Extra Credit Partial Sum

George Onwubuya

October 6, 2018

1 Main

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Partial_Sum_Kernel.cu"

#define BLOCK_SIZE  32;
#define SAMPLE_SIZE 32

void FATAL (const char * s )
{
        puts(s);
        exit(1);
}

int main(int argc, char**argv) {

    unsigned int data_size;
    cudaError_t cuda_ret;


    if(argc == 1) {
        data_size= 64;
    } else if(argc == 2) {
        data_size= atoi(argv[1]);
    } else {
        printf("\n    Invalid input parameters!"
           "\n    Usage: ./vecadd               # Vector of size 10,000 is used"
           "\n    Usage: ./vecadd <m>           # Vector of size m is used"
           "\n");
        exit(0);
    }

//Device data
int * array_dev;
int array_size = data_size;

//Host data
int * array_host = (int *) malloc (sizeof(int)*array_size);
for(int i = 0; i < data_size; ++i)
        array_host[i] = i + 1;

for (int i = data_size; i < array_size; ++i)
        array_host[i] = 0;

int expected_sum = data_size * (array_host[0] + array_host[data_size - 1]) / 2;

//Allocating & copying device memory
cuda_ret = cudaMalloc((void**)&array_dev, array_size*sizeof(int));
        if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
cuda_ret = cudaMemcpy(array_dev, array_host, array_size*sizeof(int), cudaMemcpyHostToDevice);
        if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to device");

cudaDeviceSynchronize();

//Invoke Kernel}
vecSum_final_int1<<<dim3(1, 1, 1), dim3(SAMPLE_SIZE, 1, 1)>>>(array_dev);
vecSum_final_int<<<dim3(1, 1, 1), dim3(SAMPLE_SIZE, 1, 1)>>>(array_dev);

//Copying to host memory
int *result = (int *) malloc(sizeof(int)*array_size);
cuda_ret = cudaMemcpy(result, array_dev, sizeof(int)*array_size, cudaMemcpyDeviceToHost);
if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to host");

cudaDeviceSynchronize();

printf("Array size = %d\n", array_size);
printf("Expected result = %d\n", expected_sum);
printf("Calculated result = %d\n", result[0]);
```

```

for (int i = 0; i < data_size; ++i){
        printf("[%2d] : %5d, %5d\n", i, array_host[i], result[i]);
}


fflush(stdout);

free(array_host);
cudaFree(array_dev);

return 0;

};
```

2 Kernel

```
__global__ void vecSum_final_int(int * array)
{


  for(unsigned int offset = blockDim.x; offset  > 0; offset = offset >> 1){
      __syncthreads();

      if (threadIdx.x < offset)
          array[threadIdx.x] += array[threadIdx.x + offset];
  }
}

__global__ void vecSum_final_int1(int * array)
{
  const int tidx = threadIdx.x << 1;

  for (unsigned int stride = 1; stride <= blockDim.x; stride = stride << 1 ){

      __syncthreads();

      if(tidx % stride == 0)
         array[tidx] += array[tidx + stride];

 }
}
```

3 Output

```
==28325==NVPROF is profiling process 28325,command:/home/onwubuyag/Partial_Sum/partial_sum
==28325==Profiling application:/home/onwubuyag/Partial_Sum/partial_sum
==28325==Profiling result:
Type  Time(%)  Time         Calls  Avg       Min       Max      Name
GPU: 50.92%  8.8960us         1  8.8960us  8.8960us  8.8960us   vecSum_final_int1(int*)
     27.29%  4.7680us         1  4.7680us  4.7680us  4.7680us   vecSum_final_int(int*)
     13.19%  2.3040us         1  2.3040us  2.3040us  2.3040us   [CUDA memcpy DtoH]
      8.60%  1.5030us         1  1.5030us  1.5030us  1.5030us   [CUDA memcpy HtoD]
API: 99.46%  145.53ms         1  145.53ms  145.53ms  145.53ms   cudaMalloc
      0.15%  219.99us         2  109.99us  15.702us  204.29us   cudaLaunch
      0.12%  181.70us        94  1.9330us     452ns  47.434us   cuDeviceGetAttribute
      0.11%  167.89us         1  167.89us  167.89us  167.89us   cuDeviceTotalMem
      0.08%  112.70us         1  112.70us  112.70us  112.70us   cudaFree
      0.04%  64.552us         2  32.276us  28.324us  36.228us   cudaMemcpy
      0.01%  15.640us         1  15.640us  15.640us  15.640us   cuDeviceGetName
      0.01%  15.548us         2  7.7740us  6.0780us  9.4700us   cudaDeviceSynchronize
      0.00%  6.1690us         3  2.0560us     565ns  4.2580us   cuDeviceGetCount
      0.00%  6.1590us         2  3.0790us     765ns  5.3940us   cudaSetupArgument
      0.00%  3.3440us         2  1.6720us  1.0260us  2.3180us   cudaConfigureCall
      0.00%  1.9730us         2     986ns     627ns  1.3460us   cuDeviceGet
```

4 Output Analysis

The output shows that the ”vecSum final int” fucntion has a faster execution time than the ”vecSum final int1” function. This is down to the fact that the latter function reduces coalescing. The ”vecSum final int1” has an execution time of 8.8960us and ”vecSum final int1” has an execution time of 4.7680us.