УДК 519.6

МГТУ им. Н.Э. Баумана

karpenko@nog.ru

Введение

Методы решения задачи глобальной безусловной оптимизации делятся на детерминированные, стохастические и эвристические [0]. Среди эвристических методов выделяются эволюционные и поведенческие (имитационные) методы, которые представляют собой относительно новый и быстро развивающийся класс методов глобальной оптимизации

Известность получили следующие поведенческие методы решения задачи глобальной оптимизации - метод поведения пчёл, метод колонии муравьев, метод роя частиц. В работе рассматривается параллельный метод глобальной оптимизации роем частиц (particle swarm optimization - PSO). Обзор как последовательных, так и параллельных вариантов метода PSO приведен в работе [0].

Предложенный в данной работе параллельный метод GIPSO (от GPU Island Particle Swarm Optimization) ориентирован на использование графических процессорных устройствах (GPU). Основу GIPSO составляет известный метод IPSO (Island Particle Swarm Optimization), использующий островную модель параллелизма [0].

Современные графические процессорные устройства (ГПУ) являются мощными специализированными вычислительными устройствами, которые могут быть использованы для решения целого ряда сложных вычислительных задач. В настоящее время одно ГПУ позволяет получить до 900 Гфлопс реальной производительности и обеспечить примерно десятикратное ускорение по сравнению центральным процессором хост-ЭВМ. Важно, что помимо высокой производительности ГПУ обладают низким энергопотреблением и низкой относительной стоимостью. Основная идея использования ГПУ в

качестве универсального вычислителя состоит в использовании пиксельных шейдеров как универсальных сопроцессоров для операций с плавающей точкой [4].

В настоящее время существует два основных производителя ГПУ для высокопроизводительных вычислений – компании AMD и NVidia. Архитектуры ГПУ этих производителей существенно отличаются. Работа ориентирована на ГПУ производства компании Nvidia.

С точки зрения программирования вычислительная система на основе ГПУ представляет собой сложную иерархическую структуру, имеющую несколько адресных пространств и ряд специфических механизмов. Широкому использованию ГПУ для решения вычислительных задач до недавнего времени препятствовало отсутствие адекватных средств программирования, не связанных с программированием графики. Ситуация изменилась с появлением в 2008 году библиотеки CUDA (Compute Unified Driver Architecture) компании NVidia [4]. Основу библиотеки (нижний уровень программирования) составляет ассемблер ГПУ, верхний уровень библиотеки представляет собой параллельный диалект языка С, отражающий архитектурные особенности ГПУ.

Первый раздел работы содержит постановку задачи и схему метода GIPSO. Алгоритм метода GIPSO (алгоритм GIPSO) учитывает архитектурные особенности ГПУ nVidia, поэтому во втором разделе работы рассмотрены основные из этих особенностей. В третьем разделе представлен алгоритм GIPSO, а в четвертом разделе – его программная реализация. Пятый раздел посвящен экспериментальному исследованию эффективности алгоритма. В заключении сформулированы основные результаты работы и перспективы ее развития.

Отметим, что эффективность алгоритма GIPSO сравнивается с эффективностью последовательной реализации алгоритма IPSO на однопроцессорном компьютере архитектуры x86. В работе [3] описаны похожие реализации алгоритма IPSO в классических параллельных средах с использованием технологии MPI.

1. Постановка задачи и схема метода GIPSO

Рассмотрим задачу глобальной безусловной минимизации целевой функций $F(X)$, определенной в n-мерном арифметическом пространстве R^n (пространстве поиска):

$$\min_X F(X) = F(X^*), \quad X \in R^n. \tag{1}$$

Множество частиц (рой, популяцию) обозначим $S = \{P_i, \ i \in [\overline{1,N}]\}$, где N – количество частиц в рое (размер популяции). Частица P_i определяется n-мерными векторами координат $X_{i,t}$ и скоростей $V_{i,t}$. Здесь $t \in [\overline{0,\mathrm{T}}]$ - дискретный момент времени (номер итерации); T - общее число итераций. Начальные координаты и скорости частицы P_i известны и равны $X_{i,0} = X_i^0$, $V_{i,0} = V_i^0$ соответственно.

Метод GIPSO (как и метод IPSO) основан на каноническом методе PSO, итерации в котором выполняются по следующей схеме [0]:

$$X_{i,t+1} = X_{i,t} + V_{i,t+1}; \tag{2}$$

$$V_{i,t+1} = \alpha V_{i,t} + U[0,\phi_1] \otimes (X_{i,t}^b - X_{i,t}) + U[0,\phi_2] \otimes (X_{g,t} - X_{i,t}). \tag{3}$$

Здесь $U[a,b]$ - n-мерный вектор псевдослучайных чисел, равномерно распределенных в интервале $[a,b]$; $\otimes$ - символ покомпонентного умножения векторов; $X_{i,t}^b$ - вектор координат частицы P_i с наилучшим (в смысле формулы (1)) значением целевой функции $F(X)$ за все время поиска $[\overline{0,t}]$; $X_{g,t}$ - вектор координат соседней с данной частицы с наилучшим за то же время значением целевой функции $F(X)$; α, ϕ_1, ϕ_2 - свободные параметры алгоритма; соседство частиц определяется используемой топологией соседства [0].

В методе GIPSO весь рой частиц S разбивается на K суброев S_k, $k = [\overline{1,K}]$,

число частиц в каждои из которых равно ν так, что $N = \nu K$. Суброи обрабатываются независимо на протяжении «сезона», состоящего из T_{mig} итераций, $T_{mig} \le T$. По окончании каждого из сезонов выполняется процедура миграции частиц между суброями, которая может осуществляться с использованием стратегии перемещения и стратегии репликации.

