Кафедра прикладной математики и кибернетики

Выполнили:

Студенты 3 курса группы ИП-111  
Корнилов А.А.,  
Попов М.И.,

Толкач А.А.

Проверил:

Профессор кафедры ПМиК  
Малков Е.А.

ЛАБОРАТОРНАЯ РАБОТА №7

По дисциплине: «Программирование графических процессоров»

Сибирский государственный университет  
телекоммуникаций и информатики

Новосибирск, 2024

Министерство цифрового развития, связи  
и массовых коммуникаций Российской Федерации

**Задание:**

* Реализовать вычисление скалярного произведения векторов на GPU, используя CUDA API (“сырой код”) и, отдельно, используя библиотеку Thrust. Сравнить время выполнения программ при различной длине векторов.
* Реализовать транспонирование матрицы на GPU, используя CUDA API (“сырой код”) и, отдельно, используя библиотеку Thrust. Сравнить время выполнения программ при различной размерности матрицы.

**Цель:** освоить использование библиотеки Thrust.

**Оборудование**: Видеокарта GTX 1050TI (Pascal)

**Выполнение работы:**

Для выполнения работы был взят фрагмент программы с предыдущей лабораторной работы с сложением вектора размера (N\*2^12)\*(K\*2^12), и из библиотеки Thrust был использован метод Transform для выбора матриц их начала и конца и оператор multiplies для их перемножения:

|  |
| --- |
| #include <thrust/generate.h> #include <thrust/gather.h> #include <thrust/device\_vector.h> #include <thrust/host\_vector.h> #include <cstdlib> #include <iostream> #include <iomanip>  using namespace std;  **\_\_global\_\_** void gFunc(int \*A, int \*B, int \*C, int N) {  unsigned int i = threadIdx.x + blockIdx.x \* blockDim.x;  if (i >= N) return;  C[i] = A[i] + B[i]; }  int main() {  const int num = 1 << 12;  int N = 4 \* num, K = 8 \* num, threads\_per\_block = 128;  float elapsedTime = 0;  cudaEvent\_t start, stop;   int \*hA, \*hB, \*hC;  int \*A = (int \*) calloc(N \* K, sizeof(int));  int \*B = (int \*) calloc(N \* K, sizeof(int));  int \*C = (int \*) calloc(N \* K, sizeof(int));   cudaMalloc((void \*\*) &hA, N \* K \* sizeof(int));  cudaMalloc((void \*\*) &hB, N \* K \* sizeof(int));  cudaMalloc((void \*\*) &hC, N \* K \* sizeof(int));   for (int i = 0; i < N; ++i) {  A[i] = i;  B[i] = i + 1;  }   cudaMemcpy(hA, A, N \* K \* sizeof(int), cudaMemcpyHostToDevice);  cudaMemcpy(hB, B, N \* K \* sizeof(int), cudaMemcpyHostToDevice);   cudaEventCreate(&start);  cudaEventCreate(&stop);   cudaEventRecord(start, 0);   gFunc<<<dim3(threads\_per\_block),  dim3((N \* K + threads\_per\_block - 1) / threads\_per\_block)  >>>(hA, hB, hC, N \* K);  cudaDeviceSynchronize();   cudaEventRecord(stop, 0);  cudaEventSynchronize(stop);   cudaEventElapsedTime(&elapsedTime, start, stop);  cout << "Time using raw CUDA code: " << setprecision(15) << elapsedTime << endl;  elapsedTime = 0;   cudaMemcpy(C, hC, N \* K \* sizeof(float), cudaMemcpyDeviceToHost);   cudaFree(hA);  cudaFree(hB);  cudaFree(hC);   thrust::host\_vector<int> vA(A, A + N \* K);  thrust::host\_vector<int> vB(B, B + N \* K);  thrust::host\_vector<int> vC(N \* K);   thrust::device\_vector<int> dA = vA;  thrust::device\_vector<int> dB = vB;  thrust::device\_vector<int> dC(N \* K);   cudaEventRecord(start, 0);  thrust::transform(dA.begin(), dA.end(), dB.begin(), dC.begin(), thrust::multiplies<int>());  cudaDeviceSynchronize();   cudaEventRecord(stop, 0);  cudaEventSynchronize(stop);   cudaEventElapsedTime(&elapsedTime, start, stop);  cout << "Time using Thrust lib: " << setprecision(15) << elapsedTime << endl; } |

Листинг 1 – программа LR07\_1.cu

Результат работы программы:

|  |
| --- |
| D:\Projects\CUDA\_CMake\cmake-build-debug\LR07\_GPU\_1.exe  Time using raw CUDA code: 1.53593599796295  Time using Thrust lib: 175.661056518555  Process finished with exit code 0 |

К сожалению nvprof не получилось применить к программе.

