Воевода П.Е., Зернов М.М. Способ отображения генетических алгоритмов на структуру гетерогенной вычислительной системы // Международный журнал информационных технологий и энергоэффективности. – 2017. – Т.2 №1(3) с. 42-48

Международный журнал информационных технологий и энергоэффективности

Сайт журнала: http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.023

СПОСОБ ОТОБРАЖЕНИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ НА СТРУКТУРУ ГЕТЕРОГЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Воевода П.Е., Зернов М.М.

Филиал федерального государственного бюджетного образовательного учреждения высшего образования «Национальный исследовательский университет МЭИ» в г. Смоленске, Россия (214013, г. Смоленск, Энергетический проезд, дом 1); pavel-voevoda@mail.ru

Рассмотрены основные модели параллельных генетических алгоритмов: «клиент-серверная», «модель островов» и асинхронный генетический алгоритм. Реализован параллельный вариант алгоритма на основе модели «клиент-сервер» в существующей библиотеке классов. Библиотека адаптирована для вычислений на гибридном вычислительном кластере СФ МЭИ.

Ключевые слова: параллельный генетический алгоритм, параллельные модели генетических алгоритмов, вычислительный кластер.

THE MAPPING OF GENETIC ALGORITHMS ON THE STRUCTURE OF HETEROGENEOUS COMPUTING SYSTEM

Voevoda P.E., Zernov M.M.

The Smolensk brunch of the Moscow Power Engineering Institute (National Research University), Russia (214013, Smolensk, Energetichesky proezd, 1*); pavel-voevoda@mail.ru*

Had been considered main models of parallel GAs, including master-slave, multiple-population coarse-grained GAs and asynchronous GA. Had been implemented parallel version of GA using master-slave model in existing GA library. It may be used for calculating on computer cluster in Smolensk Branch of MPEI.

Key words: parallel genetic algorithm, models of parallel GAs, computer cluster.

Рассмотрим основные способы параллельной реализации генетических алгоритмов. Из наиболее распространенных можно выделить модель «клиент-сервер», «модель островов» и асинхронные генетические алгоритмы.

Наиболее простой в реализации вариант - это модель «клиент-сервер». В классическом варианте данная модель предполагает, что есть основной цикл построения популяции, который соответствует процессору (потоку) «серверу». Все остальные процессоры (потоки) являются клиентскими и реализуют вычисления.

Данная модель требует наименьших изменений в существующей версии программного обеспечения, реализующего последовательный ГА и дает неплохие результаты.

При этом затраты по вычислению значений фитнесс-функций равномерно распределяются по всем процессорам, для которых используется одна и та же фитнесс-функция. Поэтому для n особей и P (одинаковых) процессоров мы каждому процессору относим $\frac{n}{P}$ особей. Значения фитнесс-функции вычисляются соответствующими (рабочими) процессорами и посылаются в один процессор (хозяин), который собирает всю информацию, обрабатывает и передает ее снова рабочим процессорам. Процессор «хозяин» имеет информацию о значениях фитнесс-функции для всех особей и может генерировать следующее поколение на этой основе.

В конце каждого из этапов помещаются точки синхронизации. Когда процессор-хозяин достигает эти точки, он переходит в режим ожидания, пока все рабочие процессоры не закончат свои задания, что гарантирует глобальную корректность алгоритма. При этом работа между процессором-хозяином и рабочими распределяется следующим образом.

Каждый «рабочий» принимает задание от «хозяина» и определяет значение фитнесс-функции для особей, полученный результат посылает хозяину и ожидает следующего задания. Поскольку размер популяции много больше числа процессоров, достигается хороший баланс в загрузке процессоров.

При использовании модели «клиент-сервер» окончательные результаты (поиска решения данной задачи) близки к тем, что получены на однопроцессорной компьютерной системе с использованием аналогичного алгоритма. Качество решения при этом не теряется и в большинстве случаев несколько улучшается, а время его поиска существенно сокращается. В целом данная модель позволяет быстро (с минимальными модификацями) выполнить параллелизацию ГА и дает, прежде всего, ускорение процесса поиска решения. Данную модель не сложно реализовать в локальной сети с использованием технологии сокетов [1].

«Модель островов» используется в распределенных генетических алгоритмах, где каждая подпопуляция развивается на своем "острове". Между островами производится (достаточно редко) обмен лучшими особями. Эта модель может быть реализована в распределенной памяти компьютерной системы, имеющей MIMD-архитектуру. Преимущество распределенных генетических алгоритмов в том, что они работают быстрее даже на однопроцессорных компьютерных системах вследствие лучшей структуризации. Причина заключается в том, что число вычислений сокращается благодаря распределению поиска в различных областях пространства решений [4].

В целом распределенные параллельные генетические алгоритмы более целесообразно использовать для повышения качества решения, в том случае, если его не удается достичь с помощью обычного (последовательного) генетического алгоритма. При этом мощность каждой подпопуляции должна быть достаточно большой, чтобы не происходило "вырождение особей", которое часто ведет к преждевременной сходимости к локальному экстремуму. Увеличение быстродействия по сравнению с последовательным генетическим алгоритмом также имеет место, но может быть меньше, чем у модели «клиент-сервер». Для повышения быстродействия в этом случае мощность подпопуляций должна быть небольшой, что существенно увеличивает быстродействие (ценой потери качества решения).

