ЛАБОРАТОРИЙН АЖИЛ №1
Параллел програмчлалын хэрэгслүүд

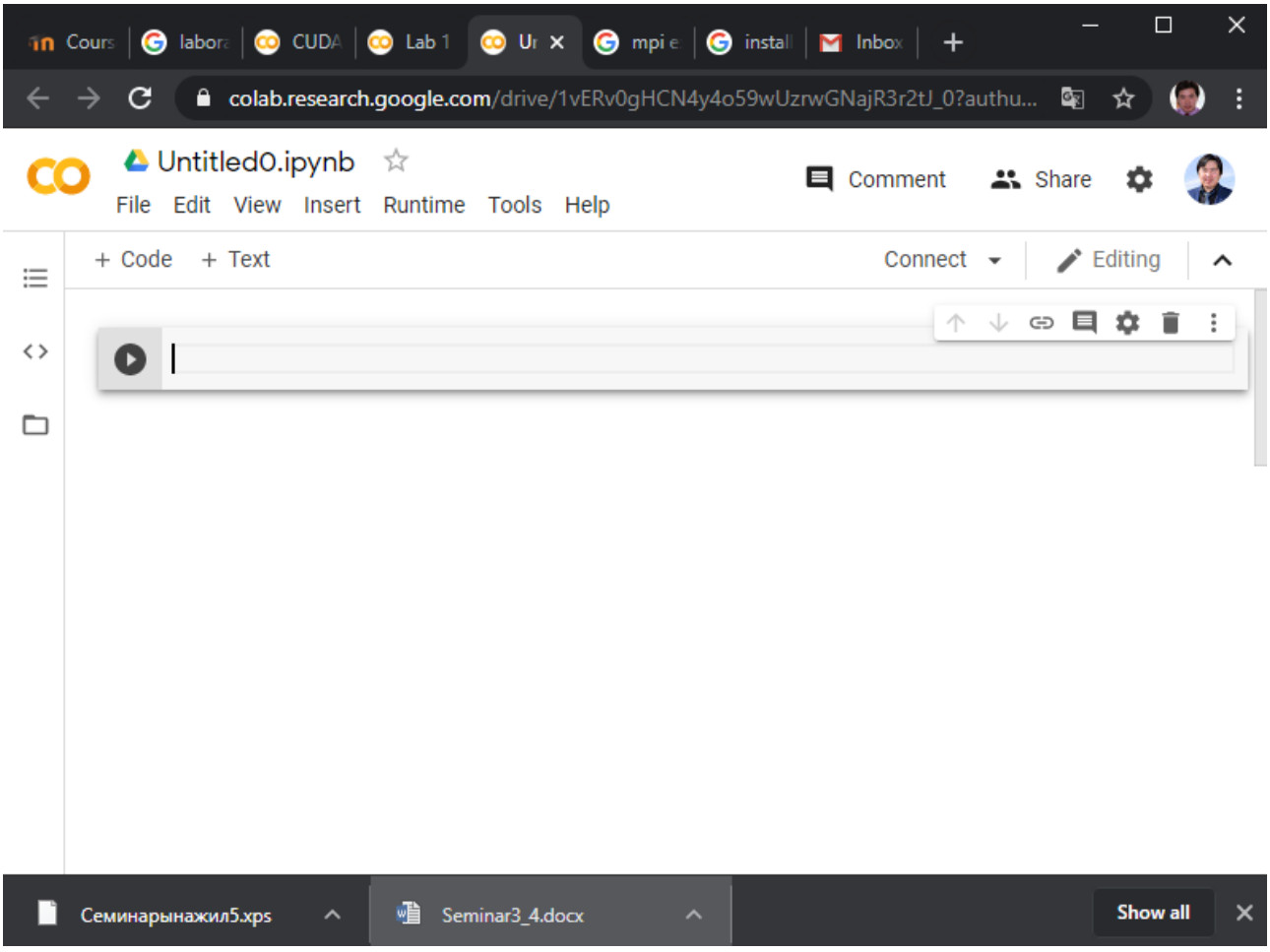
Зорилго: Лабораторийн хичээлээр хэрэглэх програм хангамж, хэрэгслийг ашиглаж сурах

Бид энэ хичээлээр C++11 or C++14, OpenMP, CUDA, MPI API-уудыг ашиглаж сурна. Хичээлийн явцад эдгээрийг ажиллуулж туршиж болох **Google Colaboratory**-г ашиглана. Мөн өөрсдийн хүссэн програмчлалын хэл, IDE-г ашиглах боломжтой.
Жич: CUDA ашиглах үед компьютер маань заавал NVIDIA-ийн график кардтай байхыг шаардана.

