HanLP Handbook

version 1.0

hankcs

January 03, 2018

Contents

Welcome to CUDA Programming documentation!

Contents:

1helloword

1. stepsby 1hello 写最简单的cuda程序

```
#include "../common/common.h"
#include <stdio.h>

/*
 * A simple introduction to programming in CUDA. This program prints "Hello
 * World from GPU! from 10 CUDA threads running on the GPU.
 */

__global__ void helloFromGPU(int count)
{
    for (int i=blockIdx.x * blockDim.x + threadIdx.x; i < count; i += blockDim.x * gridDim.x  )
    {
    printf("gridDim.x,%d | blockDim.x %d | blockIdx.x %d | threadIdx.x %d\n",gridDim.x,blockDim.x,blockIdx.x,threadIdx.x);
    printf("%d, Hello World from GPU!,blockIdx: %d,threadIdx:%d \n", i,blockIdx.x, threadIdx.x);
    }
}



int main(int argc, char **argv)
{
    printf("Hello World from CPU!\n");

    helloFromGPU<<<5, 10>>>(50);

    CHECK(cudaDeviceReset());
    return 0;
}
```

2. 了解cuda编程的主要步骤

- 1. Allocate GPU memories.
- 2. Copy data from CPU memory to GPU memory.
- 3. Invoke the CUDA kernel to perform program-specific computation.
- 4. Copy data back from GPU memory to CPU memory.
- 5. Destroy GPU memories.

3. 会用cmake来编译cuda程序

```
cmake_minimum_required(VERSION 2.8)
find_package(CUDA QUIET REQUIRED)
#=============================================
# Pass options to NVCC
set( CUDA_NVCC_FLAGS ${CUDA_NVCC_FLAGS};
-O3 -gencode arch=compute_50,code=sm_50
    -gencode arch=compute_60,code=sm_60
)
# Specify include directories
#=============================================

include_directories(
../common
)

#=============================================
# Specify library paths
#link_directories(
#)

#=============================================
# For compilation ...
# Specify target & source files to compile it from
cuda_add_executable(
hello
hello.cu
../common/common.h
)
#=============================================
# For linking ...
# Specify target & libraries to link it with
```

```
# target_link_libraries(
# )
#============================================
```

2gridBlock

目标:学会如何定义grid和Block

- 1.明确概念:

  grid中定义一个grid中有几个block，block 中定义一个block有几个thread

- 2.用dim3来定义block和grid

```
dim3 block (1024);
dim3 grid  ((nElem + block.x - 1) / block.x);
```

- 3.用grid.x来访问block的index，用block.x来访问thread的index
- 4.程序

```
#include "../common/common.h"
#include <cuda_runtime.h>
#include <stdio.h>

/*
 * Demonstrate defining the dimensions of a block of threads and a grid of
 * blocks from the host.
 */

int main(int argc, char **argv)
{
    // define total data element
    int nElem = 1024;

    // define grid and block structure
    dim3 block (1024);
    dim3 grid  ((nElem + block.x - 1) / block.x);
    printf("grid.x %d block.x %d \n", grid.x, block.x);

    // reset block
    block.x = 512;
    grid.x  = (nElem + block.x - 1) / block.x;
    printf("grid.x %d block.x %d \n", grid.x, block.x);

    // reset block
    block.x = 256;
    grid.x  = (nElem + block.x - 1) / block.x;
    printf("grid.x %d block.x %d \n", grid.x, block.x);

    // reset block
    block.x = 128;
    grid.x  = (nElem + block.x - 1) / block.x;
    printf("grid.x %d block.x %d \n", grid.x, block.x);

    // reset device before you leave
    CHECK(cudaDeviceReset());

    return(0);
}
```