Асинхронный генетический алгоритм не имеет явного разбиения эволюции на поколения, а процесс развития реализуется "по событию" в виде непрерывного потока событий отбора, кроссинговера, мутации и т.п. Здесь вновь порожденные потомки сразу замещают (в некотором смысле худшие) особи (не дожидаясь остальных родительских пар своего поколения). Следует

отметить, что такой метод, как правило, проще в реализации (особенно аппаратной) и имеет некоторые преимущества [1].

Как отмечалось, наиболее простой способ создания параллельной модели на основе уже существующей - использование модели «клиент-сервер». Здесь нет необходимости постоянно проводить контроль динамики популяции, а результат параллельной реализации будет наиболее близок к последовательной версии генетического алгоритма. Таким образом, модель будет показывать приблизительно одинаковые результаты независимо от платформы и типа ускорения. В данном случае необходимо создать планировщик задач, который будет распределять вычисления одного этапа по потокам или процессорам. Рассмотрим более подробно реализацию алгоритма на основе данной модели.

Отображение структуры генетического алгоритма на вычислительную систему на основе модели «клиент-сервер»

Рассмотрим наиболее простой способ реализации параллельного генетического алгоритма, при котором параллелизм будет использоваться за счет разделения этапов на несколько процессов. На каждом этапе получения потомков с помощью разделения на $m/2$ потоков (где m – число потомков в поколении, причем считаем, что в создании потомка участвует 2 родителя) основного процесса выполнения. Каждый поток проводит скрещивание двух особей, затем при вычислении функции приспособленности каждый узел вычислительной системы отыскивает значения F_i (функция приспособленности на i-м этапе) для порождённых на нём особей.

В таком случае можно осуществлять скрещивание и вычисление значений функции приспособленности, следовательно, и целевой функции, одновременно на нескольких потоках (вычислительных узлах). При этом необходимо выделить отдельный поток для управления работой алгоритма и распределением задач и шагов алгоритма.

Схема такого варианта параллельного варианта ГА представлена на рисунке 1.

Выполнение генетических операторов и вычисление целевой функции происходит в клиентских потоках. По достижении условия сходимости или заданного количества итераций в серверной части осуществляется выбор решения с наибольшим значением функции приспособленности, что соответствует наименьшему значению ЦФ.

В качестве основы воспользуемся ранее разработанной библиотекой для работы с генетическими алгоритмами. В ее основе реализована последовательная версия генетического алгоритма. В дальнейшем, была реализована возможность параллельного выполнения команд с использованием OpenMP. Для возможности вычислений с использованием графического процессора были написаны реализации отдельных операторов с использованием OpenCL [3].

При реализации модели «клиент-сервер» и доработке существующей библиотеки необходимо вынести параллельную обработку на уровень выше.

Для проектирования приложения, использующего параллельную обработку данных применяются соответствующие паттерны проектирования. Это обусловлено сложностью распределения команд по потокам и их отделением друг от друга. Для управления всеми потоками должен быть специальный планировщик. Этими задачами занимается паттерн проектирования «активный объект» (ActiveObject), поэтому разумно использовать его базовую версию для адаптации существующей библиотеки под многопоточную работу.

Основная задача - возможность параллельно выполнять генетические операторы. Каждый оператор представлен уже существующим классом, являющимся наследником COperator, причем модификаций этого класса почти не планируется.

Структура данного паттерна после реализации выглядит следующим образом (рисунок 2).

