```
%load_ext nvcc_plugin
```

```
%%file mat_example.cu


/*
 * Compile:  nvcc -o matrix_sum matrix_sum.cu
 * Run:      ./matrix_sum <10> <10>
 *              m = the number of rows
 *              n = the number of columns
 */
 #include <stdio.h>
 #include <stdlib.h>
 #include <math.h>

 /*--------------------------------------------------------------------
  *  matrix_sum
  */
 __global__ void matrix_sum(float A[], float B[], float C[], int m, int n) {
    /* blockDim.x = threads_per_block  */
    int ij = blockDim.x * blockIdx.x + threadIdx.x;

    /* The test shouldn't be necessary */
    if (blockIdx.x < m && threadIdx.x < n)
       C[ij] = A[ij] + B[ij];
 }  /* matrix_sum */


 /*--------------------------------------------------------------------
  get_matrix
  */
  void get_matrix(float A[], int m, int n) {
    int i, j;

    for (i = 0; i < m; i++)
       for (j = 0; j < n; j++)
```

```
         scanf("%f", &A[i*n+j]);
}  /* get_matrix */


/*--------------------------------------------------------------------
 show_matrix
 */
void show_matrix(char title[], float A[], int m, int n) {
   int i, j;

   printf("%s\n", title);
   for (i = 0; i < m; i++) {
      for (j = 0; j < n; j++)
         printf("%.1f ", A[(i*n)+j]);
      printf("\n");
   }
}  /* show_matrix */


/* Host code - CPU  */
int main(int argc, char* argv[]) {
   int m, n;
   float *h_A, *h_B, *h_C;
   float *d_A, *d_B, *d_C;
   size_t size;

   /* Get size of matrixes */
   if (argc != 3) {
      fprintf(stderr, "usage: %s <row count> <col count>\n", argv[0]);
      exit(0);
   }
   m = strtol(argv[1], NULL, 10);
   n = strtol(argv[2], NULL, 10);
   printf("m = %d, n = %d\n", m, n);
   size = m*n*sizeof(float);

  /* declare pointers to vectors in device memory and allocate memory */
   h_A = (float*) malloc(size);
   h_B = (float*) malloc(size);
```

```
    h_C = (float*) malloc(size);

    printf("Enter some numbers and we create 2 matrix , A and B , first A : \n");
    get_matrix(h_A, m, n);
    printf("Enter second matrix: ");
    get_matrix(h_B, m, n);

    show_matrix("matrix A =", h_A, m, n);
    show_matrix("matrix B =", h_B, m, n);

    /* Allocate matrixes in device memory */
    cudaMalloc(&d_A, size);
    cudaMalloc(&d_B, size);
    cudaMalloc(&d_C, size);

    /* Copy matrixes from host memory to device memory */
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    /* Invoke kernel using m thread blocks, each of    */
    /* which contains n threads                        */
    dim3 block_size( 16, 16 );
    dim3 num_blocks( ( n - 1 + block_size.x ) / block_size.x,
                     ( m - 1 + block_size.y ) / block_size.y );

    matrix_sum<<<block_size, num_blocks>>>(d_A, d_B, d_C, m, n);

    /* Wait for the kernel to complete */
    cudaThreadSynchronize();

    /* Copy result from device memory to host memory */
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    show_matrix("The result is: ", h_C, m, n);

    /* Free device memory */
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
```

```
    /* Free host memory */
    free(h_A);
    free(h_B);
    free(h_C);

    return 0;
 }  /* main */
```

```
 Writing mat_example.cu
```

```
!nvcc mat_example.cu -o mat_example
```

```
!./mat_example 2 2
```

```
 m = 2, n = 2
 Enter some numbers and we create 2 matrix , A and B , first A :
 5
 4
 2
 7
 Enter second matrix: 4
 5
 2
 1
 matrix A =
 5.0 4.0
 2.0 7.0
 matrix B =
 4.0 5.0
 2.0 1.0
 The result is:
 9.0 9.0
 0.0 0.0
```

```
!nvprof ./mat_example 2 2
```

```
m = 2, n = 2
Enter some numbers and we create 2 matrix , A and B , first A :
3
3
4
3
Enter second matrix: 2
4
5
6
matrix A =
3.0 3.0
4.0 3.0
matrix B =
2.0 4.0
5.0 6.0
==18345== NVPROF is profiling process 18345, command: ./mat_example 2 2
The result is:
5.0 7.0
0.0 0.0
==18345== Profiling application: ./mat_example 2 2
==18345== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   38.96%  3.8400us         2  1.9200us  1.5680us  2.2720us  [CUDA memcpy HtoD]
                   35.06%  3.4560us         1  3.4560us  3.4560us  3.4560us  matrix_sum(float*, float*, float*, int, ir
