```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

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

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to
/tmp/pip-req-build-cz0tn_qr
  Running command git clone --filter=blob:none --quiet
https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-
cz0tn_qr
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit
aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... e=NVCCPlugin-0.0.2-py3-none-
any.whl size=4287
sha256=194a071cdce43ec0da1bcd2a72836d9047a5ac7365efa00d8deb8b1bd157a2d
7
  Stored in directory:
/tmp/pip-ephem-wheel-cache-00_2ab5x/wheels/a8/b9/18/23f8ef71ceb0f63297
dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

```
%load_ext nvcc_plugin
```

```
created output directory at /content/src
Out bin /content/result.out
```

VECTOR ADDITION

```
%%cu

#include <stdio.h>

// CUDA kernel for vector addition
__global__ void vectorAdd(int* a, int* b, int* c, int size)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if (tid < size) {
        c[tid] = a[tid] + b[tid];
    }
}
```

```
int main()
{
    int size = 100;  // Size of the vectors
    int* a, * b, * c;    // Host vectors
    int* dev_a, * dev_b, * dev_c;  // Device vectors

    // Allocate memory for host vectors
    a = (int*)malloc(size * sizeof(int));
    b = (int*)malloc(size * sizeof(int));
    c = (int*)malloc(size * sizeof(int));

    // Initialize host vectors
    for (int i = 0; i < size; i++) {
        a[i] = i;
        b[i] = 2 * i;
    }

    // Allocate memory on the device for device vectors
    cudaMalloc((void**)&dev_a, size * sizeof(int));
    cudaMalloc((void**)&dev_b, size * sizeof(int));
    cudaMalloc((void**)&dev_c, size * sizeof(int));

    // Copy host vectors to device
    cudaMemcpy(dev_a, a, size * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, size * sizeof(int), cudaMemcpyHostToDevice);

    // Launch kernel for vector addition
    int blockSize = 256;
    int gridSize = (size + blockSize - 1) / blockSize;
    vectorAdd<<<gridSize, blockSize>>>(dev_a, dev_b, dev_c, size);

    // Copy result from device to host
    cudaMemcpy(c, dev_c, size * sizeof(int), cudaMemcpyDeviceToHost);

    // Print result
    for (int i = 0; i < size; i++) {
        printf("%d + %d = %d\n", a[i], b[i], c[i]);
    }

    // Free device memory
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);

    // Free host memory
    free(a);
    free(b);
    free(c);
    return 0;
}
```

```
0 + 0 = 0
1 + 2 = 3
2 + 4 = 6
3 + 6 = 9
4 + 8 = 12
5 + 10 = 15
6 + 12 = 18
7 + 14 = 21
8 + 16 = 24
9 + 18 = 27
10 + 20 = 30
11 + 22 = 33
12 + 24 = 36
13 + 26 = 39
14 + 28 = 42
15 + 30 = 45
16 + 32 = 48
17 + 34 = 51
18 + 36 = 54
19 + 38 = 57
20 + 40 = 60
21 + 42 = 63
22 + 44 = 66
23 + 46 = 69
24 + 48 = 72
25 + 50 = 75
26 + 52 = 78
27 + 54 = 81
28 + 56 = 84
29 + 58 = 87
30 + 60 = 90
31 + 62 = 93
32 + 64 = 96
33 + 66 = 99
34 + 68 = 102
35 + 70 = 105
36 + 72 = 108
37 + 74 = 111
38 + 76 = 114
39 + 78 = 117
40 + 80 = 120
41 + 82 = 123
42 + 84 = 126
43 + 86 = 129
44 + 88 = 132
45 + 90 = 135
46 + 92 = 138
```