Миграция перемещением частиц заключается в обмене наилучшей частицы из одного суброя и наихудшей частицы из другого суброя: для каждого суброя S_k лучшая частица P_{best}^k меняется местами с частицей P_{worst}^l из суброя S_l; $k,l = [\overline{1,K}]$; $k \ne l$. Выбор суброя назначения S_l может происходить случайным образом либо в соответсвие с заданной топологией соседства частиц [0].

В миграции репликацией сначала в каждом из суброев S_k осуществляется поиск лучшей частицы P_{best}^k и худшей частицы P_{worst}^k; $k = [\overline{1,K}]$. Из всех частиц P_{best}^k выбирается наилучшая частица всего роя P_{best}^κ. Затем из суброя S_κ осуществляется репликация (копирование) частицы P_{best}^κ во все остальные суброи S_k на место частиц P_{worst}^k.

Заметим, что миграция перемещением имеет целью исключить наиболее успешную частицу из суброя в случае тенденции его к преждевременной сходимости. При этом уменьшается скорость сходимости алгоритма, но обеспечивается более планомерное исследование пространства поиска. Миграция репликацией, наоборот, преследует цель обеспечить более высокую скорость сходимости итераций.

2. Основные особенности архитектуры и программирования ГПУ

Основу ГПУ компании NVidia, таких как видеокарта nVidia GeForce 8 и выше, составляет массив, включающий в себя от четырех до тридцати двух SIMD-мультипроцессоров. Мультипроцессор состоит из восьми суперскалярных потоковых процессоров, каждый из которых имеет свой

модуль памяти. В соответствии со спецификацией Compute Capability 1.2 емкость этой памяти равна G_{reg}=16 КВ [4].

Мультипроцессор включает в себя независимую кэш-память команд и аналогичную память данных (констант), планировщик потоков и модуль памяти, разделяемой между всеми восемью потоковыми процессорами. По спецификации Compute Capability 1.2 емкость разделяемой памяти составляет G_{shm}=16 КБ.

Помимо собственно ГПУ, видеокарта содержит также модули глобальной оперативной памяти емкостью до 1 ГБ.

Модель CUDA-программы представляет собой сеть вычислительных блоков, состоящих из большого количества потоков. Планировщик потоков ГПУ осуществляет динамическое распределение вычислительных блоков на мультипроцессоры, а потоки внутри блока - на потоковые процессоры. Эффективно распараллелена может быть только та задача, которая допускает представление в виде множества независимо и одновременно (на логическом уровне) выполняемых копий функции, обрабатывающих разные наборы данных.

Обмен данными возможен только между потоками, но не между блоками. Коммуникация потоков может осуществляться через разделяемую память с применением барьерной синхронизации.

Известной проблемой современных процессоров является дисбаланс между скоростью работы процессора и скоростью обмена с памятью. Для ГПУ эта проблема стоит еще более остро, поскольку задержка доступа потокового процессора к глобальной памяти может составлять до 300 тактов. ГПУ позволяет снизить латентность доступа к памяти путем планирования исполнения на мультипроцессоре совокупности 32 потоков, которая называется варпом (warp). При блокировании на доступе к памяти потоков одного из варпов, исполнение быстро переключается на следующий варп. Важно, что при этом переключение контекста занимает только один такт. Отметим, что для эффективной работы указанного механизма количество потоков в блоке

должно многократно превышать количество потоковых процессоров в мультипроцессоре.

3. Отображения алгоритма GIPSO на архитектуру ГПУ

Поскольку частицы внутри каждого из суброев S_k достаточно интенсивно коммутируют (после каждой итерации выбирается лучшая частица), в алгоритме GIPSO частицы каждого из суброев обрабатываются одним вычислительным блоком. Каждой из частиц ставится в соответсвие свой CUDA-поток. Для уменьшения влияния латентности глобальной памяти, целесообразно использовать число блоков (а значит и суброев), которое значительно превосходит число мультипроцессоров ГПУ.

В CUDA нет средств явного управления объемом памяти, используемой потоком. Если локальные данные потока не помещаются в регистровую память, то CUDA-компилятор (NVCC) использует для этих данных разделяемую память мультипроцессора (очень медленную, по сравнению с регистровой памятью). Поэтому при отображении алгоритма GIPSO на архитектуру ГПУ важен предварительный анализ требуемых для реализации алгоритма объемов памяти.

Пусть потоку для его выполнения требуется R_{st} регистров. Тогда для вычислительного блока, состоящего из ν потоков, число требуемых регистров равно

$$R_b = round\left(R_{st} \cdot round(\nu, 32), \frac{G_{reg}}{32} \right) \le G_{reg} \qquad (4)$$

где $round(p,q)$ - ближайшее целое, меньшее p и кратное целому q.

Объем общей памяти (в байтах), используемой алгоритмом GIPSO для одного суброя, равен

$$Q_{shm} = 4(3n\nu + \nu + n + 1) \le G_{shm}.$$

Отсюда имеем следующее ограничение на число частиц в суброе:

$$\nu \le \frac{\frac{G_{shm}}{4} - (n+1)}{3n+1}. \qquad (5)$$

Таким образом, на размеры суброев ν накладываются ограничения (4), (5), в то время как число суброев K практически неограничено. Отметим, что величины K, ν связаны между собой очевидным соотношением $N = K\nu$.

В соответствии с рекомендациями работы [5] в алгоритме GIPSO скорости частиц ограничены сверху величиной $V_{\max} = \gamma X_{\max}$, где $X_{\max}$ - размер области поиска: $X_i^- \le X_{i,t} \le X_i^+$; $X_i^+ - X_i^- \le X_{\max}$; $i = [\overline{1,n}]$, $t = [\overline{1,T}]$. Параметр $\gamma \in [0{,}01;1]$.