                   25.97%  2.5600us         1  2.5600us  2.5600us  2.5600us  [CUDA memcpy DtoH]
      API calls:   99.56%  278.19ms         3  92.730ms  2.7840us  278.18ms  cudaMalloc
                    0.19%  531.52us         1  531.52us  531.52us  531.52us  cuDeviceTotalMem
                    0.08%  233.51us        96  2.4320us     120ns  94.774us  cuDeviceGetAttribute
                    0.05%  149.00us         1  149.00us  149.00us  149.00us  cudaLaunchKernel
                    0.05%  139.07us         3  46.355us  6.2560us  116.55us  cudaFree
                    0.04%  104.43us         3  34.810us  13.150us  67.318us  cudaMemcpy
                    0.02%  46.203us         1  46.203us  46.203us  46.203us  cuDeviceGetName
                    0.00%  8.9010us         1  8.9010us  8.9010us  8.9010us  cudaThreadSynchronize
                    0.00%  6.7680us         1  6.7680us  6.7680us  6.7680us  cuDeviceGetPCIBusId
                    0.00%  1.9550us         3     651ns     151ns     968ns  cuDeviceGetCount
                    0.00%  1.8480us         2     924ns     329ns  1.5190us  cuDeviceGet
```

```
%%file mat_2.cu
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


__global__ void Mat_sum(float A[], float B[], float C[], int m, int n) {
   /* blockDim.x = threads_per_block   */

   int ij = blockDim.x * blockIdx.x + threadIdx.x;

   /* Not necessary Test*/
   if (blockIdx.x < m && threadIdx.x < n)
      C[ij] = A[ij] + B[ij];
}  /* Mat_sum */


/*--------------------------------------------------------------------*/

void get_matrix(float A[], int m, int n) {
   int i, j;

   for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
         scanf("%f", &A[i*n+j]);
}  /* get_matrix */


/*--------------------------------------------------------------------- */
void show_matrix(char title[], float A[], int m, int n) {
   int i, j;

   printf("%s\n", title);
   for (i = 0; i < m; i++) {
      for (j = 0; j < n; j++)
         printf("%.1f ", A[(i*n)+j]);
      printf("\n");
   }
}  /* show_matrix */
```

```
 /* Host code - CPU*/
 int main(int argc, char* argv[]) {
    int m, n;
    float *h_A, *h_B, *h_C;
    float *d_A, *d_B, *d_C;
    size_t size;

    /* Get size */
    if (argc != 3) {
       fprintf(stderr, "usage: %s <row count> <col count>\n", argv[0]);
       exit(0);
    }
    m = strtol(argv[1], NULL, 10);
    n = strtol(argv[2], NULL, 10);
    printf("m = %d, n = %d\n", m, n);
    size = m*n*sizeof(float);

/* declare pointers to vectors in device memory and allocate memory */
    h_A = (float*) malloc(size);
    h_B = (float*) malloc(size);
    h_C = (float*) malloc(size);

    printf("Enter some numbers and we create 2 matrix , A and B , first A : \n");
    get_matrix(h_A, m, n);
    printf("Enter second matrix: ");
    get_matrix(h_B, m, n);

    show_matrix("matrix A =", h_A, m, n);
    show_matrix("matrix B =", h_B, m, n);

    /* Allocate matrixes in device memory */
    cudaMalloc(&d_A, size);
    cudaMalloc(&d_B, size);
    cudaMalloc(&d_C, size);

    /* Copy matrixes from host memory to device memory */
```

```
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    /* Invoke kernel using m thread blocks, each of    */
    /* which contains n threads                        */
    Mat_sum<<<m, n>>>(d_A, d_B, d_C, m, n);

    /* Wait for the kernel to complete */
    cudaThreadSynchronize();

    /* Copy result from device memory to host memory */
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    show_matrix("The sum is: ", h_C, m, n);

    /* Free device memory */
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    /* Free host memory */
    free(h_A);
    free(h_B);
    free(h_C);

    return 0;
 }  /* main */
```

```
Writing mat_2.cu
```

```
!nvcc mat_2.cu -o mat_2
```

```
!nvprof·./mat_2·2·2
```

```
m = 2, n = 2
Enter some numbers and we create 2 matrix , A and B , first A :
2
3
```

```
3
2
Enter second matrix: 3
2
3
2
matrix A =
2.0 3.0
3.0 2.0
matrix B =
3.0 2.0
3.0 2.0
==18457== NVPROF is profiling process 18457, command: ./mat_2 2 2
The sum is:
5.0 5.0
6.0 4.0
==18457== Profiling application: ./mat_2 2 2
==18457== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   43.12%  4.4160us         2  2.2080us  2.0800us  2.3360us  [CUDA memcpy HtoD]