```
47 + 94 = 141
48 + 96 = 144
49 + 98 = 147
50 + 100 = 150
51 + 102 = 153
52 + 104 = 156
53 + 106 = 159
54 + 108 = 162
55 + 110 = 165
56 + 112 = 168
57 + 114 = 171
58 + 116 = 174
59 + 118 = 177
60 + 120 = 180
61 + 122 = 183
62 + 124 = 186
63 + 126 = 189
64 + 128 = 192
65 + 130 = 195
66 + 132 = 198
67 + 134 = 201
68 + 136 = 204
69 + 138 = 207
70 + 140 = 210
71 + 142 = 213
72 + 144 = 216
73 + 146 = 219
74 + 148 = 222
75 + 150 = 225
76 + 152 = 228
77 + 154 = 231
78 + 156 = 234
79 + 158 = 237
80 + 160 = 240
81 + 162 = 243
82 + 164 = 246
83 + 166 = 249
84 + 168 = 252
85 + 170 = 255
86 + 172 = 258
87 + 174 = 261
88 + 176 = 264
89 + 178 = 267
90 + 180 = 270
91 + 182 = 273
92 + 184 = 276
93 + 186 = 279
94 + 188 = 282
95 + 190 = 285
```

```
96 + 192 = 288
97 + 194 = 291
98 + 196 = 294
99 + 198 = 297
```

MATRIX MULTIPLICATION

```
%%cu

#include <stdio.h>

// CUDA kernel for matrix multiplication
__global__ void matrixMul(int* a, int* b, int* c, int rowsA, int
colsA, int colsB) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int sum = 0;
    if (row < rowsA && col < colsB) {
        for (int i = 0; i < colsA; i++) {
            sum += a[row * colsA + i] * b[i * colsB + col];
        }
        c[row * colsB + col] = sum;
    }
}

int main() {
    int rowsA = 10;  // Rows of matrix A
    int colsA = 10;  // Columns of matrix A
    int rowsB = colsA; // Rows of matrix B
    int colsB = 10;  // Columns of matrix B

    int* a, * b, * c;  // Host matrices
    int* dev_a, * dev_b, * dev_c;  // Device matrices

    // Allocate memory for host matrices
    a = (int*)malloc(rowsA * colsA * sizeof(int));
    b = (int*)malloc(rowsB * colsB * sizeof(int));
    c = (int*)malloc(rowsA * colsB * sizeof(int));

    // Initialize host matrices
    for (int i = 0; i < rowsA * colsA; i++) {
        a[i] = i;
    }
    for (int i = 0; i < rowsB * colsB; i++) {
        b[i] = 2 * i;
    }

    // Allocate memory on the device for device matrices
    cudaMalloc((void**)&dev_a, rowsA * colsA * sizeof(int));
    cudaMalloc((void**)&dev_b, rowsB * colsB * sizeof(int));
    cudaMalloc((void**)&dev_c, rowsA * colsB * sizeof(int));

    // Copy host matrices to device
    cudaMemcpy(dev_a, a, rowsA * colsA * sizeof(int),
cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, rowsB * colsB * sizeof(int),
cudaMemcpyHostToDevice);

    // Define grid and block dimensions
    dim3 blockSize(16, 16);
    dim3 gridSize((colsB + blockSize.x - 1) / blockSize.x, (rowsA +
blockSize.y - 1) / blockSize.y);

    // Launch kernel for matrix multiplication
    matrixMul<<<gridSize, blockSize>>>(dev_a, dev_b, dev_c, rowsA,
colsA, colsB);

    // Copy result from device to host
    cudaMemcpy(c, dev_c, rowsA * colsB * sizeof(int),
cudaMemcpyDeviceToHost);

    // Print result
    printf("Result:\n");
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            printf("%d ", c[i * colsB + j]);
        }
        printf("\n");
    }

    // Free device memory
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);

    // Free host memory
    free(a);
    free(b);
    free(c);

    return 0;
}
```

```
Result:
5700 5790 5880 5970 6060 6150 6240 6330 6420 6510
14700 14990 15280 15570 15860 16150 16440 16730 17020 17310
23700 24190 24680 25170 25660 26150 26640 27130 27620 28110
32700 33390 34080 34770 35460 36150 36840 37530 38220 38910
41700 42590 43480 44370 45260 46150 47040 47930 48820 49710
```

```
50700 51790 52880 53970 55060 56150 57240 58330 59420 60510
59700 60990 62280 63570 64860 66150 67440 68730 70020 71310
68700 70190 71680 73170 74660 76150 77640 79130 80620 82110
77700 79390 81080 82770 84460 86150 87840 89530 91220 92910
86700 88590 90480 92370 94260 96150 98040 99930 101820 103710
```