Метод PSO отличается от многих других эволюционных методов оптимизации наличием большого количества свободных настраиваемых параметров. Значения этих параметров могут оказывать существенное влияние на основные характеристики алгоритма такие, как скорость сходимости, сам факт сходимости, объем исследованной части пространства поиска и др. Рекомендации по выбору параметров метода PSO приведены в работе [5].

Поскольку алгоритмы IPSO и GIPSO основываются на классическом алгоритме PSO, они наследуют указанные выше параметры, но также добавляют несколько новых параметров:

- число суброев K;
- число частиц в одном суброе - ν;
- интервал миграции - T_{mig};
- стратегия миграции.

4. Программная реализация алгоритма GIPSO

Основу программы GIPSO составляют два типа вычислительных блоков, которые называются вычислительными ядрами.

Вычислительное ядро первого типа реализует движение частиц соответствующего суброя в течение одного сезона. Вычислительное ядро второго типа осущетсвляет миграцию частиц между суброями.

Вычислительные ядра первого типа образуют K вычислительных блоков, а

вычислительные ядра второго типа - один блок, состоящий из K потоков.

Программа GIPSO включает в себя две реализации ядра движения частиц:

- в реализации *PsoV1* поток соответсвует одной частице, так что общее число потоков равно N, а число потоков в блоке - ν;
- в реализации *PsoV2* поток отвечает за движение одной частицы по одной размерности области поиска; общее число потоков равно Nn, а число потоков в одном блоке - νn.

Ядро миграции в программе GIPSO реализует стратегию миграции, как репликацией, так и перемещением. Ядро миграции призвано обеспечить распараллеливание операций миграции и переноса данных непосредственно в глобальную память ГПУ, не копируя их в память host-процессора.

Структура программы GIPSO имеет следующий вид.

a) Инициализация суброев (на хост-процессоре) – задание начальных положений и скоростей частиц.

b) Копирование начальных положений и скоростей частиц в глобальную память ГПУ.

c) Генерация на host-процессоре двух массивов, содержащих по nNT случайных чисел, двухравномерно распределенных в интервалах $[0,\phi_1]$, $[0,\phi_2]$ соответственно (см. формулу (2)).

d) Копирование указанных массивов в глобальную память ГПУ.

e) Выполнение на ГПУ цикла оптимизации на протяжении первого сезона:

   e1) выполнение на ГПУ вычислительной сети из ядер движения роя;

   e2) передача host-процессору информации о лучших и худших частицах каждого из суброев (путем копирования соответствующих данных из глобальной памяти ГПУ в память host-процессора);

   e3) построение на host-процессоре плана миграции частиц;

   e4) передача соответствующих данных ГПУ;

   e5) выполнение на ГПУ вычислительной сети из ядер миграции суброев, которая в соответствии с разработанным планом осуществляет

перемещение частиц между суброями.

f) По схеме п^{0}е) выполнение на ГПУ цикла оптимизации на протяжении второго сезона и т.д. до последнего сезона, который завершается передачей host-процессору информации о лучших частицах каждого из суброев.

g) Нахождение host-процессором результата оптимизации – положения наилучшей частицы среду лучших частиц всех суброев, а также соответсвующего значения критерия оптимальности $F(X)$.

Заметим, что необходимость предварительной генерации на host-процессоре случайных чисел (см. шаг *c*), обусловлена тем обстоятельством, что ГПУ не способно генерировать такие числа.

Рассмотренная структура программы GIPSO показывает наличие значительного количества последовательных фрагментов, которые не могут быть распараллелены – шаги *a – d*, *e2 – e4*, *g*. По закону Амдала наличие таких фрагментов значительно снижает максимально возможное ускорение алгоритма.

Для оптимизации отображения алгоритма GIPSO на архитектуру CUDA следует соблюдать следующие требования:

- доступ потоков в варпе к глобальной памяти и памяти констант должен быть выровненным по адресам и осуществляться по определенным шаблонам;
- доступ различных потоков должен осуществляться к различным банкам разделяемой памяти;
- предпочтительно использовать регистры вместо локальной памяти;
- количество ветвлений, барьерных синхронизаций следует минимизировать.

В программе GIPSO обеспечены требования по доступу к разделяемой памяти с состоянием роя и к глобальной памяти с массивами случайных частиц на время итераций. Однократные чтение и запись начального состояния роя в глобальной памяти при инициализации (см. шаг *b*) не выравниваются, что

уменьшает производительность. Поскольку операция выполняется однократно, это уменьшение производительности не является критичным.

5. Исследование эффективности алгоритма GIPSO

При экспериментальном исследовании алгоритма GIPSO в качестве тестовых функций использованы функции известного набора CEC, который широко используется для анализа эффективности методов глобальной оптимизации. Рассмотрены функции Экли (Ackley), Растригина (Rastrigin), Розенброка (Rosenbrock) и сферическая функция (Sphere) [6].

В качестве области поиска рассмотрен гиперкуб $-20 \le X_i \le 20$, $i = [\overline{1,n}]$. Таким образом, принято $\left|X_i^-\right| = X_i^+ = X_{\max} = 20$ (см. п^{0}4). Длительность сезона миграции $T_{mig} = 50$.

5.1. Влияние свободных параметров. В работе исследовано влияние на характеристики алгоритма следующих его свободных параметров: число суброев K; размер популяции N.

Число суброев. Для случая использования стратегии миграции репликацией, результаты исследования влияния числа суброев K на скорость сходимости алгоритма приведены на рисунке 1. Для всех тестовых функций рисунок показывают примерно равномерное увеличение скорости сходимости по мере увеличения числа суброев.

