Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школы электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ
о лабораторной работе №9
по дисциплине «Технологии параллельного программирования»

Выполнил:
студент группы КЭ-220
_____/Голенищев А. Б.
_____ 2024 г.

Отчет принял:
_____/Жулев А. Э.
_____ 2024 г.

Челябинск
2024

Разработана программа, которая вычисляет число π двумя способами: на CPU (с использованием последовательной функции CalcPi) и на GPU (с использованием параллельного CUDA-ядра CalcPiKernel), листнинг 1. я. Продемонстрирован результат выполнения программы, рисунок 1.

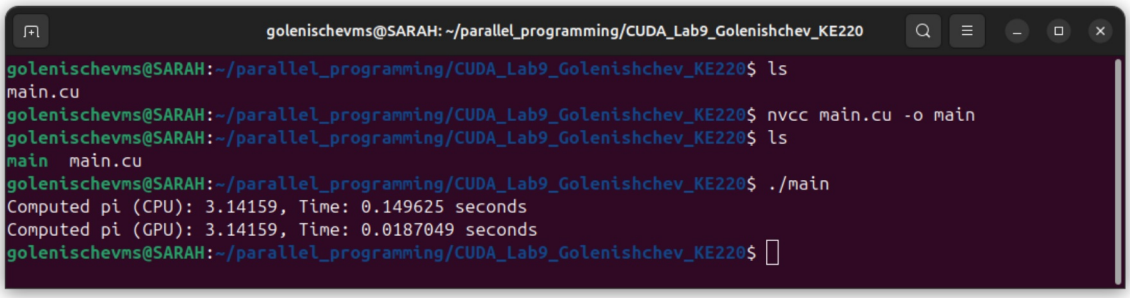

Рисунок 1. Результат вычислений числа π на ЦП и ГП.

```
#include <cuda_runtime.h>
#include <iostream>
#include <chrono>
// Golenishchev Artem, KE-220 Lab 9
int main() {
const int n = 100000000; // Количество интервалов для вычисления π
// CPU вычисление
auto startCPU = std::chrono::high_resolution_clock::now();
double piCPU = CalcPi(n);
auto endCPU = std::chrono::high_resolution_clock::now();
double cpuTime = std::chrono::duration<double>(endCPU - startCPU).count();
std::cout << "Computed pi (CPU): " << piCPU << ", Time: " << cpuTime << " seconds\n";
// GPU вычисление
float gpuTime = 0;
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);
double piGPU = CalcPiGPU(n);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&gpuTime, start, stop);
cudaEventDestroy(start);
cudaEventDestroy(stop);
std::cout << "Computed pi (GPU): " << piGPU << ", Time: " << gpuTime / 1000.0 << " seconds\n";
return 0;
}
```

Листнинг 1. Код первой параллельной программы

В CUDA-ядре каждый поток обрабатывает часть итераций, вычисляет локальную сумму, а затем выполняется редукция в пределах блока с использованием общей памяти. После выполнения всех блоков частичные суммы копируются на хост, где происходит финальная редукци. Преставлен код CUDA-ядра, листнинг 2.

```
/ CUDA ядро для параллельного вычисления суммы
__global__ void calcPiKernel(const int n, double* partialSums) {
extern __shared__ double sharedSums[]; // Shared memory для частичных сумм
int idx = blockIdx.x * blockDim.x + threadIdx.x;
int tid = threadIdx.x;
const double coef = 1.0 / n;
sharedSums[tid] = 0.0;
if (idx < n) {
const double xi = (idx + 0.5) * coef;
sharedSums[tid] = 4.0 / (1.0 + xi * xi);
}
__syncthreads();
// Редукция внутри блока
for (int stride = blockDim.x / 2; stride > 0; stride /= 2) {
if (tid < stride) {
sharedSums[tid] += sharedSums[tid + stride];
}
__syncthreads();
}
// Запись результата блока в глобальную память
if (tid == 0) {
partialSums[blockIdx.x] = sharedSums[0];
}
}
```

Листнинг 2. Реализация функций для CPU и GPU, реализация CUDA-ядра

В функции CalcPiGPU используется CUDA-ядро calcPiKernel, которое выполняет параллельное вычисление суммы элементов для нахождения числа π. Каждая нить вычисляет вклад одного интервала в сумму, используя $4/(1 + x^2)$, где x определяется исходя из индекса нити. Частичные результаты каждого блока сохраняются в shared memory для ускорения редукции, а итоговая сумма блока записывается в глобальную память. На хосте частичные суммы из всех блоков собираются в итоговое значение π. Для CPU-версии (CalcPi) используется последовательный цикл, суммирующий вклад каждого интервала, что проще, но

медленнее для больших значений n. GPU-реализация благодаря параллелизму обрабатывает большие объемы данных значительно быстрее. Показана реализация функции CalcPiGPUи исходной CalcPi, листнинг 3.

```
// Оригинальная функция для вычисления числа π на CPU
double CalcPi(const int n) {
double pi = 0;
const double coef = 1.0 / n;

for (int i = 0; i < n; ++i) {
const double xi = (i + 0.5) * coef;
pi += 4.0 / (1.0 + xi * xi);
}
return pi * coef;
}

// Функция для вычисления числа π на GPU
double CalcPiGPU(const int n) {
const int blockSize = 256; // Количество потоков в блоке
const int numBlocks = (n + blockSize - 1) / blockSize; // Количество блоков
// Выделение памяти на устройстве
double* d_partialSums;
cudaMalloc(&d_partialSums, numBlocks * sizeof(double));
// Запуск CUDA ядра
calcPiKernel<<<numBlocks, blockSize, blockSize * sizeof(double)>>>(n, d_partialSums);
// Копирование частичных сумм на хост
double* h_partialSums = new double[numBlocks];
cudaMemcpy(h_partialSums, d_partialSums, numBlocks * sizeof(double), cudaMemcpyDeviceToHost);
// Редукция на хосте
double pi = 0.0;
for (int i = 0; i < numBlocks; ++i) {
pi += h_partialSums[i];
}
// Освобождение памяти
delete[] h_partialSums;
cudaFree(d_partialSums);
return pi * (1.0 / n);
}
```

Листнинг 3. Реализация функций вычисления числа на ЦП и ГП

Выводы:

Изучили реализацию вычисления числа π методом прямоугольников на CPU и GPU, что позволило сравнить их производительность. Реализация на GPU использует CUDA-ядро для параллельного вычисления, эффективно распределяя задачи между потоками и ускоряя процесс за счет массового параллелизма. CPU-реализация показала простоту, но значительно уступила по времени выполнения при больших объемах данных, что демонстрирует преимущества GPU для вычислительных задач.