CUDA编程简介

陈航

嵌入式系统实验室
EMBEDDED SYSTEM LABORATORY
SUZHOU INSTITUTE FOR ADVANCED STUDY OF USTC

在CPU断运行的程序（vectoradd.cu）

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>
#include "vector_kernel.cu"
#define eps 1e-9

int main(int argc, char **argv)
{
        int numElements = 50000 ;
        size_t size = numElements * sizeof(float) ;

        printf("[Vector addtion of %d elements]\n", numElements) ;
        float *A = (float *)malloc(size) ;
        float *B = (float *)malloc(size) ;
        float *C = (float *)malloc(size) ;
        float *D = (float *)malloc(size) ;
```

```
        for(int i = 0; i < numElements; ++i)
        {
                A[i] = rand()/(float)RAND_MAX ;
                B[i] = rand()/(float)RAND_MAX ;
        }

printf("CPU result\n");
for(int i = 0;i < numElements; ++i)
{
      C[i] = A[i] + B[i];
}
        //alloc the device input vector A
        float *d_A = NULL ;
        cudaMalloc((void **)&d_A, size) ;
        float *d_B = NULL ;
        cudaMalloc((void **)&d_B, size) ;
        float *d_C = NULL ;
        cudaMalloc((void**)&d_C, size) ;
```

```
printf("copy input data from the host memory to the CUDA device\n") ;
cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice) ;
cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice) ;

//Launch the Vector Add CUDA Kernel
int threadsPerBlock = 256 ;
int blocksPerGrid = (numElements + threadsPerBlock - 1)/threadsPerBlock ;
printf("CUDA kernel launch with %d blocks of %d threads\n", blocksPerGrid,
threadsPerBlock) ;

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, numElements) ;
cudaGetLastError() ;

//copy data from cuda memory to the host memory
printf("copy output data from the CUDA device to the host memory\n") ;
```

```
//          cudaMemcpy(d_C, D, size, cudaMemcpyDeviceToHost) ;
            cudaMemcpy(D, d_C, size, cudaMemcpyDeviceToHost) ;

            //verify that the result vector is correct
            for(int i = 0; i < numElements; ++i)
            {
                        if(fabs( C[i] - D[i]) > eps)
                        {
                                    fprintf(stderr, "Result verification failed at element %d!\n", i) ;
                                    exit(EXIT_FAILURE) ;
                        }
            }
printf("Test PASSED\n") ;
```

```
    //free device global memory
    cudaFree(d_A) ;
    cudaFree(d_B) ;
    cudaFree(d_C) ;

    //free host memory
    free(A) ;
    free(B) ;
    free(C) ;

    //reset the device and exit
    cudaDeviceReset() ;

    printf("Done\n") ;
    return 0 ;
}
```

在GPU端运行的程序(vectoradd_kernel.cu)

```
#ifndef __VECTORADD_KERNEL__
#define __VECTORADD_KERNEL__

__global__ void vectorAdd(const float *A, const float *B, float *C, int numElements)
{
        int i = blockDim.x* blockIdx.x + threadIdx.x ;

        if(i < numElements)
                C[i] = A[i] + B[i] ;
}

#endif
```

编译选项：nvcc –g filename.cu –o filename

eg: nvcc –g vectoradd.cu –o vectoradd

作业提交方式

bsub –q c2050 –o %J.log –e %J.err ./vectoradd

注：请大家在服务器上以自己的学号命名建立文件夹。

Kernel函数说明

<<<>>>运算符对kernel函数完整的执行参数形式是

<<<Dg, Db, Ns, S>>>,其中个参数含义分别是：

Dg：用于定义整个grid的维度和尺寸，为dim3类型。Dim3 Dg(Dg.x, Dg.y, 1)表示grid中每行有Dg.x个block，每列Dg.y个block，第三维恒为1

Db: 用于定义每个block的维度和尺寸。Dim3 Db(Db.x, Db.y Db.z)表示每行有Db.x个thread，每列有Db.y个thread，高度为Db.z。这个block可以定义Db.x*Db.y*Db.z个线程

Ns: 是一个可选参数，用于设置每个block除了静态分配的shared memory以外，最多能够动态分配的shared memory大小，单位为byte。当不需要动态分配时，这个值可以写成0，或者省略不写

S: 是一个cudaStream_t类型的可选参数，初始值为0