                   30.62%  3.1360us         1  3.1360us  3.1360us  3.1360us  Mat_sum(float*, float*, float*, int, int)
                   26.25%  2.6880us         1  2.6880us  2.6880us  2.6880us  [CUDA memcpy DtoH]
      API calls:   99.41%  195.95ms         3  65.317ms  2.6650us  195.94ms  cudaMalloc
                    0.24%  465.29us         1  465.29us  465.29us  465.29us  cuDeviceTotalMem
                    0.12%  235.30us         1  235.30us  235.30us  235.30us  cudaLaunchKernel
                    0.10%  194.65us        96  2.0270us     124ns  88.250us  cuDeviceGetAttribute
                    0.08%  150.67us         3  50.224us  6.3800us  121.06us  cudaFree
                    0.03%  64.330us         3  21.443us  12.545us  26.229us  cudaMemcpy
                    0.01%  26.477us         1  26.477us  26.477us  26.477us  cuDeviceGetName
                    0.00%  7.5460us         1  7.5460us  7.5460us  7.5460us  cudaThreadSynchronize
                    0.00%  6.6470us         1  6.6470us  6.6470us  6.6470us  cuDeviceGetPCIBusId
                    0.00%  1.8700us         3     623ns     139ns     908ns  cuDeviceGetCount
                    0.00%  1.5840us         2     792ns     312ns  1.2720us  cuDeviceGet
```

```
!nvprof ./mat_example 4 4
```

```
8
m = 4, n = 4
Enter some numbers and we create 2 matrix , A and B , first A :
5
5
```

```
3
5
3
8
3
5
3
5
3
3
5
3
5
Enter second matrix: 3
5
8
3
5
3
5
3
3
3
3
3
5
3
3
matrix A =
8.0 5.0 5.0 3.0
5.0 3.0 8.0 3.0
5.0 3.0 5.0 3.0
3.0 5.0 3.0 5.0
matrix B =
3.0 5.0 8.0 3.0
5.0 3.0 5.0 3.0
3.0 3.0 3.0 3.0
3.0 5.0 3.0 3.0
==18740== NVPROF is profiling process 18740, command: ./mat_example 4 4
The result is:
11.0 10.0 13.0 6.0
```

```
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
==18740== Profiling application: ./mat_example 4 4
==18740== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   39.29%  3.8720us         2  1.9360us  1.5680us  2.3040us  [CUDA memcpy HtoD]
                   34.74%  3.4240us         1  3.4240us  3.4240us  3.4240us  matrix_sum(float*, float*, float*, int,
                   25.97%  2.5600us         1  2.5600us  2.5600us  2.5600us  [CUDA memcpy DtoH]
      API calls:   99.36%  189.53ms         3  63.177ms  2.5080us  189.52ms  cudaMalloc
```

```
!nvprof ./mat_2 4 4
```

```
5
5
25
2
1
0
3
1
3
Enter second matrix: 3
2
3
5
5
6
7
8
2

1
0
3
4
5
6
7
matrix A =
2.0 2.0 2.0 2.0
```

```
2.0 3.0 4.0 5.0
5.0 25.0 2.0 1.0
0.0 3.0 1.0 3.0
matrix B =
3.0 2.0 3.0 5.0
5.0 6.0 7.0 8.0
2.0 1.0 0.0 3.0
4.0 5.0 6.0 7.0
==18623== NVPROF is profiling process 18623, command: ./mat_2 4 4
The sum is:
5.0 4.0 5.0 7.0
7.0 9.0 11.0 13.0
7.0 26.0 2.0 4.0
4.0 8.0 7.0 10.0
==18623== Profiling application: ./mat_2 4 4
==18623== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   42.20%  3.8080us         2  1.9040us  1.5360us  2.2720us  [CUDA memcpy HtoD]
                   31.21%  2.8160us         1  2.8160us  2.8160us  2.8160us  Mat_sum(float*, float*, float*, int, int
                   26.60%  2.4000us         1  2.4000us  2.4000us  2.4000us  [CUDA memcpy DtoH]
      API calls:   99.43%  189.98ms         3  63.326ms  2.1060us  189.97ms  cudaMalloc
                    0.23%  444.37us         1  444.37us  444.37us  444.37us  cuDeviceTotalMem
                    0.10%  196.46us        96  2.0460us     112ns  80.243us  cuDeviceGetAttribute
                    0.10%  186.26us         1  186.26us  186.26us  186.26us  cudaLaunchKernel
                    0.08%  146.70us         3  48.899us  5.7160us  125.29us  cudaFree
                    0.03%  61.510us         3  20.503us  12.236us  25.471us  cudaMemcpy
                    0.02%  40.828us         1  40.828us  40.828us  40.828us  cuDeviceGetName
                    0.00%  9.2440us         1  9.2440us  9.2440us  9.2440us  cudaThreadSynchronize
                    0.00%  6.3720us         1  6.3720us  6.3720us  6.3720us  cuDeviceGetPCIBusId
                    0.00%  1.9720us         3     657ns     219ns     942ns  cuDeviceGetCount
                    0.00%  1.5880us         2     794ns     227ns  1.3610us  cuDeviceGet
```