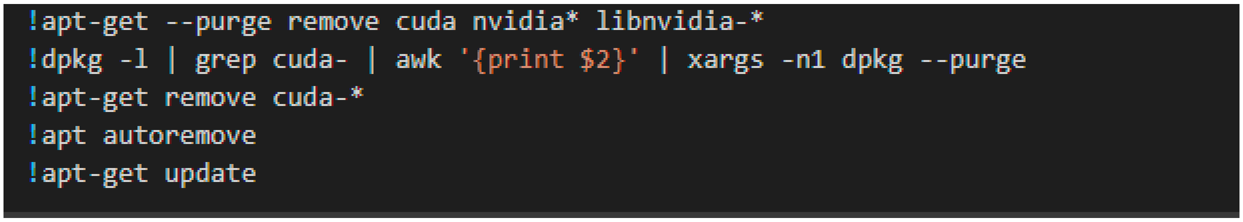
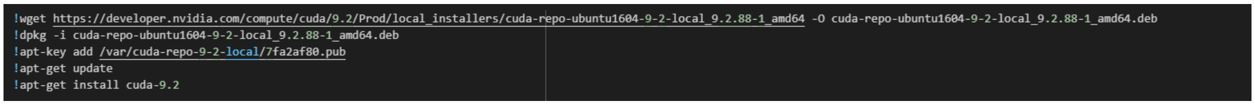
CUDA

Rao Nauman

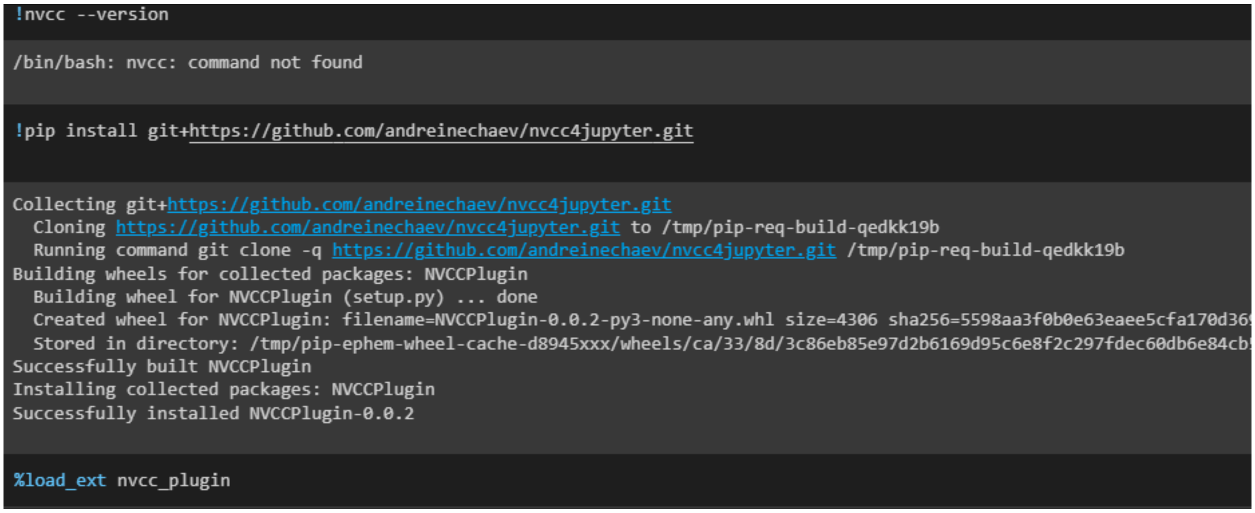
P19-0073 [Company address]



```
!apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update
```



```
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2
```



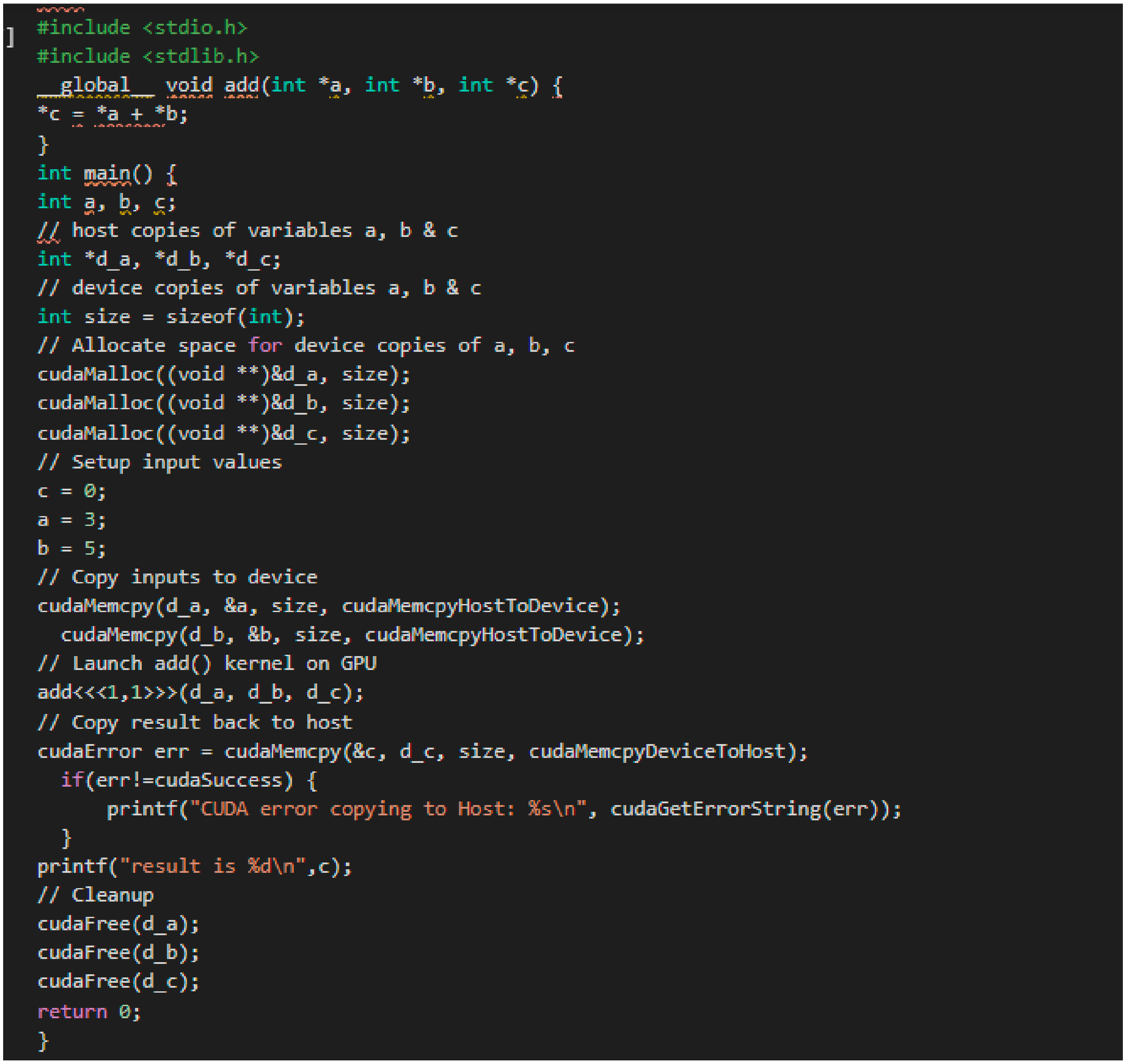
```
!nvcc --version
```

```
/bin/bash: nvcc: command not found
```

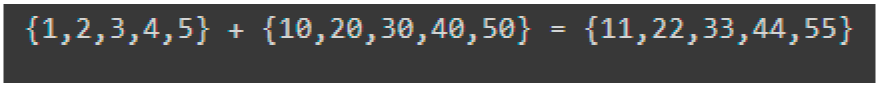
```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-qedkk19b
  Running command git clone -q https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-qedkk19b
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=4306 sha256=5598aa3f0b0e63eaee5cfa170d36