По результату мы видим что программа с использованием библеотеки Thrust работает горазда медленней, возможно из-за отсутствия выбрать количество нитей на блок

|  |  |  |
| --- | --- | --- |
| скалярного произведения векторов | | |
|  | CUDA | Thrust |
| 1 << 2 | 1,234848022 | 0,114688002 |
| 1 << 4 | 1,079167962 | 0,516095996 |
| 1 << 6 | 1,146175981 | 0,091136001 |
| 1 << 8 | 1,253792048 | 0,315232009 |
| 1 << 10 | 2,063231945 | 4,273151875 |
| 1 << 12 | 39,45622253 | 186,9219818 |

Также была собрана программа из прядущих лабораторных работ с транспонированием матрицы, использованием shared памяти и разрешением конфликта банков, и из библиотеки Thrust была взята функция gather для копирования элементов d\_map для транспонирования.

|  |
| --- |
| #include <cuda.h> #include <thrust/host\_vector.h> #include <thrust/device\_vector.h> #include <thrust/copy.h> #include <thrust/gather.h> #include <iomanip>  #define **CUDA\_NUM** 32  using namespace std;  **\_\_global\_\_** void gFunc(int \*A, int N, int K) {  **\_\_shared\_\_** int B[**CUDA\_NUM**][**CUDA\_NUM** + 1];  unsigned int k = threadIdx.x + blockIdx.x \* blockDim.x;  unsigned int n = threadIdx.y + blockIdx.y \* blockDim.y;  if (k >= N || n >= K) return;   B[threadIdx.y][threadIdx.x] = A[n + k \* K];  \_\_syncthreads();  A[k + n \* N] = B[threadIdx.y][threadIdx.x]; }   int main() {  const int num = 1 << 12;  int N = 4 \* num, K = 8 \* num, threads\_per\_block = 128;  float elapsedTime = 0;  cudaEvent\_t start, stop;   int \*map = (int \*) calloc(N \* K, sizeof(int));  int \*A = (int \*) calloc(N \* K, sizeof(int));   int \*hA;  cudaMalloc((void \*\*) &hA, K \* N \* sizeof(int));   for (int i = 0; i < N; ++i) {  for (int j = 0; j < K; ++j) {  A[j + i \* K] = j + i \* K;  }  }   cudaMemcpy(hA, A, N \* K \* sizeof(int), cudaMemcpyHostToDevice);   cudaEventCreate(&start);  cudaEventCreate(&stop);   cudaEventRecord(start, 0);  gFunc<<<dim3(threads\_per\_block, threads\_per\_block),  dim3((N + threads\_per\_block - 1) / threads\_per\_block,(K + threads\_per\_block - 1) / threads\_per\_block)  >>>(hA, N,K);  cudaDeviceSynchronize();   cudaEventRecord(stop, 0);  cudaEventSynchronize(stop);   cudaEventElapsedTime(&elapsedTime, start, stop);  cout << "Time using raw CUDA code: " << setprecision(15) << elapsedTime << endl;  elapsedTime = 0;   cudaMemcpy(A, hA, N \* K \* sizeof(float), cudaMemcpyDeviceToHost);   free(A);  cudaFree(hA);   thrust::host\_vector<int> vA(N \* K);  for (int i = 0; i < N \* K; ++i) {  vA[i] = i;  }  thrust::device\_vector<int> dA = vA;  thrust::device\_vector<int> dA\_T(N \* K);    for (int i = 0; i < K \* N; ++i) {  map[i] = (i % N) \* K + (i / N);  }   thrust::device\_vector<int> d\_map(map, map + K \* N);    cudaEventRecord(start, 0);  thrust::gather(d\_map.begin(), d\_map.end(), dA.begin(), dA\_T.begin());  cudaDeviceSynchronize();   cudaEventRecord(stop, 0);  cudaEventSynchronize(stop);   cudaEventElapsedTime(&elapsedTime, start, stop);  cout << "Time using Thrust lib: " << setprecision(15) << elapsedTime << endl; } |

Результат работы программы:

|  |
| --- |
| D:\Projects\CUDA\_CMake\cmake-build-debug\LR07\_GPU\_2.exe  Time using raw CUDA code: 1.84943997859955  Time using Thrust lib: 599.636596679688  Process finished with exit code 0 |

К сожалению nvprof не получилось применить к программе.

|  |  |  |
| --- | --- | --- |
| транспонирование матрицы | | |
|  | CUDA | Thrust |
| 1 << 2 | 0,975040019 | 0,175104007 |
| 1 << 4 | 0,889855981 | 0,076800004 |
| 1 << 6 | 1,47388804 | 0,111904003 |
| 1 << 8 | 1,601631999 | 0,478655994 |
| 1 << 10 | 2,158911943 | 8,643551826 |
| 1 << 12 | 1,600128055 | 602,6151733 |

Тут также наблюдаем сильное снижение производительности при использовании библиотеки

Вывод: в ходе выполнения лабораторной работы, была исследована и применена работа с библиотекой Thrust, с методами device\_vector для выделения памяти на видеоускорителе, host\_vector для выделение памяти на хосте, метод transform для операция над массивами с указанием начала и конца и оператором multiplies умножением, и также метод gather для копирования. Библиотека thrust создана для облегчения работы вычислений с видеокартой, но имеет заметные отставания по производительности по с равнению с сырым CUDA кодом