МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Программирование графических процессоров»

Моделирование поведения роя частиц на GPU

Выполнил: Н.И.Забарин
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2017

Условие

1. **Цель работы.** Научиться использовать GPU для моделирования поведения косяка рыб (стаи птиц) при решении задачи глобальной оптимизации.
   Общая постановка задачи.
   - Реализовать алгоритм роя частиц для задачи глобальной оптимизации Inertia Weighted SPO. Коэффициент инерции w и параметры a1, a2 задаются пользователем.
   - Для решения проблемы коллизий, необходимо ввести силу отталкивания обратно пропорциональную расстоянию в некоторой степени (например в четвертой) между частицами. Производить учёт только локальных взаимодействий. Для локализации частиц требуется использовать разбиение пространства.
   - Должна присутствовать 2D визуализация работы алгоритма. Частицы (рыбы/птицы) можно отобразить кружочками. На экран необходимо вывести температурную карту рассматриваемой функции (задается вариантом), по которой будут перемещаться частицы. Камера должна следовать за роем частиц (например, за центром масс), т.е. если частицы переместились за область видимости экрана, то камера перемещается в след за ними. У пользователя должна быть возможность интерактивно изменять масштаб.

   На разработанном программном обеспечении выполнить исследование зависимости скорости сходимости от параметров w, a1 и a2, и отразить результаты в отчете. Так же необходимо подобрать такие параметры w, a1 и a2, при которых поведение роя частиц будет наиболее соответствовать поведению стаи птиц (косяка рыб) в природе. Провести сравнение производительности gpu и cpu.

2. **Вариант задания:** 2. Функция Химмельблау

Программное и аппаратное обеспечение

Спецификации GPU

Name:	GeForce GT 620M
Compute capability:	2.1
Warp size:	32
Max threads per block:	1024
Clock rate:	1250000
Multiprocessor count:	2
Max threads dim:	1024 1024 64
Max grid size:	65535 65535 65535

Спецификации видеопамяти

Total global memory:	1024 MB
Shared memory per block:	48 KB
Registers per block:	32 KB
Total constant memory:	64 KB

Спецификации CPU

Процессор	Intel Core i5-3317U
Ядер	4
Базовая частота	1.7 GHz

Спецификация оперативной памяти

Объем памяти	10 Гб
Частота	1600 МГц

Спецификация жесткого диска

Тип	SSD
Интерфейс	M.2
Объём	240Gb

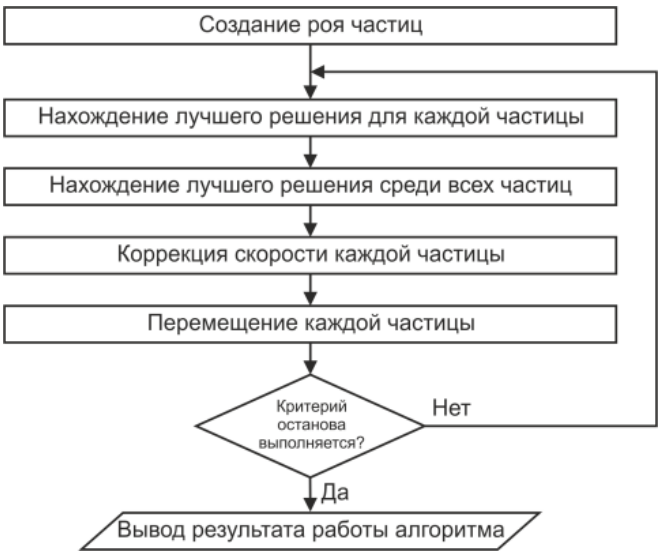
Спецификация программного обеспечения

CUDA Toolkit	7.5
OS	Ubuntu 16.10
IDE	Vim
Compiler	**nvcc V7.5.17**

Метод решения

Алгоритм роя частиц моделирует многоагентную систему, где агенты-частицы двигаются к оптимальным решениям, обмениваясь при этом информацией с соседями. Текущее состояние частицы характеризуется координатами в пространстве решений, а также вектором скорости перемещения. Оба этих параметра выбираются случайным образом на этапе инициализации. Кроме того, каждая частица хранит координаты лучшего из найденных ей решений, а также лучшее из пройденных всеми частицами решений – этим имитируется мгновенный обмен информацией между птицами.

Блок схема алгоритма роя частиц



В первоначальном виде алгоритма коррекция скорости выглядела следующим образом:

$$v_{i,t+1} = v_{i,t} + \varphi_p r_p (p_i - x_{i,t}) + \varphi_g r_g (g_i - x_{i,t})$$

Здесь $v_{i,t}$ – i-я компонента скорости при t-ой итерации алгоритма $x_{i,t}$ – i-я координата частицы при t-ой итерации алгоритма p_i – i-я координата лучшего решения, найденного частицей g_i – i-я координата лучшего решения, найденного всеми частицами r_p, r_g – случайные числа в интервале (0, 1) φ_p, φ_g – весовые коэффициенты, которые надо подбирать под конкретную задачу.