Google Colaboratory-н линк: https://colab.research.google.com/

1. **Colaboratory ашиглах тухай.**

Colaboratory (colab) системд Google Account ашиглана нэвтрэнэ. +Code, +Text товчийг ашиглаж блок үүсгэнэ. Кодын блокын өмнөх сумыг дарж блокыг ажиллуулна. Текстийн блок ашиглан тайлбарыг бичнэ. Ингэснээр код бичих явцаа бичиг баримттай болчихно гэсэн үг. Үүнийг дараа нь шууд Share хийж бусадтай хуваалцах, багш руу явуулж үр дүнгээ хянуулах гэх мэт өөр зүйлүүдэд ашиглаж болно😉. Багш руу явуулахдаа текст блокыг сайн ашиглана. Өөрөөр хэлбэл тайлбар сайтай, энэ лабораторын ажлыг удирдамж шиг бичиг баримт бэлтгэнэ гэсэн үг.



2. Програмын код бичих

Colab дээр шинэ notebook нээж дараах блокыг үүсгэн програмын кодыг дараах байдлаар бичнэ. Шууд програмын кодыг бичихээс гадна өөр сайн IDE дээр биччихээд хуулж тавьж болно. Доорх хэсэгт програмын эх кодын файлыг `%%writefile` команд ашиглан үүсгэн хадгалж байна.

```
%%writefile test.cpp

#include <iostream>
using namespace std;
//Main function
int main(int Argc,char* Args[]){
    cout << "Hello world!";
  return 0;
}
```

3. C++, OpenMP, MPI, AVX2 регистр ашигласан програмын кодыг ажиллуулах

Өмнөх байдраар програмын кодыг бичиж тухайн notebook дээрээ дахин блокыг нээж прогам ажиллуулах зааврыг бичнэ. Энд `%%script bash`-г ашиглаж системийн бүлэг командыг ажиллуулна.

a. C++

```
%%script bash
g++ test.cpp -o test
./test
```

b. MPI

```
%%script bash
mpiCC -o test test.c
mpirun --allow-run-as-root -np 4 test
```

c. OpenMP

```
%%script bash
g++ test.c -o test -fopenmp
./test
```

d. AVX2 регистр

```
%%script bash
g++ -mavx2 -std=c++17 shuffle.cpp -o shuffle
./shuffle
```

4. CUDA програмын кодыг ажиллуулах

Эхлээд програм ажиллуулах орчныг бэлтгэе. nvcc4jupyter санг суулгана. Дараах командыг шинэ блок дотор бичээд ажиллуулна.

```
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
```

Дараа нь суулгасан сангаа доорх командыг ажиллуулан ачаална.

```
%load_ext nvcc_plugin
```

Одоо програмын кодоо бичихдээ шинэ блок үүсгээд файлд хадгалах командын оронд %%cu кодоор эхлүүлэн бичнэ. Colab notebook чинь өөрөө хадгалагдаж байгаа учир файл хадгаласангүй гэж санаа зоволтгүй. Программаа биччихээд энэ блокоо ажиллуулна.

```
%%cu

//Эндээс эхлэн програмыг кодыг бичнэ.
...
```

Лабораторийн ажлын даалгавар.

Дараах кодуудыг Colaboratory дээр ажиллуулна. Алгоритм хэрхэн ажиллаж байгааг текст блок ашиглан тайлбарлан бичээрэй.

Жич: *Даалгаврыг CoLab дээр гүйцэтгээд Moodle системээр Shareable link -ийг явуулна. Даалгаврыг гүйцэтгэхдээ үндсэн сурах бичгийг ашиглаарай.*

Нэр. OpenMP program to print Hello World using C language

```
// OpenMP header
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  // Beginning of parallel region
  #pragma omp parallel
  {
    printf("Hello World... from thread = %d\n",
      omp_get_thread_num());
  }
  // Ending of parallel region
}
```

