```
!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-qijt8ggn
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-qijt8ggn
  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=9b93440b9278cc7086add15cc40be6cb288a5c736da5
  Stored in directory: /tmp/pip-ephem-wheel-cache-mtmqod53/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>

// Size of array
#define N 1048576

// Kernel
__global__ void add_vectors(double *a, double *b, double *c)
{
  int id = blockDim.x * blockIdx.x + threadIdx.x;
  if(id < N) c[id] = a[id] + b[id];
}

// Main program
int main()
{
  // Number of bytes to allocate for N doubles
  size_t bytes = N*sizeof(double);

  // Allocate memory for arrays A, B, and C on host
  double *A = (double*)malloc(bytes);
  double *B = (double*)malloc(bytes);
  double *C = (double*)malloc(bytes);

  // Allocate memory for arrays d_A, d_B, and d_C on device
  double *d_A, *d_B, *d_C;
  cudaMalloc(&d_A, bytes);
  cudaMalloc(&d_B, bytes);
  cudaMalloc(&d_C, bytes);

  // Fill host arrays A and B
  for(int i=0; i<N; i++)
  {
    A[i] = 1.0;
    B[i] = 2.0;
  }

  // Copy data from host arrays A and B to device arrays d_A and d_B
  cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
  cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

  // Set execution configuration parameters
  //    thr_per_blk: number of CUDA threads per grid block
  //    blk_in_grid: number of blocks in grid
  int thr_per_blk = 256;
  int blk_in_grid = ceil( float(N) / thr_per_blk );

  // Launch kernel
  add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

  // Copy data from device array d_C to host array C
  cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);
  // Verify results
    double tolerance = 1.0e-14;
  for(int i=0; i<N; i++)
  {
    if( fabs(C[i] - 3.0) > tolerance)
    {
      printf("\nError: value of C[%d] = %d instead of 3.0\n\n", i, C[i]);
      exit(1);
    }
  }

  // Free CPU memory
  free(A);
  free(B);
  free(C);

  // Free GPU memory
  cudaFree(d_A);
  cudaFree(d_B);
  cudaFree(d_C);

  printf("\n---------------------------\n");
  printf("__SUCCESS__\n");
  printf("---------------------------\n");
  printf("N                 = %d\n", N);
  printf("Threads Per Block = %d\n", thr_per_blk);
  printf("Blocks In Grid    = %d\n", blk_in_grid);
  printf("---------------------------\n\n");

  return 0;
}
```

```

---------------------------
__SUCCESS__
---------------------------
N                 = 1048576
Threads Per Block = 256
Blocks In Grid    = 4096
---------------------------
```

Colab paid products - Cancel contracts here

✓ 0s completed at 10:32 PM