  Stored in directory: /tmp/pip-ephem-wheel-cache-d8945xxx/wheels/ca/33/8d/3c86eb85e97d2b6169d95c6e8f2c297fdec60db6e84cb
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

```
%load_ext nvcc_plugin
```

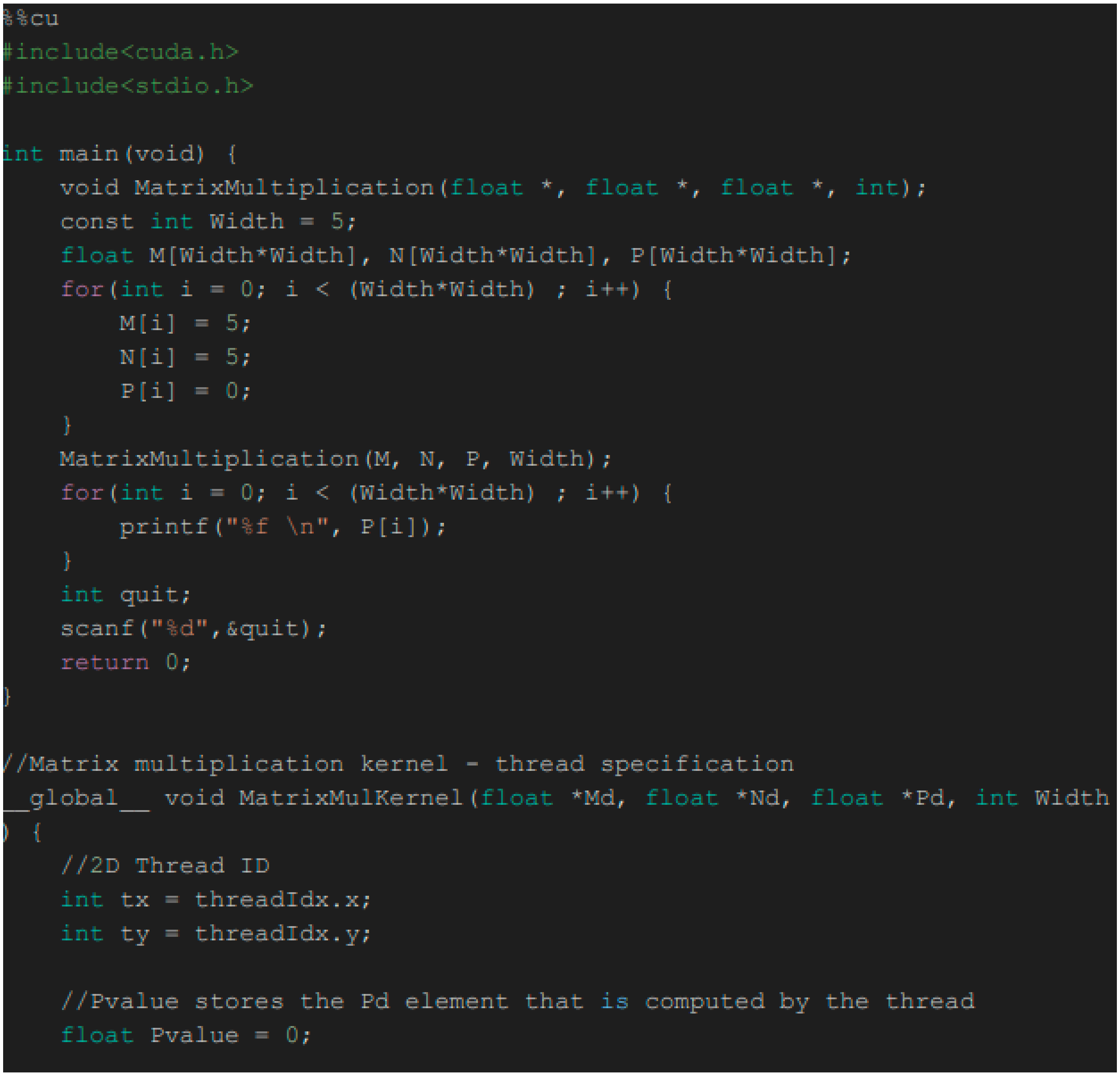


```
#include <stdio.h>
#include <stdlib.h>
__global__ void add(int *a, int *b, int *c) {
*c = *a + *b;
}
int main() {
int a, b, c;
// host copies of variables a, b & c
int *d_a, *d_b, *d_c;
// device copies of variables a, b & c
int size = sizeof(int);
// Allocate space for device copies of a, b, c
cudaMalloc((void **)&d_a, size);
cudaMalloc((void **)&d_b, size);
cudaMalloc((void **)&d_c, size);
// Setup input values
c = 0;
a = 3;
b = 5;
// Copy inputs to device
cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
  cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);
