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

```
Collecting git+git://github.com/andreinechaev/nvcc4jupyter.git
  Cloning git://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-
  Running command git clone -q git://github.com/andreinechaev/nvcc4jupyter.git /
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-cp36-none-any.whl size
  Stored in directory: /tmp/pip-ephem-wheel-cache-ofia2bzv/wheels/10/c2/05/ca241
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>
#include <math.h>
#include<bits/stdc++.h>
#include<chrono>
using namespace std::chrono;
using namespace std;
#define N 5000
__global__ void vecAdd(double *a, double *b, double *c,int th){
    int id = threadIdx.x;
    __shared__ double temp[N];
    for(int i=id ; i<N ; i+=th){
        temp[i] = a[i] + b[i];
    }
    __syncthreads();
    if(id==0){
        double sum = 0;
        for(int i=0 ; i<N ; ++i){
            sum+=temp[i];
        }
        *c = sum;
    }
}

int main( int argc, char* argv[] ){
    double *a,*b,*c;
    double *d_a,*d_b,*d_c;
    size_t size = N*sizeof(double);

    a = (double*)malloc(size);
    b = (double*)malloc(size);
    b = (double*)malloc(size);
    c = (double*)malloc(sizeof(double));

    cudaMalloc(&d_a, size);
    cudaMalloc(&d_b, size);
    cudaMalloc(&d_c, sizeof(double));

    int i;
    for( i = 0; i < N; i++ ) {
        a[i] = rand()%100000 + (1.0/(rand()%1000));
        b[i] = 0;
    }

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

    int tt[10] ={1,2,4,8,16,32,64,128,256,500};
    for(int t=0 ; t<10 ; ++t){
        auto start = high_resolution_clock::now();
        vecAdd<<<1, tt[t]>>>(d_a, d_b, d_c,tt[t]);
        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<microseconds>(stop - start);
        cout << "Time taken by function: "<< duration.count() << " microseconds" << e
    }
    cudaMemcpy( c, d_c, sizeof(double), cudaMemcpyDeviceToHost );

    /*for(i=0; i<N; i++)
      printf("%lf ",a[i]);
    cout<<endl;
    for(i=0; i<N; i++)
      printf("%lf ",b[i]);
    cout<<endl;
    cout<<*c<<endl;*/

    cudaFree(d_a);cudaFree(d_b);cudaFree(d_c);
    free(a);free(b);free(c);

    return 0;
}
```

```
Time taken by function: 14 microseconds
Time taken by function: 7 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
Time taken by function: 3 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
```

```
%%cu
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include<bits/stdc++.h>
#include<chrono>
using namespace std::chrono;
using namespace std;
#define N 4
#define M 2
__global__ void vecAdd(double *a, double *b, double *c,int th){
    int id = threadIdx.x;
    __shared__ double temp[N];
    for(int i=id ; i<N ; i+=th){
        temp[i] = a[i] * b[i];
    }
    __syncthreads();
    if(id==0){
        double sum = 0;
        for(int i=0 ; i<N ; ++i){
            sum+=temp[i];
        }
        *c = sum;
    }
}

int main( int argc, char* argv[] ){
    double *a,*b,*c;
    double *d_a,*d_b,*d_c;

    size_t size = N*sizeof(double);

    a = (double*)malloc(size);
    b = (double*)malloc(size);
    c = (double*)malloc(sizeof(double));

    cudaMalloc(&d_a, size);
    cudaMalloc(&d_b, size);
    cudaMalloc(&d_c, sizeof(double));

    int i;
    for( i = 0; i < N; i++ ) {
        /*a[i] = rand()%100000 + (1.0/(rand()%1000));
        b[i] = rand()%100000 + (1.0/(rand()%1000));*/
        a[i] = rand()%10;
        b[i] = rand()%10;
    }

    // Copy host vectors to device
    cudaMemcpy( d_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy( d_b, b, size, cudaMemcpyHostToDevice);
    int tt[10] ={1,2,4,8,16,32,64,128,256,500};
    for(int t=0 ; t<10 ; ++t){
        auto start = high_resolution_clock::now();
        vecAdd<<<1, tt[t]>>>(d_a, d_b, d_c,tt[t]);
        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<microseconds>(stop - start);
        cout << "Time taken by function: "<< duration.count() << " microseconds" << e
    }

    cudaMemcpy( c, d_c, sizeof(double), cudaMemcpyDeviceToHost );

    /*for(i=0; i<N; i++)
      printf("%lf ",a[i]);
    cout<<endl;
    for(i=0; i<N; i++)
      printf("%lf ",b[i]);
    cout<<endl;
    cout<<*c<<endl;*/

    cudaFree(d_a);cudaFree(d_b);cudaFree(d_c);
    free(a);free(b);free(c);

    return 0;
}
```

```
Time taken by function: 14 microseconds
Time taken by function: 7 microseconds
Time taken by function: 5 microseconds
Time taken by function: 5 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
Time taken by function: 4 microseconds
Time taken by function: 6 microseconds
Time taken by function: 5 microseconds
Time taken by function: 5 microseconds
```