3checkDimension从Host,Device端访问维度

- 1.如何从Host端访问grid和block
  - grid(grid.x,grid.y, grid.z)
  - block(block.x,block.y,block,z)
- 2.如何从Device端访问grid和block
  - gridDim(gridDim.x, gridDim.y, gridDim.z)
  - blockDim(blockDim.x, blockDim.y, blockDim.z)
  - blockIdx.x, …
  - threadIdx.x, …
- 3.例子

```
#include "../common/common.h"
#include <cuda_runtime.h>
```

```
#include <stdio.h>

/*
 * Display the dimensionality of a thread block and grid from the host and
 * device.
 */

__global__ void checkIndex(void)
{
    printf("threadIdx:(%d, %d, %d)\n", threadIdx.x, threadIdx.y, threadIdx.z);
    printf("blockIdx:(%d, %d, %d)\n", blockIdx.x, blockIdx.y, blockIdx.z);

    printf("blockDim:(%d, %d, %d)\n", blockDim.x, blockDim.y, blockDim.z);
    printf("gridDim:(%d, %d, %d)\n", gridDim.x, gridDim.y, gridDim.z);

}

int main(int argc, char **argv)
{
    // define total data element
    int nElem = 6;

    // define grid and block structure
    dim3 block(3);
    dim3 grid((nElem + block.x - 1) / block.x);

    // check grid and block dimension from host side
    printf("grid.x %d grid.y %d grid.z %d\n", grid.x, grid.y, grid.z);
    printf("block.x %d block.y %d block.z %d\n", block.x, block.y, block.z);

    // check grid and block dimension from device side
    checkIndex<<<grid, block>>>();

    // reset device before you leave
    CHECK(cudaDeviceReset());

    return(0);
}
```

4checkThreadIdx

通过这个程序掌握基本的cuda编程模型

-1. 分配cpu内存

-2. 分配gpu内存

-3. copy 数据从host到device

-4. host端launch cuda kernel 对数据进行计算

-5. 将结果从device端copy到host端(可省略,数据验证时会用,我是这么理解的)

-6. 释放gpu内存

-7. 释放cpu内存

例子

```
#include "../common/common.h"
#include <cuda_runtime.h>
#include <stdio.h>

/*
 * This example helps to visualize the relationship between thread/block IDs and
 * offsets into data. For each CUDA thread, this example displays the
 * intra-block thread ID, the inter-block block ID, the global coordinate of a
 * thread, the calculated offset into input data, and the input data at that
 * offset.
 */

void printMatrix(int *C, const int nx, const int ny)
{
    int *ic = C;
    printf("\nMatrix: (%d.%d)\n", nx, ny);

    for (int iy = 0; iy < ny; iy++)
    {
        for (int ix = 0; ix < nx; ix++)
        {
            printf("%3d", ic[ix]);

        }

        ic += nx;
        printf("\n");
```

```
    }

    printf("\n");
    return;
}

__global__ void printThreadIndex(int *A, const int nx, const int ny)
{
    int ix = threadIdx.x + blockIdx.x * blockDim.x;
    int iy = threadIdx.y + blockIdx.y * blockDim.y;
    unsigned int idx = iy * nx + ix;

    printf("thread_id (%d,%d) block_id (%d,%d) coordinate (%d,%d) global index"
           " %2d ival %2d\n", threadIdx.x, threadIdx.y, blockIdx.x, blockIdx.y,
           ix, iy, idx, A[idx]);
}

int main(int argc, char **argv)
{
    printf("%s Starting...\n", argv[0]);

    // get device information
    int dev = 0;
    cudaDeviceProp deviceProp;
    CHECK(cudaGetDeviceProperties(&deviceProp, dev));
    printf("Using Device %d: %s\n", dev, deviceProp.name);
    CHECK(cudaSetDevice(dev));

    // set matrix dimension
    int nx = 8;
    int ny = 6;
    int nxy = nx * ny;
    int nBytes = nxy * sizeof(float);

    // malloc host memory
    int *h_A;
    h_A = (int *)malloc(nBytes);

    // iniitialize host matrix with integer
    for (int i = 0; i < nxy; i++)
    {
        h_A[i] = i;
    }
    printMatrix(h_A, nx, ny);

    // malloc device memory
    int *d_MatA;
    CHECK(cudaMalloc((void **)&d_MatA, nBytes));

    // transfer data from host to device
    CHECK(cudaMemcpy(d_MatA, h_A, nBytes, cudaMemcpyHostToDevice));

    // set up execution configuration
    dim3 block(4, 2);
    dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);
    printf("block.x block.y %d,%d\n", block.x,block.y);
    printf("grid.x grid.y %d,%d\n", grid.x,grid.y);

    // invoke the kernel
    printThreadIndex<<<grid, block>>>(d_MatA, nx, ny);
    CHECK(cudaGetLastError());

    // free host and devide memory
    CHECK(cudaFree(d_MatA));
    free(h_A);

    // reset device
    CHECK(cudaDeviceReset());

    return (0);
}
```