// Launch add() kernel on GPU
add<<<1,1>>>(d_a, d_b, d_c);
// Copy result back to host
cudaError err = cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
  if(err!=cudaSuccess) {
      printf("CUDA error copying to Host: %s\n", cudaGetErrorString(err));
  }
printf("result is %d\n",c);
// Cleanup
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
return 0;
}
```

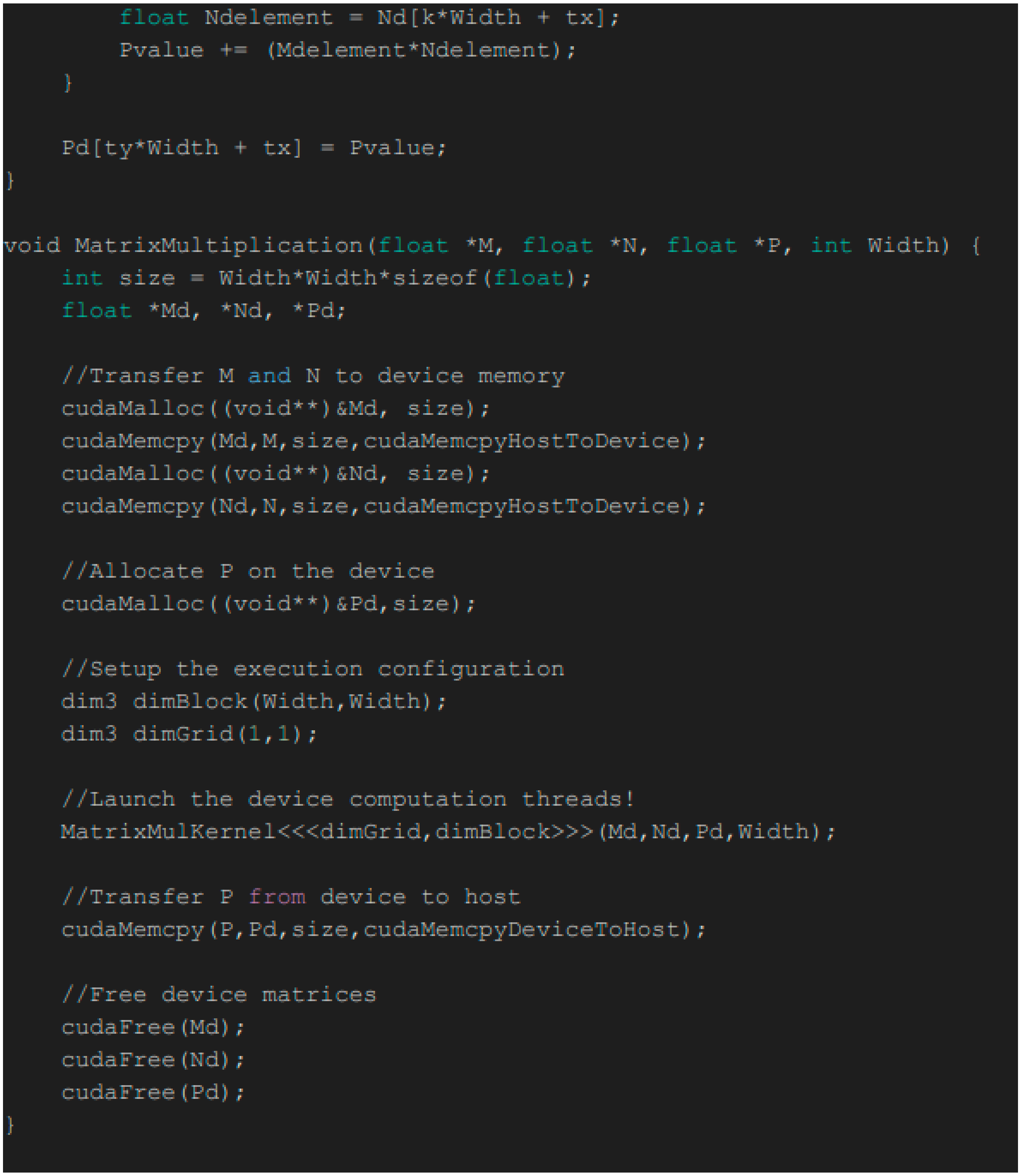


```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```

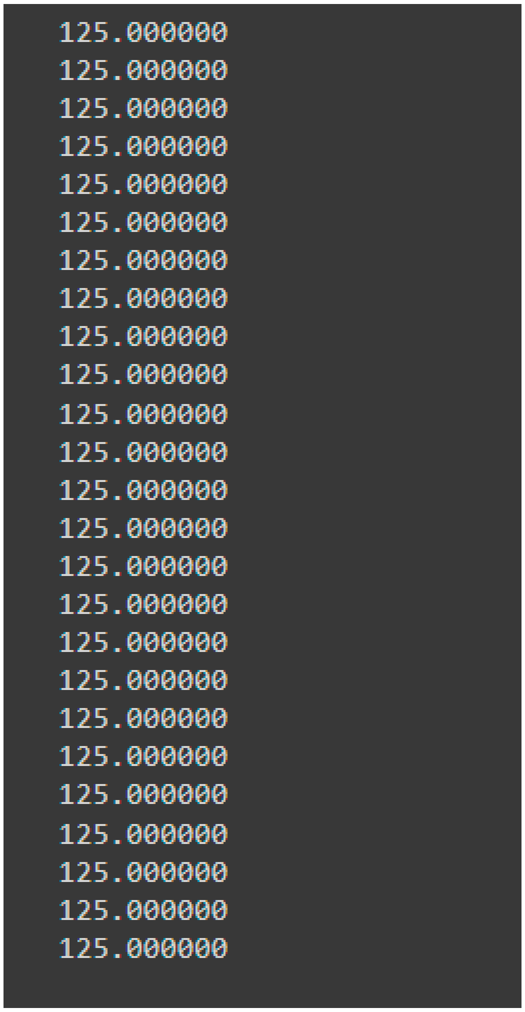
Matrix:



```
%%cu
#include<cuda.h>
#include<stdio.h>

int main(void) {
    void MatrixMultiplication(float *, float *, float *, int);
    const int Width = 5;
    float M[Width*Width], N[Width*Width], P[Width*Width];
    for(int i = 0; i < (Width*Width) ; i++) {
