__ void gInit(REAL* D, int s){
……
  D[j+i*J]=__float2half(s*(float)((j+i*J)*1.0E-5)+(1-s)*1.0f);
}
```

```
int main(){
  half *A, *B, *C;
  cudaMalloc((void**)&A, M*K*sizeof(half));
……
  hMultMatsBlas(A,B,C);
……
}
```

```
……
fprintf(stdout, "%g\t", __half2float(Dh[i+j*l]));
……
```

```
/Lecture8/Lab8blas> nvprof ./lab8blasH
13.92%  118.59us   2  59.295us  59.231us  59.359us
                              gInit(__half*, int)
11.68%  99.519us   1  99.519us  99.519us  99.519us
                              turing_h1688gemm_128x128_ldg8_stages_32x1_tn
```

```
~/Lecture8/Lab8blas>  ncu --replay-mode application --metrics
sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active
lab8blas
turing_h1688gemm_128x128_ldg8_stages_32x1_tn, 2023-Mar-20 14:42:07, Context
1, Stream 7
   Section: Command line profiler metrics
  ---------------------------------------------------------------------------------------------
sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active    %      62.27
  ---------------------------------------------------------------------------------------------
```

Тензорные ядра активны.

Потеря точности.

5.23828 5.23828…… 5.23828
1344 1344…….. .1344
2688 2688…… ..2688
……..
9392 9392……..9392

Ускорение по отношению к оптимизированному алгоритму на основе CUDA API на GPU **29.46.**

Ускорение по отношению к алгоритму на CPU **39992.36.**

В сорок тысяч раз!

ЗАДАНИЕ.

Реализовать вычисление произведения матриц на GPU, используя CUDA API (“сырой код”) и, отдельно, используя библиотеку *cuBLAS*. Сравнить время выполнения программ при различной размерности матриц.