Размер популяции. Для классического метода PSO размер популяции оказывает слабое влияние на скорость сходимости. Так метод практически нечувствителен к увеличению размера роя свыше 50 частиц. Общие рекомендации сводятся к использованию не более 50 частиц в рое для сложных задач оптимизации и 20 - 50 частиц - для простых задач [5].

По отношению к методу GIPSO такие рекомендации, вообще говоря, неприменимы. Дело в том, что, во-первых, многороевая структура метода и миграции частиц между суброями меняют смысл термина “размер популяции”. Во-вторых, ограничения, рассмотренные в п^{0}3, не позволяют использовать

большие суброи для решения задач оптимизации в пространстве высокой размерности.

Результаты исследования скорости сходимости алгоритма GIPSO в зависимости от размера популяции N для четырехмерного пространства поиска приведены на рисунке 2.

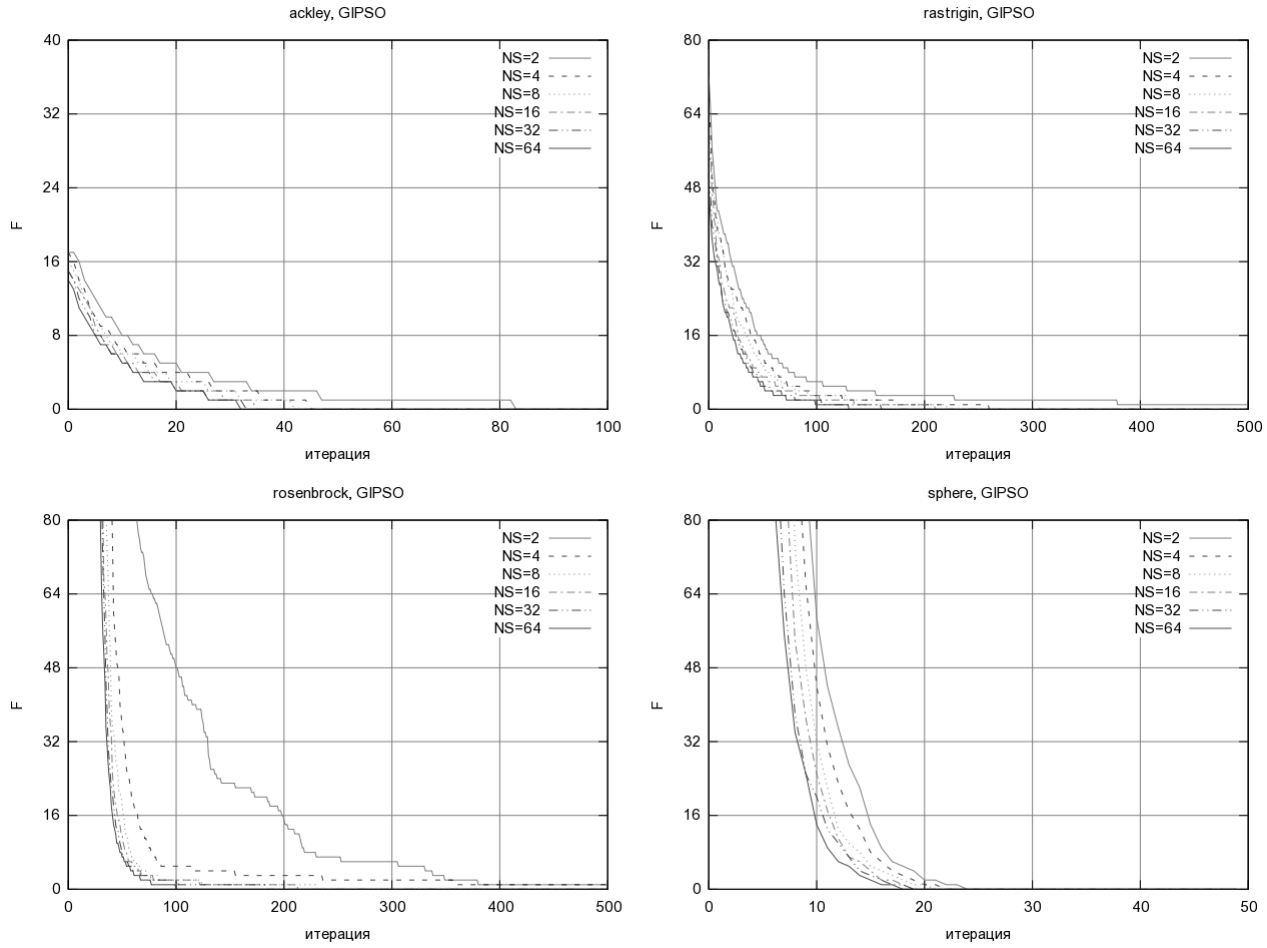

Рис. 1. Зависимость скорости сходимости алгоритма от числа суброев

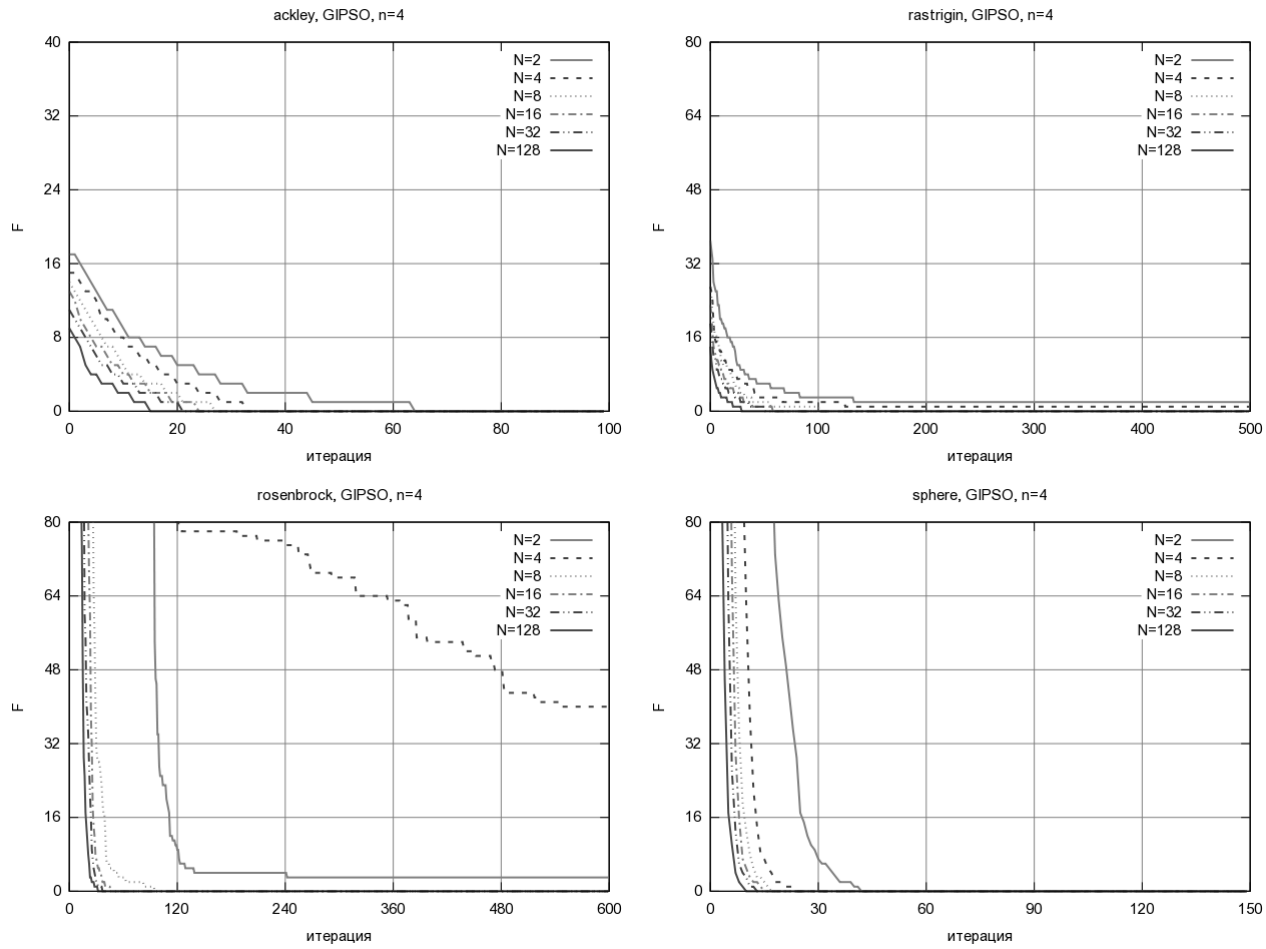

Рис. 2. Зависимость скорости сходимости от размера популяции ($n = 4$)

Количество суброев K во всех случаях равно четырем, количество частиц ν в суброях меняется от двух до 32, так что общее число частиц в популяции N меняется от восьми до 64.

Рисунок 2 показывает, что, как и для канонического метода PSO, при малой размерности пространства ($n = 4$) размер роя слабо влияет на скорость сходимости. Приемлемые результаты по скорости сходимости дают рои уже из четырех – восьми частиц. При этом во всех случаях имеет место локализация глобального минимума.

Аналогичные результаты для восьмимерного пространства поиска ($n = 8$) приведены на рисунке 3.

