```
!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-37mde376
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-37mde376
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4287 sha256=23b392ce69ed4342c353bb19cbd11b7f7df2c74a115e
  Stored in directory: /tmp/pip-ephem-wheel-cache-7la0x2e0/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
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
```

```
%%cu

#include <stdio.h>
#include <stdlib.h>
#define N 1024
#define BLOCK_SIZE 32

__global__ void matrixMultiply(float *a, float *b, float *c, int n) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    float sum = 0;

    for (int k = 0; k < n; k++) {
      sum += a[row * n + k] * b[k * n + col];
    }
    c[row * n + col] = sum;
}

int main() {

    float *a, *b, *c;
    float *d_a, *d_b, *d_c;
    int size = N * N * sizeof(float);

    // Allocate memory on host

    a = (float *)malloc(size);
    b = (float *)malloc(size);
    c = (float *)malloc(size);

    // Initialize matrices

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
          a[i * N + j] = i + j;
          b[i * N + j] = i - j;
      }
    }

    // Allocate memory on device
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    // Copy matrices from host to device
    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

    // Define grid and block sizes
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dim3 dimGrid(N / BLOCK_SIZE, N / BLOCK_SIZE);

    // Launch kernel on device
    matrixMultiply<<<dimGrid, dimBlock>>>(d_a, d_b, d_c, N);
```

```
    // Copy result from device to host
    cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

    // Verify result
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
          float sum = 0;
          for (int k = 0; k < N; k++) {
              sum += a[i * N + k] * b[k * N + j];
          }
          if (c[i * N + j] != sum) {
            printf("Error: c[%d][%d] = %f\n", i, j, c[i * N + j]);
            break;
          }
      }
    }

    // Free memory on host and device
    free(a);
    free(b);
    free(c);

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```


✓ 9s completed at 10:42 PM