5checkYourCudaResults

通过比较cpu结果与gpu结果来检查我们的gpu运算结果是否正确.

```
#include "../common/common.h"
#include <cuda_runtime.h>
#include <stdio.h>

/*
 * This example demonstrates a simple vector sum on the GPU and on the host.
```

```
 * sumArraysOnGPU splits the work of the vector sum across CUDA threads on the
 * GPU. Only a single thread block is used in this small case, for simplicity.
 * sumArraysOnHost sequentially iterates through vector elements on the host.
 */

void checkResult(float *hostRef, float *gpuRef, const int N)
{
    double epsilon = 1.0E-8;
    bool match = 1;

    for (int i = 0; i < N; i++)
    {
        if (abs(hostRef[i] - gpuRef[i]) > epsilon)
        {
            match = 0;
            printf("Arrays do not match!\n");
            printf("host %5.2f gpu %5.2f at current %d\n", hostRef[i],
                   gpuRef[i], i);
            break;
        }
    }

    if (match) printf("Arrays match.\n\n");

    return;
}


void initialData(float *ip, int size)
{
    // generate different seed for random number
    time_t t;
    srand((unsigned) time(&t));

    for (int i = 0; i < size; i++)
    {
        ip[i] = (float)(rand() & 0xFF) / 10.0f;
    }

    return;
}


void sumArraysOnHost(float *A, float *B, float *C, const int N)
{
    for (int idx = 0; idx < N; idx++)
        C[idx] = A[idx] + B[idx];
}

__global__ void sumArraysOnGPU(float *A, float *B, float *C, const int N)
{
    int i = threadIdx.x;

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


int main(int argc, char **argv)
{
    printf("%s Starting...\n", argv[0]);

    // set up device
    int dev = 0;
    CHECK(cudaSetDevice(dev));

    // set up data size of vectors
    int nElem = 1 << 5;
    printf("Vector size %d\n", nElem);

    // malloc host memory
    size_t nBytes = nElem * sizeof(float);

    float *h_A, *h_B, *hostRef, *gpuRef;
    h_A     = (float *)malloc(nBytes);
    h_B     = (float *)malloc(nBytes);
    hostRef = (float *)malloc(nBytes);
    gpuRef  = (float *)malloc(nBytes);

    // initialize data at host side
    initialData(h_A, nElem);
    initialData(h_B, nElem);

    memset(hostRef, 0, nBytes);
```

```
    memset(gpuRef,  0, nBytes);

    // malloc device global memory
    float *d_A, *d_B, *d_C;
    CHECK(cudaMalloc((float**)&d_A, nBytes));
    CHECK(cudaMalloc((float**)&d_B, nBytes));
    CHECK(cudaMalloc((float**)&d_C, nBytes));

    // transfer data from host to device
    CHECK(cudaMemcpy(d_A, h_A, nBytes, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_B, h_B, nBytes, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_C, gpuRef, nBytes, cudaMemcpyHostToDevice));

    // invoke kernel at host side
    dim3 block (nElem);
    dim3 grid  (1);

    sumArraysOnGPU<<<grid, block>>>(d_A, d_B, d_C, nElem);
    printf("Execution configure <<<%d, %d>>>\n", grid.x, block.x);

    // copy kernel result back to host side
    CHECK(cudaMemcpy(gpuRef, d_C, nBytes, cudaMemcpyDeviceToHost));

    // add vector at host side for result checks
    sumArraysOnHost(h_A, h_B, hostRef, nElem);

    // check device results
    checkResult(hostRef, gpuRef, nElem);

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

    // free host memory
    free(h_A);
    free(h_B);
    free(hostRef);
    free(gpuRef);

    CHECK(cudaDeviceReset());
    return(0);
}
```

