exercise1:

change h_A to d_A in line 54

```
    gpuCheck( hipMemcpy(d_A, h_A, bytes, hipMemcpyHostToDevice) );
```

exercise2:

add the code in line 66

```
    gpuCheck( hipMemcpy(h_A, d_A, bytes, hipMemcpyDeviceToHost) );
```

exercise3:

add code in line 21

```
  if(id<n) A[id]= A[id]* A[id];
// if use math fuction : pow(A[id],2 ); error as follow
// Error: h_A[53] = 2808 instead of 2809
```

exercise4:

Kernel is as follow

```
/* --------------------------------------------------
Matrix multiply kernel
-------------------------------------------------- */
__global__ void matrix_multiply(double *A, double *B, double *C, int n)
{
    int col = blockDim.x * blockIdx.x + threadIdx.x;
    int row = blockDim.y * blockIdx.y + threadIdx.y;

    if (col < n && row < n){

        int index = n * row + col;
        double element = 0.0;

        for (int i=0; i<n; i++){

            int row_index = n * row + i;
            int col_index = n * i   + col;

            element = element + A[row_index] * B[col_index]; //multiply the data
in Martix A and Martix B
        }

        C[index]=element; //write the element to C
    }
}
```

exercise5:

通过比较发现使用hipBLAS version of DGEMM ，运行的速度更快，效率更高，比用自己实现kenerl完成矩阵乘法的方式提速近8倍。

	Name	Calls	TotalDurationNs	AverageNs	Percentage
1	matrix_multiply(double, *double*, double*, int)	1	3226886	3226886	88.0429518540558
2	hipBLAS version of DGEMM	1	438241	438241	11.957048145944192

exercise6:

cuda-->hip修改的部分

```
[HIPIFY] info: file './exercises/06_hipify_pingpong/pingpong.cu' statistics:
  CONVERTED refs count: 21
  TOTAL lines of code: 104
  WARNINGS: 0
[HIPIFY] info: CONVERTED refs by names:
  cudaDeviceSynchronize => hipDeviceSynchronize: 1
  cudaError_t => hipError_t: 1
  cudaEventCreate => hipEventCreate: 2
  cudaEventElapsedTime => hipEventElapsedTime: 1
  cudaEventRecord => hipEventRecord: 2
  cudaEventSynchronize => hipEventSynchronize: 1
  cudaEvent_t => hipEvent_t: 1
  cudaGetErrorString => hipGetErrorString: 1
  cudaMalloc => hipMalloc: 1
  cudaMallocHost => hipHostMalloc: 1
  cudaMemcpy => hipMemcpy: 4
  cudaMemcpyHostToDevice => hipMemcpyHostToDevice: 4
  cudaSuccess => hipSuccess: 1
```

