

GPU naudojimas Google colab aplinkoje

1. Prisijungti su Google paskyra <https://colab.research.google.com>.
2. Rinktis „Nauja užrašinė“.
3. Atsidariusiame lange iš meniu pasirinkti Vykdymo aplinka -> Keisti vykdymo aplinkos tipą.
4. Pasirinkti vieną iš leidžiamų GPU, pvz., T4 GPU.
5. Suinstaliuoti nvcc plėtinį su komanda
`!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git`
6. Įjungti plėtinį su komanda `%load_ext nvcc_plugin`
7. Toliau galima dėti savo kodą pirmoje eilutėje prirašius `%%cu`

Galimas ekrano vaizdas norint paleisti vieną iš BitBucket pavyzdinių programų:

```
[6] !pip install git+https://github.com/andreinechaev/nvcc4jupyter.git

Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-jylh6m3u
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-jylh6m3u
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 0a71d56e5dce3ff1f0dd2c47c29367629262f527
  Preparing metadata (setup.py) ... done

%load_ext nvcc_plugin

The nvcc plugin extension is already loaded. To reload it, use:
  %reload_ext nvcc_plugin

[ ]

%%cu
#include "cuda_runtime.h"
#include <iostream>
#include <iomanip>
#include <random>
#include <algorithm>

using namespace std;

const size_t ARRAY_SIZE = 100;
const size_t INNER_ARRAY_SIZE = 50;
const size_t FULL_ARRAY_SIZE = ARRAY_SIZE * INNER_ARRAY_SIZE;

void generate_random_array(int *array, size_t size);
void print_matrix(int* matrix);
__global__ void get_doubled_matrix(const int* original, int* result);

// a program that demonstrates how to use blocks and threads to multiply all elements in a matrix by 2.
int main() {
    int* flat_matrix = new int[FULL_ARRAY_SIZE]; // matrix will be held in a flat array
    // pointers that will be assigned by cuda memory allocation
    int* flat_matrix_device;
    int* doubled_matrix;
    // result will be held in this array
    int* doubled_matrix_host = new int[FULL_ARRAY_SIZE];
    // fill array with random values
    generate_random_array(flat_matrix, FULL_ARRAY_SIZE);
    // output what we generated
    print_matrix(flat_matrix);

    // make memory on GPU to fit our original data and write the address of that memory to flat_matrix_device pointer
    cudaMalloc((void**)&flat_matrix_device, FULL_ARRAY_SIZE * sizeof(int));
    // copy to flat_matrix_device from flat_matrix (from CPU to GPU memory)
    cudaMemcpy(flat_matrix_device, flat_matrix, FULL_ARRAY_SIZE * sizeof(int), cudaMemcpyHostToDevice);
    // make memory on GPU for our result
    cudaMalloc((void**)&doubled_matrix, FULL_ARRAY_SIZE * sizeof(int));

    // launch ARRAY_SIZE blocks on GPU, each block containing INNER_ARRAY_SIZE of threads.
    get_doubled_matrix<<ARRAY_SIZE, INNER_ARRAY_SIZE>>>(flat_matrix_device, doubled_matrix);
    // wait for CUDA operations to finish
    cudaDeviceSynchronize();

    // retrieve our result from GPU
    cudaMemcpy(doubled_matrix_host, doubled_matrix, FULL_ARRAY_SIZE * sizeof(int), cudaMemcpyDeviceToHost);
}
```