```cpp
/*
  G p u T e s t . c u
*/


#include <iostream>

#include <stdio.h>
#include <stddef.h>

#include <cuda.h>
#include "cuda_runtime.h"

#include "../Library/ReduceAdd.h"
#include "GridHelper.cuh"
#include "EarlyTerm.cuh"

using namespace std;

// ****************************************************************************
void Ok(cudaError_t status, char* message)
{
  if (status != cudaSuccess) {
    printf(message);
    abort();
  }
}

// ****************************************************************************
template<typename ElemT>
  __global__ void WarmingUp(ElemT* partSum, ElemT* data, unsigned dataSize)
{
  unsigned numBlock = blockDim.x;
  int tid = blockIdx.x * numBlock + threadIdx.x;

```

```
36    partSum[tid % numBlock] = tid;
37  }
38
39  // ********************************************************************
40  template<typename ElemT>
41    void ReduceAddGpu(
42      ElemT& result, const ElemT* data, size_t numElem, unsigned threadPerBlock)
43  {
44    ElemT* data_d = NULL;
45    ElemT* partSum_d = NULL;
46
47    // Choose which GPU to run on, change this on a multi-GPU system.
48    Ok(cudaSetDevice(0), "No cuda devices.");
49
50    // Compute the grid size
51    cudaDeviceProp devProp;
52    Ok(cudaGetDeviceProperties(&devProp, 0), "Can't get device properties");
53
54    unsigned numThread = (numElem - 1) / 2 + 1;
55    unsigned numBlock = (numThread - 1) / threadPerBlock + 1;
56
57    // Allocate GPU buffers for data and partSum
58    const size_t dataBytes = numElem * sizeof(ElemT);
59    Ok(cudaMalloc((void**)&data_d, dataBytes), "Data allocation failed");
60
61    const size_t resultBytes = numBlock * sizeof(ElemT);
62    Ok(cudaMalloc((void**)&partSum_d, resultBytes), "PartSum allocaiton failed");
63
64    // Copy input vectors from host memory to GPU buffers.
65    Ok(cudaMemcpy(data_d, data, dataBytes, cudaMemcpyHostToDevice), "Copying data failed");
66
67    // Create timmers
68    cudaEvent_t preWarm, middle, postReduce;
69    Ok(cudaEventCreate(&preWarm), "Creation of PreWarm event failed");
70    Ok(cudaEventCreate(&middle), "Creation of Midle event failed");
```

```
 71    Ok(cudaEventCreate(&postReduce), "Creation of PostReduce event failed");
 72
 73    // *********************************
 74    // Do warmup
 75    Ok(cudaEventRecord(preWarm), "Recording PreWarm event failed");
 76    WarmingUp <<< numBlock, threadPerBlock >>> (partSum_d, data_d, numElem);
 77
 78    // Check for any errors launching the kernel
 79    cudaError_t cudaStatus = cudaGetLastError();
 80    if (cudaStatus != cudaSuccess) {
 81      fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
 82      abort();
 83    }
 84
 85    // waits for the kernel to finish
 86    cudaStatus = cudaDeviceSynchronize();
 87    if (cudaStatus != cudaSuccess) {
 88      fprintf(stderr, "cudaDeviceSynchronize returned error code %d\n", cudaStatus);
 89      abort();
 90    }
 91
 92    // compute elapsed time
 93    Ok(cudaEventRecord(middle), "Recording middle event failed");
 94
 95    float warmTime;
 96    Ok(cudaEventElapsedTime(&warmTime, preWarm, middle), "Warmup time failed.");
 97    warmTime *= 1e-3;
 98
 99    // *********************************
100    // Do Add Reduce
101    AddReduceEarlyTerm << < numBlock, threadPerBlock >> > (partSum_d, data_d, numElem);
102
103    // Check for any errors launching the kernel
104    cudaStatus = cudaGetLastError();
105    if (cudaStatus != cudaSuccess) {
```

```
106       fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
107       abort();
108     }
109
110     // cudaDeviceSynchronize waits for the kernel to finish, and returns
111     // any errors encountered during the launch.
112     cudaStatus = cudaDeviceSynchronize();
113     if (cudaStatus != cudaSuccess) {
114       fprintf(stderr, "cudaDeviceSynchronize returned error code %d\n", cudaStatus);
115       abort();
116     }
117
118     // Deal with time
119     Ok(cudaEventRecord(postReduce), "Recording Stop event failed");
120
121     float reduceTime;
122     Ok(cudaEventElapsedTime(&reduceTime, middle, postReduce),"reduce time feaild");
123     reduceTime *= 1e-3;
124
125     cout << numElem << ", " << threadPerBlock << ", " << warmTime << ", " << reduceTime << '\n';
126
127     // Copy output vector from GPU buffer to host memory.
128     ElemT* partSum = new ElemT[numBlock];
129     Ok(
130       cudaMemcpy(partSum, partSum_d, resultBytes, cudaMemcpyDeviceToHost),
131       "Copy of PartSum failed");
132
133     result = ReduceAdd(partSum, numBlock);
134     delete[] partSum;
135
136     // Clean up
137     if (data_d != NULL)
138       cudaFree(data_d);
139     if (partSum_d != NULL)
140       cudaFree(partSum_d);
```

```
141 }
142
143 // ***********************************
144 // Actually create something to like to
145 template void ReduceAddGpu<int>(
146   int& result, const int* data, size_t numElem, unsigned threadPerBlock);
```