带宽测试：

H2D最好峰值27.862368899GB/s

D2H最好峰值28.116584723GB/s

```
----- H2D----
Buffer Size (MiB):    0.007812500, Time (ms):    0.432633996, Bandwidth (GB/s):
0.946758701
Buffer Size (MiB):    0.015625000, Time (ms):    0.462233007, Bandwidth (GB/s):
1.772266341
Buffer Size (MiB):    0.031250000, Time (ms):    0.496313006, Bandwidth (GB/s):
3.301142588
Buffer Size (MiB):    0.062500000, Time (ms):    0.616631985, Bandwidth (GB/s):
5.314028596
Buffer Size (MiB):    0.125000000, Time (ms):    0.773428977, Bandwidth (GB/s):
8.473434793
```

```
Buffer Size (MiB):    0.250000000, Time (ms):    1.105584979, Bandwidth (GB/s):
11.855443271
Buffer Size (MiB):    0.500000000, Time (ms):    1.880133986, Bandwidth (GB/s):
13.942836090
Buffer Size (MiB):    1.000000000, Time (ms):    3.336113930, Bandwidth (GB/s):
15.715530436
Buffer Size (MiB):    2.000000000, Time (ms):    5.778161049, Bandwidth (GB/s):
18.147226966
Buffer Size (MiB):    4.000000000, Time (ms):   11.227206230, Bandwidth (GB/s):
18.679197273
Buffer Size (MiB):    8.000000000, Time (ms):   21.916187286, Bandwidth (GB/s):
19.137927347
Buffer Size (MiB):   16.000000000, Time (ms):   42.918327332, Bandwidth (GB/s):
19.545514752
Buffer Size (MiB):   32.000000000, Time (ms):   85.535835266, Bandwidth (GB/s):
19.614254012
Buffer Size (MiB):   64.000000000, Time (ms):  159.622558594, Bandwidth (GB/s):
21.021108981
Buffer Size (MiB):  128.000000000, Time (ms):  290.969421387, Bandwidth (GB/s):
23.063888872
Buffer Size (MiB):  256.000000000, Time (ms):  511.107788086, Bandwidth (GB/s):
26.260160993
Buffer Size (MiB):  512.000000000, Time (ms):  969.863586426, Bandwidth (GB/s):
27.677650729
Buffer Size (MiB): 1024.000000000, Time (ms): 1926.867431641, Bandwidth (GB/s):
27.862368899
----- D2H -----
Buffer Size (MiB):    0.007812500, Time (ms):    0.445273995, Bandwidth (GB/s):
0.919883048
Buffer Size (MiB):    0.015625000, Time (ms):    0.466073990, Bandwidth (GB/s):
1.757660839
Buffer Size (MiB):    0.031250000, Time (ms):    0.515833974, Bandwidth (GB/s):
3.176215765
Buffer Size (MiB):    0.062500000, Time (ms):    0.586713016, Bandwidth (GB/s):
5.585013304
Buffer Size (MiB):    0.125000000, Time (ms):    0.786390007, Bandwidth (GB/s):
8.333778336
Buffer Size (MiB):    0.250000000, Time (ms):    1.171664953, Bandwidth (GB/s):
11.186815791
Buffer Size (MiB):    0.500000000, Time (ms):    1.964934945, Bandwidth (GB/s):
13.341103259
Buffer Size (MiB):    1.000000000, Time (ms):    3.307157040, Bandwidth (GB/s):
15.853132879
Buffer Size (MiB):    2.000000000, Time (ms):    5.848566055, Bandwidth (GB/s):
17.928770746
Buffer Size (MiB):    4.000000000, Time (ms):   11.176016808, Bandwidth (GB/s):
18.764753455
Buffer Size (MiB):    8.000000000, Time (ms):   22.144838333, Bandwidth (GB/s):
18.940323415
Buffer Size (MiB):   16.000000000, Time (ms):   43.046489716, Bandwidth (GB/s):
19.487321859
Buffer Size (MiB):   32.000000000, Time (ms):   85.300193787, Bandwidth (GB/s):
19.668438318
Buffer Size (MiB):   64.000000000, Time (ms):  166.399322510, Bandwidth (GB/s):
20.165005178
```

```
Buffer Size (MiB):  128.000000000, Time (ms):  289.394073486, Bandwidth (GB/s):
23.189439642
Buffer Size (MiB):  256.000000000, Time (ms):  508.097656250, Bandwidth (GB/s):
26.415734524
Buffer Size (MiB):  512.000000000, Time (ms):  976.943847656, Bandwidth (GB/s):
27.477060902
Buffer Size (MiB): 1024.000000000, Time (ms): 1909.445678711, Bandwidth (GB/s):
28.116584723
```

exercise7:

```
/* --------------------------------------------------
Matrix multiply kernel
-------------------------------------------------- */
__global__ void matrix_multiply(double *A, double *B, double *C, int n)
{
    __shared__ double s_A[THREADS_PER_BLOCK_Y][THREADS_PER_BLOCK_X];
    __shared__ double s_B[THREADS_PER_BLOCK_Y][THREADS_PER_BLOCK_X];
int col  = blockDim.x * blockIdx.x + threadIdx.x;
int row  = blockDim.y * blockIdx.y + threadIdx.y;

int lcol = threadIdx.x;
int lrow = threadIdx.y;

int index  = n * row + col;

if (col < n && row < n){

    int THREADS_PER_BLOCK = THREADS_PER_BLOCK_Y;
    int num_chunks        = n / THREADS_PER_BLOCK;

    double element = 0.0;

    for (int chunk=0; chunk<num_chunks; chunk++){

        // TODO: Read data from global GPU memory into shared memory
        for (int j=0; j<THREADS_PER_BLOCK; j++){
           s_A[lrow][j]=A[j+chunk*THREADS_PER_BLOCK+col*n];
           s_B[j][lcol]=B[(j+THREADS_PER_BLOCK*chunk)*n+row];
        }

        __syncthreads();

        for (int i=0; i<THREADS_PER_BLOCK; i++){
            element = element + s_A[lrow][i] * s_B[i][lcol];
        }

        __syncthreads();
    }

    C[index] = element;
  }
}
```

结果：

```
==============================
__SUCCESS__
------------------------------
N                  : 1024
X Blocks in Grid   : 64
X Threads per Block: 16
Y Blocks in Grid   : 64
Y Threads per Block: 16
==============================
```