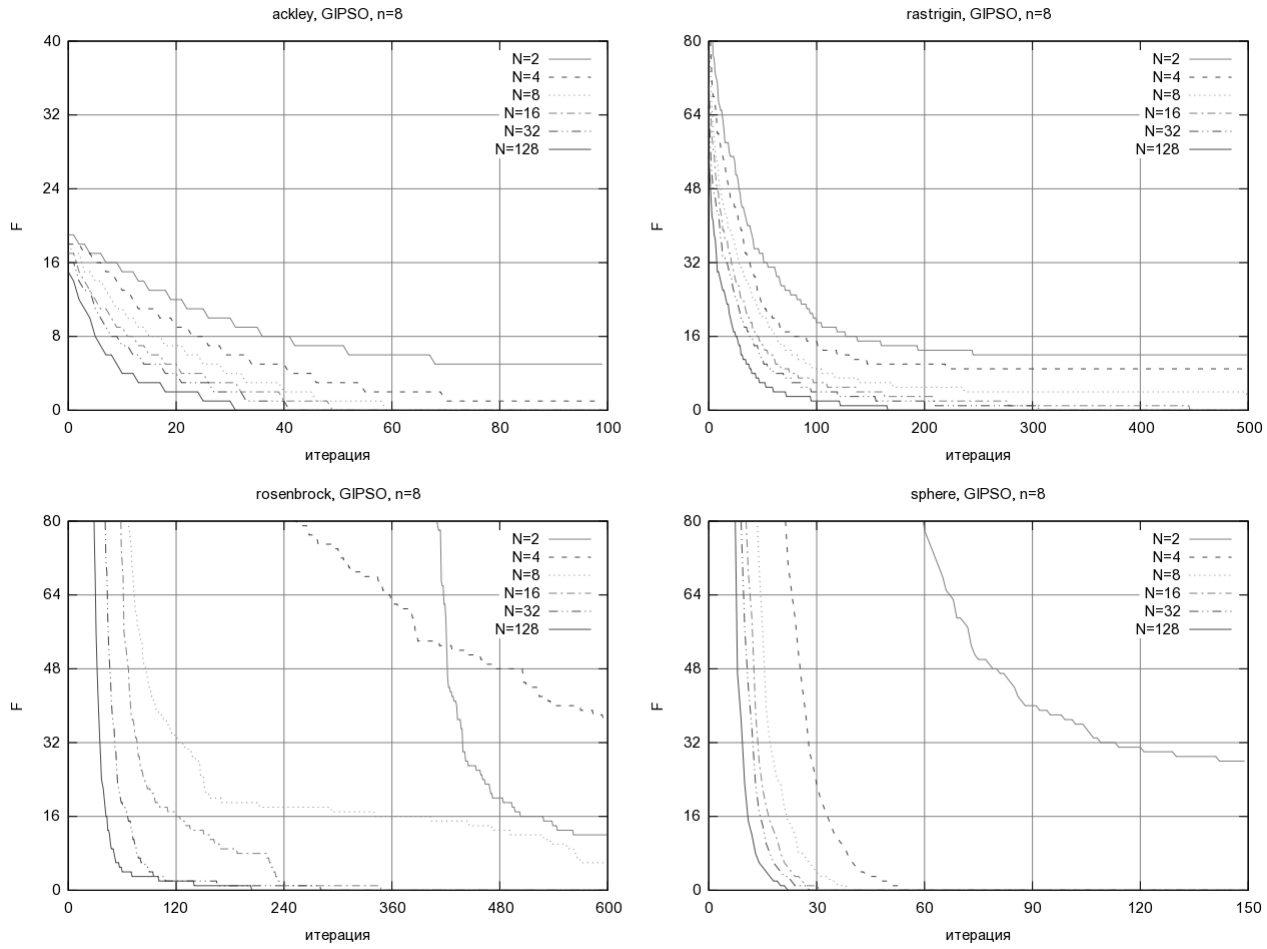

Рис. 3. Зависимость скорости сходимости от размера популяции ($n = 8$)

Рисунок показывает, что для пространства большей размерности ($n = 8$) размер роя начинает оказывать существенное влияние не только на скорость сходимости, но и на величину достигаемого минимума. Оказывается, что маленькие рои из 2 - 4 частиц неспособны вывести процесс поиска из локального минимума для функций Экли, Растригина и Розенброка. Минимальный размер роя, способного отыскать глобальный минимум всех рассматриваемых тестовых функций, составляет 8 частиц.

5.2. Оценка производительности. В работе выполнено также сравнительное исследование скорости сходимости алгоритмов GIPSO и IPSO (рисунок 4).

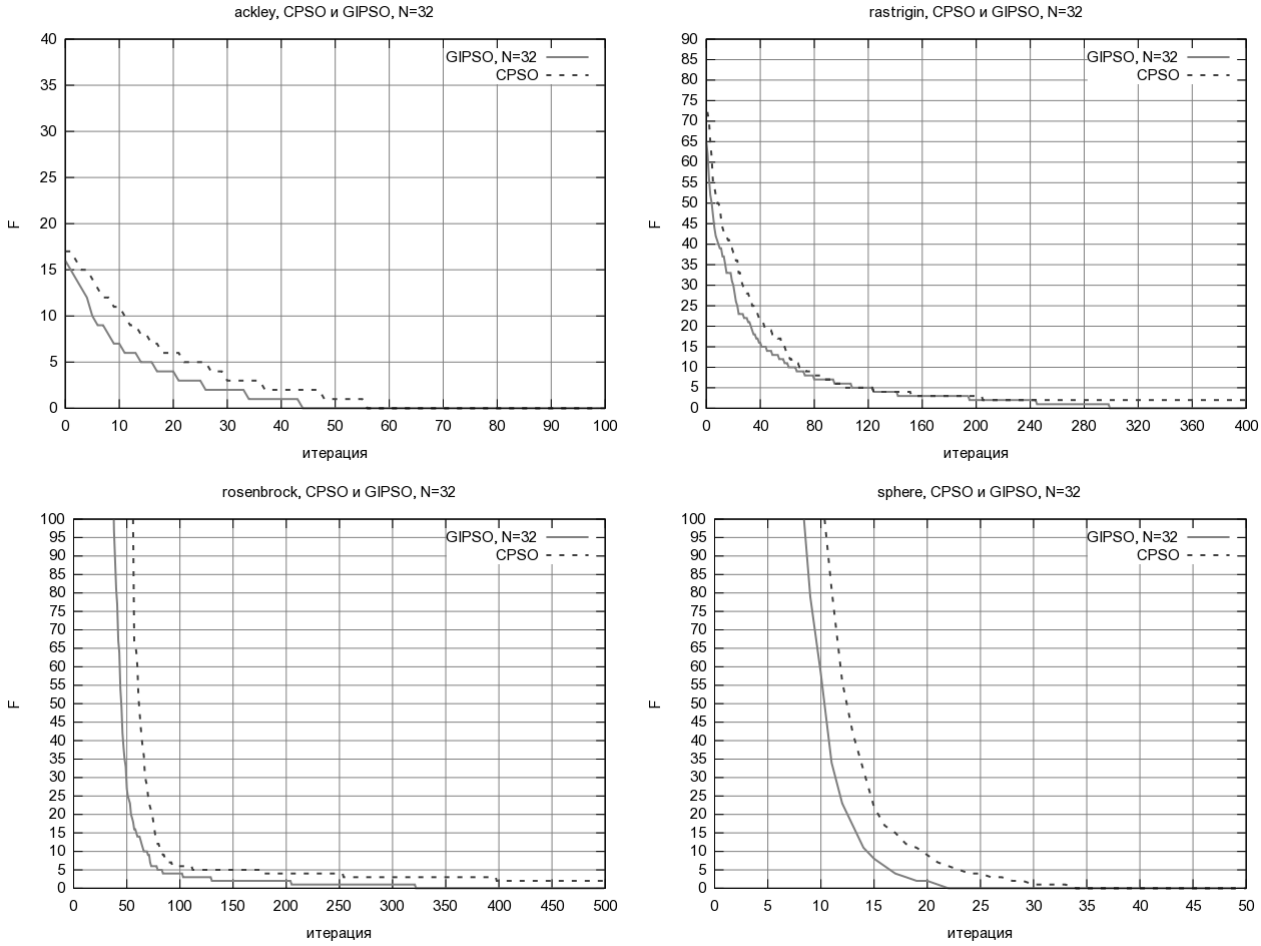

Рис. 4. Скорость сходимости алгоритмов GIPSO, IPSO

Исследование выполнено при следующих значениях свободных параметров:

