List Reduction Grade

Grade Summary (History) (/grade/history/5984)

Created:	less than a minute ago (2022-04-11 03:16:06 +0000 UTC)
Total Score:	100 out of 100 points
Coding Score:	100 out of 100 points
Questions Score:	0

Program Code

```
// MP Reduction
// Given a list (lst) of length n
// Output its sum = lst[0] + lst[1] + ... + lst[n-1];

#include <wb.h>

#define BLOCK_SIZE 512 //@@ You can change this

#define wbCheck(stmt)
  do {
    cudaError_t err = stmt;
    if (err != cudaSuccess) {
      wbLog(ERROR, "Failed to run stmt ", #stmt);
      wbLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));
      return -1;
    }
  } while (0)

__global__ void reduction(float *input, float *output, int len) {
  //@@ Load a segment of the input vector into shared memory
  __shared__ float sum[BLOCK_SIZE * 2];

  int thread_x  = threadIdx.x;

  int load_in = 2*(BLOCK_SIZE * blockIdx.x) + thread_x;

  sum[thread_x] = load_in < len ? input[load_in] : 0.0;
  if (load_in + BLOCK_SIZE < len)
      sum[BLOCK_SIZE + thread_x] = input[load_in + BLOCK_SIZE];
  else
      sum[BLOCK_SIZE + thread_x] = 0.0;

  //@@ Traverse the reduction tree
  for (int travel_tree = BLOCK_SIZE; travel_tree >= 1; travel_tree >>= 1)
      __syncthreads();
      if (thread_x < travel_tree)
          sum[thread_x] += sum[thread_x + travel_tree];
  }

  //@@ Write the computed sum of the block to the output vector at the
  //@@ correct index
  if (thread_x == 0){
    output[blockIdx.x] = sum[0];
  }
}

int main(int argc, char **argv) {
  int ii;
  wbArg_t args;
  float *hostInput;  // The input 1D list
  float *hostOutput; // The output list
  float *deviceInput;
  float *deviceOutput;
  int numInputElements;  // number of elements in the input list
  int numOutputElements; // number of elements in the output list

  args = wbArg_read(argc, argv);

  wbTime_start(Generic, "Importing data and creating memory on host");
  hostInput = (float *)wbImport(wbArg_getInputFile(args, 0), &numInputEle

  numOutputElements = numInputElements / (BLOCK_SIZE << 1);
  if (numInputElements % (BLOCK_SIZE << 1)){
    numOutputElements++;
  }

  //2 variables to help with sizes
  int Size_of_Input = numInputElements * sizeof(float);
  int Size_of_Output = numOutputElements * sizeof(float);

  hostOutput = (float *)malloc(numOutputElements * sizeof(float));

  wbTime_stop(Generic, "Importing data and creating memory on host");

  wbLog(TRACE, "The number of input elements in the input is ", numInputE
  wbLog(TRACE, "The number of output elements in the input is ", numOutpu

  wbTime_start(GPU, "Allocating GPU memory.");
  //@@ Allocate GPU memory here
  cudaMalloc(&deviceInput, Size_of_Input);
  cudaMalloc(&deviceOutput, Size_of_Output);

  wbTime_stop(GPU, "Allocating GPU memory.");

  wbTime_start(GPU, "Copying input memory to the GPU.");
  //@@ Copy memory to the GPU here
  cudaMemcpy(deviceInput, hostInput, Size_of_Input, cudaMemcpyHostToDevic

  wbTime_stop(GPU, "Copying input memory to the GPU.");
  //@@ Initialize the grid and block dimensions here
  dim3 dimGrid(numOutputElements, 1, 1);
  dim3 dimBlock(BLOCK_SIZE, 1, 1);

  wbTime_start(Compute, "Performing CUDA computation");
  //@@ Launch the GPU Kernel here
  reduction<<<dimGrid, dimBlock>>>(deviceInput, deviceOutput, numInputEle

  cudaDeviceSynchronize();
  wbTime_stop(Compute, "Performing CUDA computation");

  wbTime_start(Copy, "Copying output memory to the CPU");
  //@@ Copy the GPU memory back to the CPU here
  cudaMemcpy(hostOutput, deviceOutput, Size_of_Output, cudaMemcpyDeviceTo

  wbTime_stop(Copy, "Copying output memory to the CPU");

  /********************************************************************
   * Reduce output vector on the host
   * NOTE: One could also perform the reduction of the output vector
   * recursively and support any size input. For simplicity, we do not
   * require that for this lab.
   ********************************************************************/
  for (ii = 1; ii < numOutputElements; ii++) {
    hostOutput[0] += hostOutput[ii];
  }

  wbTime_start(GPU, "Freeing GPU Memory");
  //@@ Free the GPU memory here
  cudaFree(deviceInput);
  cudaFree(deviceOutput);

  wbTime_stop(GPU, "Freeing GPU Memory");

  wbSolution(args, hostOutput, 1);

  free(hostInput);
  free(hostOutput);

  return 0;
}

```

Designed and architected by Abdul Dakkak (https://www.dakkak.dev/).