Хоёр. MPI program to print Hello World using C language

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Гурав. AVX2 program to shuffe Matrix

```
%%writefile shuffle.cpp


#include <immintrin.h>
#include <stdio.h>

int main(int argc, char const *argv[]) {

    // Single-precision shuffle with two 256-bit vectors and 8-
bit control value
    __m256 float_256_vec_0 = _mm256_set_ps(1, 2, 3, 4, 5, 6, 7, 8);
    __m256 float_256_vec_1 = _mm256_set_ps(9, 10, 11, 12, 13 ,14, 15, 16);

    __m256 float_256_result = _mm256_shuffle_ps(float_256_vec_0, float_256_v
ec_1, 0b00010111);
        // The result should be: (max -> least) 12, 11, 3, 1 ,16, 15, 7, 5
        //                       (least -> max) 5, 7, 15, 16, 1, 3, 11, 12

    float* flo = (float*) &float_256_result;
    printf("float:\t\t%f, %f, %f, %f, %f, %f, %f, %f\n", flo[0], flo[1], flo
[2], flo[3], flo[4], flo[5], flo[6], flo[7]);

    // Double-precision shuffle with two 256-bit vectors and 4-
bit control value
    __m256d double_256_vec_0 = _mm256_set_pd(1, 2, 3, 4);
    __m256d double_256_vec_1 = _mm256_set_pd(5, 6, 7, 8);

    __m256d double_256_result = _mm256_shuffle_pd(double_256_vec_0, double_2
56_vec_1, 0b0110);
        // The result should be: (max -> least) 6, 1, 7, 4
        //                       (least -> max) 4, 7, 1, 6

    double* dou = (double*) &double_256_result;
    printf("double:\t\t%lf, %lf, %lf, %lf\n", dou[0], dou[1], dou[2], dou[3]
);

    // 32-bit integer shuffle with a 256-bit vector and 8-bit control value
    __m256i epi32_256_vec_0 = _mm256_set_epi32(1, 2, 3, 4, 5, 6, 7, 8);

    __m256i epi32_256_result = _mm256_shuffle_epi32(epi32_256_vec_0, 0b00010
111);
        // The result should be: (max -> least) 4, 3, 3, 1, 8, 7, 7, 5
        //                       (least -> max) 5, 7, 7, 8, 1, 3, 3, 4
```

```
    int* i = (int*) &epi32_256_result;
    printf("int:\t\t%d, %d, %d, %d, %d, %d, %d, %d\n", i[0], i[1], i[2], i[3
], i[4], i[5], i[6], i[7]);

    // 8-bit integer shuffle with 256-bit vector and another 256-
bit control vector
    __m256i epi8_256_vec_0 = _mm256_set_epi8(1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6
, 6, 7, 7, 8, 8,
                                             9, 9, 10, 10, 11, 11, 12, 12, 1
3, 13, 14, 14, 15, 15, 16, 16);

    __m256i epi8_256_control_vec = _mm256_set_epi8(0, 0, 1, -1, 2, -2, 3, -
3, 4, -4, 5, -5, 6, -6, 7, -7,
                                                   8, -8, 9, -9, 10, -
10, 11, -11, 12, -12, 13, -13, 14, -14, 15, -15);

    __m256i epi8_256_result = _mm256_shuffle_epi8(epi8_256_vec_0, epi8_256_c
ontrol_vec);
    // The result should be: (max -
> least) 8, 8, 8, 0, 7, 0, 7, 0, 6, 0, 6, 0, 5, 0, 5, 0,
    //                                        12, 0, 12, 0, 11, 0, 11, 0, 10,
0, 10, 0, 9, 0, 9, 0
    //                        (least -
> max) 0, 9, 0, 9, 0, 10, 0, 10, 0, 11, 0, 11, 0, 12, 0, 12,
    //                                        0, 5, 0, 5, 0, 6, 0, 6, 0, 7, 0,
 7, 0, 8, 8, 8

    char* c = (char*) &epi8_256_result;
    printf("char:\t\t%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
 %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d\n",
c[0], c[1], c[2], c[3], c[4], c[5], c[6], c[7], c[8], c[9], c[10], c[11], c[
12], c[13], c[14], c[15], c[16], c[17], c[18], c[19], c[20], c[21], c[22], c
[23], c[24], c[25], c[26], c[27], c[28], c[29], c[30], c[31]);

    return 0;

}
```

Дөрөв. CUDA program to add Array

```
%%cu

#include <stdio.h>
#define N 1000
__global__
void add(int *a, int *b) {
    int i = blockIdx.x;
    if (i<N) {
        b[i] = 2*a[i];
    }
}

int main() {
    int ha[N], hb[N];
    int *da, *db;
    cudaMalloc((void **)&da, N*sizeof(int));
    cudaMalloc((void **)&db, N*sizeof(int));
    for (int i = 0; i<N; ++i) {
        ha[i] = i;
    }
    cudaMemcpy(da, ha, N*sizeof(int), cudaMemcpyHostToDevice);
    add<<<N, 1>>>(da, db);
    cudaMemcpy(hb, db, N*sizeof(int), cudaMemcpyDeviceToHost);
    for (int i = 0; i<N; ++i) {
        printf("%d\n", hb[i]);
    }
    cudaFree(da);
    cudaFree(db);

    return 0;
}
```