$$\nu = 32 \text{ (IPSO);}$$

$$\nu = 32;\ K = 4;\ T_{mig} = 50 \text{ (GIPSO).}$$

Рисунок 4 показывает, что алгоритм GIPSO имеет более высокую скорость сходимости, чем алгоритм IPSO. Преимущества алгоритма GIPSO увеличиваются с ростом сложности тестовых функций.

Выполнено также исследование производительности алгоритма GIPSO по сравнению с алгоритмом IPSO, реализованным, напомним, на центральном процессоре системы с архитектурой x86.

Исследование выполнено для двух размеров популяции $N = 32$, $N = 64$ и размерности простанства поиска $n = 4,8,16$. В алгоритме GIPSO использовано

ядро *PsoV2* при количестве суброев $K = 4$.

Обозначим τ_{GIPSO} время решения задачи с указанной точностью алгоритмом GIPSO, а τ_{IPSO} - то же время, но при использовании алгоритма IPSO. В качестве меры производительности алгоритма GIPSO использовано ускорение

$$E = \frac{\tau_{IPSO}}{\tau_{GIPSO}}.$$

Результаты исследования иллюстрирует рисунок 5. Рисунок показывает, что в рассмотренном диапазоне изменения величины n ускорение линейно растет с ростом этой величины. Важно, что увеличение размера популяции с $N = 32$ до $N = 64$ (поскольку $K = 4$, для GIPSO это означает увеличение размеров суброев с 8 до 16) увеличивает ускорение почти в два раза. Этот эффект объясняется ростом загрузки мультипроцессоров с ростом числа частиц и / или размерности пространства поиска.

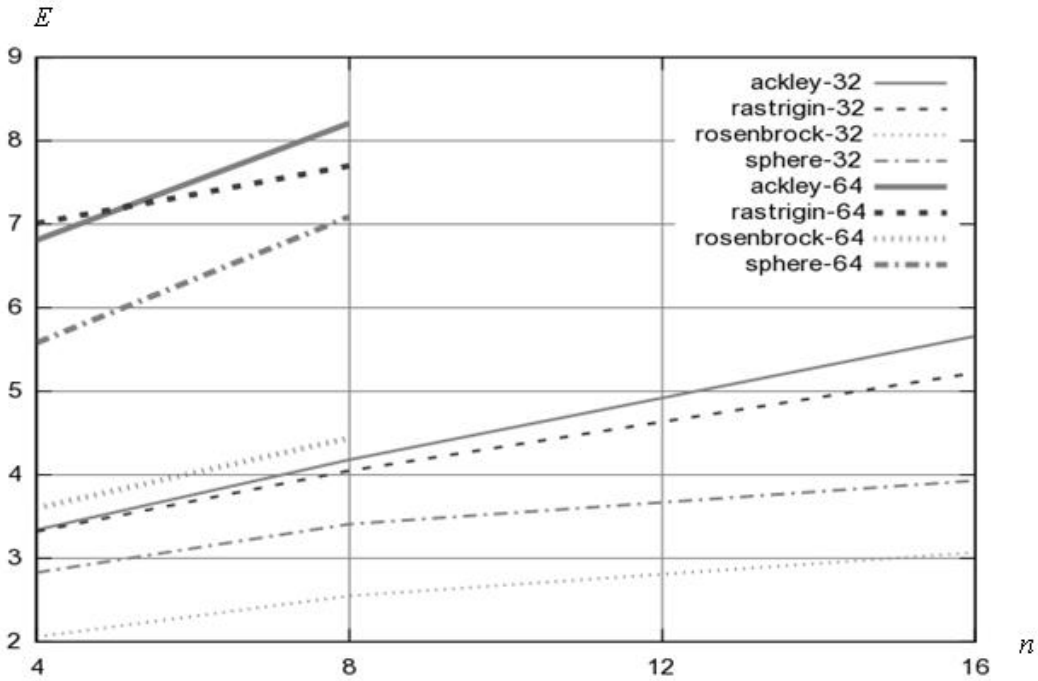

Рис. 5. Ускорение алгоритма GIPSO

5.3. Масштабируемость алгоритма. Выполнено исследование

зависимости приведенного времени $\bar{\tau}_{GIPSO}$, необходимого для решения задачи алгоритмом GIPSO, от количества суброев K при фиксированном размере популяции $N = 16$. Здесь

$$\bar{\tau}_{GIPSO} = \frac{\tau_{GIPSO}}{K}.$$

Отметим, что из-за ограничений (4), (5) при большой размерности пространства поиска n размер популяции снижался до $N = 8$.

Использовалось ядро алгоритма *PsoV2*. В качестве тестовой функции рассмотрена функция Розенброка.