6TimingYourKernel测试你的kernel性能

- 1. 用sys/time.h的函数

CPU timer can be created by using the gettimeofday system call to get the system’s wall-clock time, which returns the number of seconds since the epoch. You need to include the sys/time.h header file

```
include <sys/time.h>
inline double seconds()
{
    struct timeval tp;
    struct timezone tzp;
    int i = gettimeofday(&tp, &tzp);
    return ((double)tp.tv_sec + (double)tp.tv_usec * 1.e-6);
}
```

- 2. 用nvprof 命令行来测试cuda API所好用的时间

```
nvprof ./可执行文件
```

```
nvprof o./oumArraysOnGPU-timer

NVPROF is profiling process 12644, command: ./sumArraysOnGPU-timer
Using Device 0: GeForce GTX 1080
Vector size 16777216
initialData Time elapsed 0.604351 sec
sumArraysOnHost Time elapsed 0.012240 sec
sumArraysOnGPU <<<  32768, 512  >>>  Time elapsed 0.000959 sec
Arrays match.

==12644== Profiling application: ./sumArraysOnGPU-timer
==12644== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
73.05%  23.592ms         3  7.8640ms  7.7391ms  7.9720ms  [CUDA memcpy HtoD]
24.38%  7.8736ms         1  7.8736ms  7.8736ms  7.8736ms  [CUDA memcpy DtoH]
```

```
2.56%  828.31us         1  828.31us  828.31us  828.31us  sumArraysOnGPU(float*, float*, float*, int)

==12644== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
89.62%  323.27ms         3  107.76ms  252.03us  322.75ms  cudaMalloc
8.83%  31.856ms         4  7.9639ms  7.8661ms  8.0520ms  cudaMemcpy
1.07%  3.8497ms         3  1.2832ms  182.49us  1.8400ms  cudaFree
0.26%  923.12us         1  923.12us  923.12us  923.12us  cudaDeviceSynchronize
0.09%  312.99us        91  3.4390us     105ns  138.29us  cuDeviceGetAttribute
0.08%  305.22us         1  305.22us  305.22us  305.22us  cudaGetDeviceProperties
0.03%  110.51us         1  110.51us  110.51us  110.51us  cuDeviceTotalMem
0.01%  34.371us         1  34.371us  34.371us  34.371us  cuDeviceGetName
0.01%  28.581us         1  28.581us  28.581us  28.581us  cudaLaunch
0.00%  5.7500us         1  5.7500us  5.7500us  5.7500us  cudaSetDevice
0.00%  1.7300us         3     576ns     133ns  1.3610us  cuDeviceGetCount
0.00%  1.5580us         1  1.5580us  1.5580us  1.5580us  cudaConfigureCall
0.00%  1.2100us         4     302ns     149ns     521ns  cudaSetupArgument
0.00%  1.1950us         3     398ns     109ns     885ns  cuDeviceGet
0.00%     360ns         1     360ns     360ns     360ns  cudaGetLastError
```