        M[i] = 5;
        N[i] = 5;
        P[i] = 0;
    }
    MatrixMultiplication(M, N, P, Width);
    for(int i = 0; i < (Width*Width) ; i++) {
        printf("%f \n", P[i]);
    }
    int quit;
    scanf("%d",&quit);
    return 0;
}

//Matrix multiplication kernel - thread specification
__global__ void MatrixMulKernel(float *Md, float *Nd, float *Pd, int Width
) {
    //2D Thread ID
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    //Pvalue stores the Pd element that is computed by the thread
    float Pvalue = 0;
```



```
        float Ndelement = Nd[k*Width + tx];
        Pvalue += (Mdelement*Ndelement);
    }

    Pd[ty*Width + tx] = Pvalue;
}

void MatrixMultiplication(float *M, float *N, float *P, int Width) {
    int size = Width*Width*sizeof(float);
    float *Md, *Nd, *Pd;

    //Transfer M and N to device memory
    cudaMalloc((void**)&Md, size);
    cudaMemcpy(Md,M,size,cudaMemcpyHostToDevice);
    cudaMalloc((void**)&Nd, size);
    cudaMemcpy(Nd,N,size,cudaMemcpyHostToDevice);

    //Allocate P on the device
    cudaMalloc((void**)&Pd,size);

    //Setup the execution configuration
    dim3 dimBlock(Width,Width);
    dim3 dimGrid(1,1);

    //Launch the device computation threads!
    MatrixMulKernel<<<dimGrid,dimBlock>>>(Md,Nd,Pd,Width);

    //Transfer P from device to host
    cudaMemcpy(P,Pd,size,cudaMemcpyDeviceToHost);

    //Free device matrices
    cudaFree(Md);
    cudaFree(Nd);
    cudaFree(Pd);
}
```



```
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
125.000000
```