Затем корректируем текущую координату каждой частицы

$$x_{i,t+1} = x_{i,t} + v_{i,t+1}$$

После этого рассчитываем значение целевой функции в каждой новой точке, каждая частица проверяет, не стала ли новая координата лучшей среди всех точек, где она побывала. Затем среди всех новых точек проверяем, не нашли ли мы новую глобально лучшую точку, и, если нашли, запоминаем ее координаты и значение целевой функции в ней.

Одна из модификаций алгоритма (реализованного в этой курсовой работе) состоит в том, чтобы добавить еще один весовой коэффициент ω перед текущей скоростью (коэффициент инерции), благодаря которому скорость изменяется более плавно.

$$v_{i,t+1} = \omega\, v_{i,t} + \varphi_p\, r_p\, (p_i - x_{i,t}) + \varphi_g\, r_g\, (g_i - x_{i,t})$$

Для того, чтобы поведение частиц было больше похоже на поведение живых объектов (чтобы все частицы не собирались в одной точке), между частицами было введено взаимодействие (отталкивающая сила).

Описание программы

Основная логика реализована в функции void update(). В ней производится отрисовка очередного кадра.

```
void update() {
   uchar4* screen;
   size_t size;
   CSC(cudaGraphicsMapResources(1, &res, 0));
   CSC(cudaGraphicsResourceGetMappedPointer((void**)&screen, &size, res));
   draw_map<<<dim3(16, 16), dim3(16, 16)>>>(screen);
   draw_particle<<<16, 16>>>(screen);

   if (STATE) {
      calc_lbest<<<16, 16>>>();
      calc_gbest();
      calc_particle<<<16, 16>>>(rand());
      calc_center();
   }

   CSC(cudaDeviceSynchronize());
   CSC(cudaGraphicsUnmapResources(1, &res, 0));

   glutPostRedisplay();
}
```

Вначале вызывается ядро `draw_map`, заполняющее буфер значениями функции.

Затем вызывается ядро `draw_particle`, рисующее все частицы, центры которых хранятся в массиве float2 *points.

Для соотнесения координат в пространстве частиц и координат пикселей, реализованы четыре дефайна:

```
// P = plane, S = screen
#define point_S2Px(xa) ((float)(xa-(WIDTH/2)) * ((float)K_RADIUS / WIDTH) * 2 + K_CENTER.x)
#define point_S2Py(ya) ((float)(ya-(HEIGHT/2)) * ((float)K_RADIUS / HEIGHT) * 2 + K_CENTER.y)

#define point_P2Sx(xa) (((float)xa - K_CENTER.x) * ((float)WIDTH / K_RADIUS) / 2 + WIDTH / 2)
```

#define point_P2Sy(ya) (((float)ya - K_CENTER.y) * ((float)HEIGHT / K_RADIUS) / 2 + HEIGHT / 2)

Далее вызывается ядро calc_lbest(), обновляющее (если надо) лучшую точку, найденную каждой частицей. Результаты хранятся в массиве float3 *lbest.

Далее вызывается функция calc_gbest(), находящая минимум в массиве lbest и, если значение стало меньше, записывающая новой значение в переменную float2 gbest.

Затем вызывается ядро calc_particle(), пересчитывающее векторы скорости и координаты частиц по уже обновленным ранее данным.

Далее вызывается процедура calc_center(), обновляющая значения центра, как центра массы всех частиц.

Исследовательская часть

Если частицы равномерно распределить по области в которой находится глобальный экстремум, то алгоритм очень быстро находит его. Исключением является большой вес a1 или сильная инерция w, при близких к единице значениям частицы начинают роиться.

При отсутствии глобального и локального минимума частицы устремятся в его сторону, после какого то количества итераций, они начнут двигаться в одном направлении.

Результаты

Производительность реализации замерялась на 50, 200 и 500 точках. В таблице приведено время вычисления одной итерации в мс.

Количество частиц	GPU	CPU
50	14	80
200	20	120
500	47	372

Выводы

Алгоритм роя частиц – оптимизационный алгоритм, то есть он может широко применяться во многих насущных проблемах, в числе прочих, в задачах машинного обучения (в частности, для обучения нейросетей и распознавания изображений), параметрической и структурной оптимизации (форм, размеров и топологий) в области проектирования, в областях биохимии и биомеханики.

По эффективности он может соперничать с другими методами глобальной оптимизации, а низкая алгоритмическая сложность способствует простоте его реализации.

Сложность реализации и время работы алгоритма возрастают, при введении учета взаимодействий между частицами в виде сил отталкивания. Эти вычисления можно оптимизировать, реализовав разбиение пространства, однако это еще больше усложняет разработку. Введение сил не помогает в поиске оптимума, однако визуально делает передвижение точек более интересным и “живым”.

Одной из главных проблем при решении задач оптимизации является баланс между тщательностью исследованием пространства поиска и скоростью сходимости алгоритма. В зависимости от задачи и характеристик поискового пространства в ней, этот баланс должен быть различным. Поэтому в классическом алгоритме роя частиц был введен коэффициент инерции (Inertia Weighted PSO), определяющий упомянутый баланс между широтой исследования и вниманием к найденным субоптимальным решениям.