- 3. 例子

```
#include "../common/common.h"
#include <cuda_runtime.h>
#include <stdio.h>

/*
 * This example demonstrates a simple vector sum on the GPU and on the host.
 * sumArraysOnGPU splits the work of the vector sum across CUDA threads on the
 * GPU. Only a single thread block is used in this small case, for simplicity.
 * sumArraysOnHost sequentially iterates through vector elements on the host.
 * This version of sumArrays adds host timers to measure GPU and CPU
 * performance.
 */

 void checkResult(float *hostRef, float *gpuRef, const int N)
 {
   double epsilon = 1.0E-8;
   bool match = 1;

   for (int i = 0; i < N; i++)
   {
     if (abs(hostRef[i] - gpuRef[i]) > epsilon)
     {
       match = 0;
       printf("Arrays do not match!\n");
       printf("host %5.2f gpu %5.2f at current %d\n", hostRef[i],
           gpuRef[i], i);
       break;
     }
   }

   if (match) printf("Arrays match.\n\n");

   return;
 }

 void initialData(float *ip, int size)
 {
   // generate different seed for random number
   time_t t;
   srand((unsigned) time(&t));

   for (int i = 0; i < size; i++)
   {
     ip[i] = (float)( rand() & 0xFF ) / 10.0f;
   }

   return;
 }

 void sumArraysOnHost(float *A, float *B, float *C, const int N)
 {
   for (int idx = 0; idx < N; idx++)
   {
     C[idx] = A[idx] + B[idx];
   }
 }
 __global__ void sumArraysOnGPU(float *A, float *B, float *C, const int N)
 {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
```

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

int main(int argc, char **argv)
{
    printf("%s Starting...\n", argv[0]);

    // set up device
    int dev = 0;
    cudaDeviceProp deviceProp;
    CHECK(cudaGetDeviceProperties(&deviceProp, dev));
    printf("Using Device %d: %s\n", dev, deviceProp.name);
    CHECK(cudaSetDevice(dev));

    // set up data size of vectors
    int nElem = 1 << 24;
    printf("Vector size %d\n", nElem);

    // malloc host memory
    size_t nBytes = nElem * sizeof(float);

    float *h_A, *h_B, *hostRef, *gpuRef;
    h_A     = (float *)malloc(nBytes);
    h_B     = (float *)malloc(nBytes);
    hostRef = (float *)malloc(nBytes);
    gpuRef  = (float *)malloc(nBytes);

    double iStart, iElaps;

    // initialize data at host side
    iStart = seconds();
    initialData(h_A, nElem);
    initialData(h_B, nElem);
    iElaps = seconds() - iStart;
    printf("initialData Time elapsed %f sec\n", iElaps);
    memset(hostRef, 0, nBytes);
    memset(gpuRef,  0, nBytes);

    // add vector at host side for result checks
    iStart = seconds();
    sumArraysOnHost(h_A, h_B, hostRef, nElem);
    iElaps = seconds() - iStart;
    printf("sumArraysOnHost Time elapsed %f sec\n", iElaps);

    // malloc device global memory
    float *d_A, *d_B, *d_C;
    CHECK(cudaMalloc((float**)&d_A, nBytes));
    CHECK(cudaMalloc((float**)&d_B, nBytes));
    CHECK(cudaMalloc((float**)&d_C, nBytes));

    // transfer data from host to device
    CHECK(cudaMemcpy(d_A, h_A, nBytes, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_B, h_B, nBytes, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_C, gpuRef, nBytes, cudaMemcpyHostToDevice));

    // invoke kernel at host side
    int iLen = 512;
    dim3 block (iLen);
    dim3 grid  ((nElem + block.x - 1) / block.x);

    iStart = seconds();
    sumArraysOnGPU<<<grid, block>>>(d_A, d_B, d_C, nElem);
    CHECK(cudaDeviceSynchronize());
    iElaps = seconds() - iStart;
    printf("sumArraysOnGPU <<<  %d, %d  >>>  Time elapsed %f sec\n", grid.x,
        block.x, iElaps);

    // check kernel error
    CHECK(cudaGetLastError()) ;

    // copy kernel result back to host side
    CHECK(cudaMemcpy(gpuRef, d_C, nBytes, cudaMemcpyDeviceToHost));

    // check device results
    checkResult(hostRef, gpuRef, nElem);

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

    // free host memory
```

```
    free(h_A);
    free(h_B);
    free(hostRef);
    free(gpuRef);

    return(0);
}
```

Indices and tables

- genindex
- modindex
- search