Результаты исследования иллюстрирует рисунок 6. Рисунок показывает, что в алгоритме GIPSO возможно увеличение количества суброев, сопровождающееся линейным ростом времени решения задачи – величина $\bar{\tau}_{GIPSO}$ практические не меняется при числе роев более 16. Это обстоятельство позволяет при увеличении сложности решаемой задачи оптимизации увеличивать количество суброев для повышения эффективности алгоритма. Конечно, при этом не следует допустить преждевременной стагнации процесса поиска.

Таким образом, исследование показывает, что алгоритм GIPSO хорошо масштабируется при реализации его на ГПУ с использованием технологии CUDA.

Заключение

В работе рассмотрен параллельный метод GIPSO глобальной оптимизации роем частиц для ГПУ, а также соответствующий алгоритм GIPSO и его программная реализация с использованием технологии CUDA.

Выполнено широкое экспериментальное исследование различных аспектов эффективности метода. На представительном классе тестовых функций показано, что, по сравнению с классическим последовательным методом роя частиц, метод GIPSO обеспечивает более высокую скорость сходимости.

Исследование показало также, что метод обеспечивает значительное ускорение по сравнению с последовательным аналогом и хорошую масштабируемость относительно размера задачи.

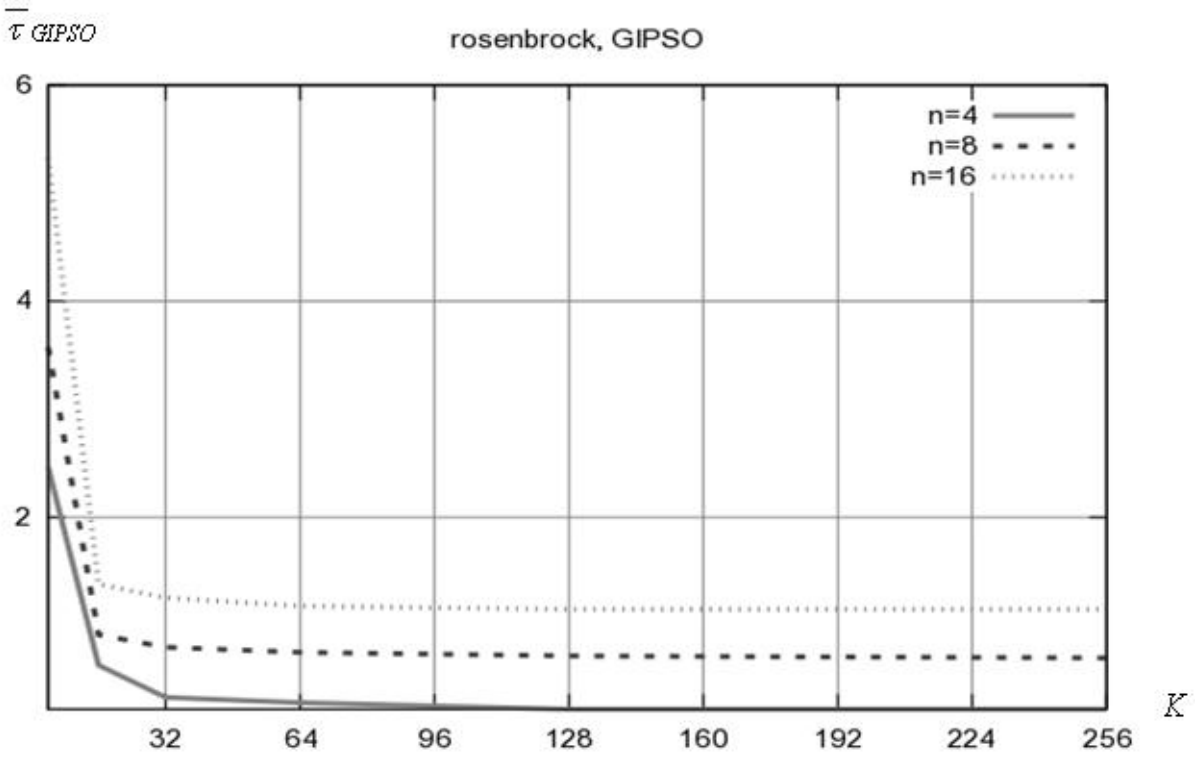

Рис. 6: Масштабируемость алгоритма GIPSO

На основе результатов работы можно констатировать, что многие задачи глобальной оптимизации могут быть эффективно решены на графических процессорах методом GIPSO.

В развитие работы предполагается разработка формальных методов, обеспечивающих оптимальное или квазиоптимальное отображение параллельных алгоритмов роя частиц на архитектуру графических процессоров, разработка методов параметрического согласования параллельных алгоритмов роя частиц с архитектурой используемого графического процессора, разработка более развитых ядер алгоритма GIPSO, использование других стратегий миграции частиц.

Литература

1. Weise, T. Global Optimization Algorithms – Theory and Application: Ph.D. thesis / University of Kassel. — 2008.

2. Карпенко А.П., Селиверстов Е.Ю. Глобальная оптимизация методом роя частиц. Обзор // Информационные технологии, 2010, 2, с. 25-34.

3. Chu, S.-C. Intelligent parallel particle swarm optimization algorithms / S.- C. Chu, J.-S. Pan // Parallel Evolutionary Computations. — Springer, 2006. — Pp. 159–175.

4. NVIDIA. — Nvidia CUDA Programming Guide Version 3.0, 2010.

5. Li-ping, Z. Optimal choice of parameters for particle swarm optimization / Z. Li-ping, Y. Huan-jun, H. Shang-xu // Journal of Zhejiang University Science. — 2005. — 06. — Vol. 6, no. 6. — Pp. 528–534.

6. Benchmark functions for the cec'2008 special session and competition on large scale global optimization: Tech. rep. / K. Tang, X. Yao, P. N. Suganthan et al.: Nature Inspired Computation and Applications Laboratory, USTC, China, 2007.