МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

**(«ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №3

по дисциплине «Параллельные вычисления на графических процессорах»

Выполнил студент группы ИВТ-32 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Щесняк Д. С./

Проверил доцент кафедры ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Вожегов Д. В./

Киров 2017

1. Задание на лабораторную работу

Целью лабораторной работы является распараллеливание алгоритма, разработанного в предыдущей лабораторной работе, с использованием SSE.

1. Блок-схемы алгоритма

Блок-схемы алгоритма представлены на рисунках 1, 2, 3.



Рисунок 1 – Блок-схема основного алгоритма



Рисунок 2 – Блок-схема функции обучения



Рисунок 3 – Блок-схема функции поиска индекса максимального элемента

1. Исходный код

Параллельные версии функций представлены на рисунке 4.

|  |
| --- |
| Matrix\* sse\_sum(Matrix\* a, Matrix\* b, int clr=3) {  Matrix\* m;  if (clr & 1) m = a;  else if (clr & 2) m = b;  else m = zeros(a->lines, a->cols, 0);  //#pragma omp parallel for  for (int i = 0; i < a->lines; i++) {  int len = a->cols / 4;  \_\_m128\* ptr\_a = (\_\_m128\*) a->value[i];  \_\_m128\* ptr\_b = (\_\_m128\*) b->value[i];  float\* m\_ptr = m->value[i];  for (int i = 0; i < len; i++, ptr\_a++, ptr\_b++, m\_ptr += 4) {  \_mm\_store\_ps(m\_ptr, \_mm\_add\_ps(\*ptr\_a, \*ptr\_b));  }  }  if ((clr & 1) && (clr & 2)) {  clear(b);  }  return m;  }    Matrix\* sse\_sub(Matrix\* a, Matrix\* b, int clr=3) {  Matrix\* m;  if (clr & 1) m = a;  else if (clr & 2) m = b;  else m = zeros(a->lines, a->cols, 0);  //#pragma omp parallel for  for (int i = 0; i < a->lines; i++) {  int len = a->cols / 4;  //if (a->cols % 4 > 0) len++;  \_\_m128\* ptr\_a = (\_\_m128\*) a->value[i];  \_\_m128\* ptr\_b = (\_\_m128\*) b->value[i % b->lines];  float\* m\_ptr = m->value[i];  for (int j = 0; j < len; j++, ++ptr\_a, ++ptr\_b, m\_ptr += 4) {  \_mm\_store\_ps(m\_ptr, \_mm\_sub\_ps(\*ptr\_a, \*ptr\_b));  }  for (int j = a->cols - (a->cols % 4); j < a->cols; j++) {  m->value[i][j] = a->value[i][j] - b->value[i % b->lines][j];  }  }  if ((clr & 1) && (clr & 2)) {  clear(b);  }  return m;  }  template <typename t>  Matrix\* sse\_mul(Matrix\* a, t b, int clr=3) {  Matrix\* m;  float\* vec = (float\*) \_aligned\_malloc(4 \* sizeof(float), 16);  vec[0] = b;  vec[1] = b;  vec[2] = b;  vec[3] = b;  \_\_m128\* ptr\_b = (\_\_m128\*) vec;  if (clr & 1) m = a;  else m = zeros(a->lines, a->cols, 0);  //#pragma omp parallel for  for (int i = 0; i < a->lines; i++) {  int len = a->cols / 4;  \_\_m128\* ptr\_a = (\_\_m128\*) a->value[i];  float\* m\_ptr = m->value[i];  for (int j = 0; j < len; j++, ptr\_a++, m\_ptr += 4) {  \_mm\_store\_ps(m\_ptr, \_mm\_mul\_ps(\*ptr\_a, \*ptr\_b));  }  for (int j = a->cols - (a->cols % 4); j < a->cols; j++) {  m->value[i][j] = a->value[i][j] \* b;  }  }  \_aligned\_free(vec);  return m;  } |

Рисунок 4 – Исходный код модуля Matrix

1. Оценка времени выполнения

Среднее время выполнения обучения одной логистической регрессии 80 секунд, что в 1.375 раз меньше, чем выполнение линейной реализации алгоритма. Результаты измерения времени представлены на рисунке 5

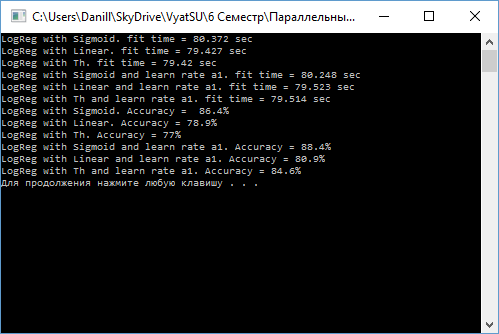


Рисунок 5 – Результаты измерения времени

1. Вывод

В ходе выполнения лабораторной работы был реализованы алгоритмы обучения и тестирования модели логистической регрессии, так же были разработаны sse реализации функции выполняющие основные операции над матрицами: сложение, умножение, вычитание и тд. Было измерено время выполнения линейной программы без использования средств распараллеливания. Среднее время обучения одной логистической регрессии получилось равным 80 секундам.