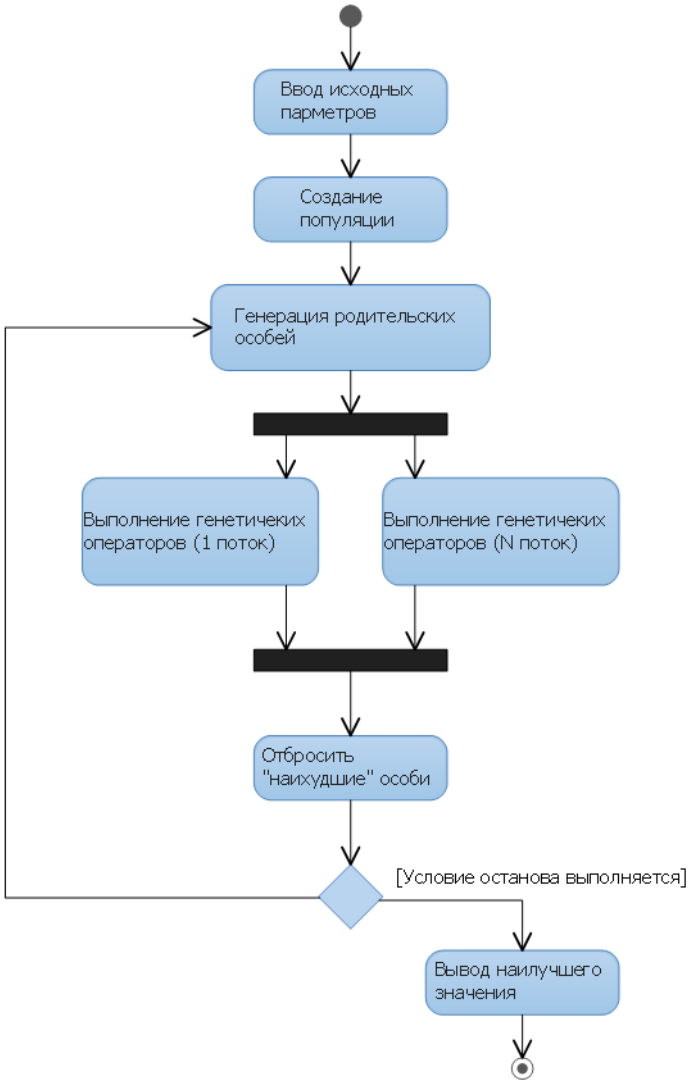

Рисунок 1 - Схема параллельного генетического алгоритма на основе модели «клиент-сервер»

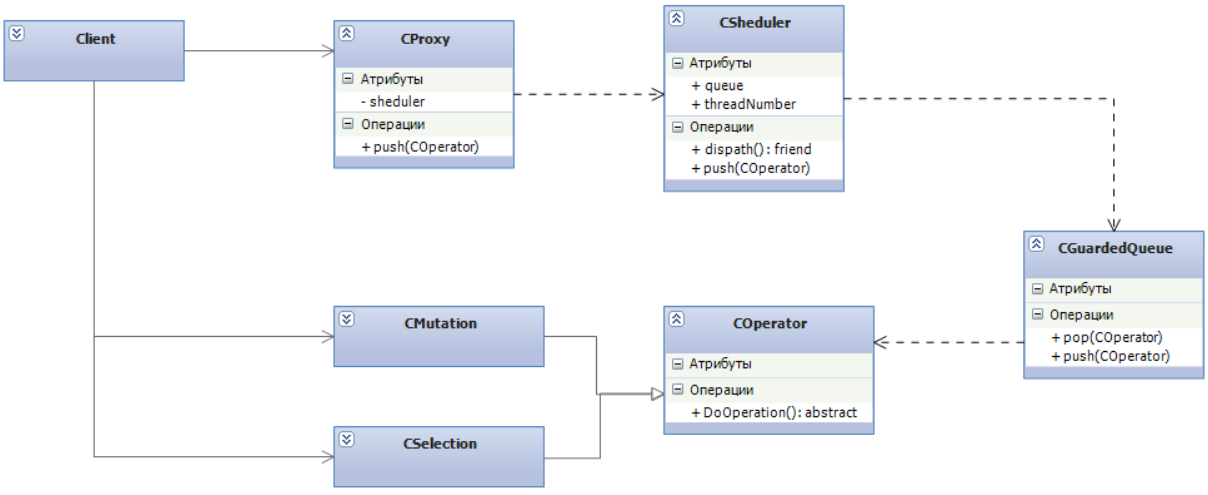

Рисунок 2 - Диаграмма классов для паттерна ActiveObject

Клиенту доступны классы-наследники класса COperator, которые переопределяют виртуальный метод DoOperation(), а также CProxy, с помощью которого будет происходить взаимодействие с создаваемой системой.

Класс GuardedQueue – это потоко-защищённая очередь, которая реализует методы push и pop. Очередь оперирует объектами подклассов класса COperator.

Основным элементом паттерна ActiveObject является класс Sheduler или «планировщик». Именно логика его построения определяет эффективность всей системы.

В конструкторе создается определенное количество потоков, каждый из которых не заканчивает свою работу до завершения работы приложения.

В каждом из созданных потоков запускается функция dispatch(), которая в бесконечном цикле выталкивает верхний элемент из очереди задач (генетических операторов) и запускает его метод DoOperation ().

Эффективность работы паттерна определяется тем, что для каждой задачи не приходится создавать свой поток (процесс создания потока, как известно, занимает продолжительное время), а также тем, что потоки «не простаивают», если есть задачи в очереди.

Применение реализации генетических алгоритмов на вычислительном кластере

Разработка параллельной версии генетического алгоритма производится с учетом особенностей и возможностей гибридного вычислительного кластера филиала НИУ МЭИ в г. Смоленске и адаптирована для использования на нем. Кластер состоит из двух вычислительных узлов, узла управления, необходимых интерфейсных (связных) компонентов.

Каждый вычислительный узел (ВУ) включает в себя два 10-ядерных процессора, оперативную память объемом 64 Гбайт, два твердотельных накопителя SSD с интерфейсом SATA III, организованных в массиве RAID 1, объемом 240 Гбайт, а также по два интерфейсных канала, обеспечивающих подключение вычислительных узлов к внутренней сети кластера.

Кроме того, ВУ 1 содержит две видеокарты на базе GPU NVIDIA, а также два дисковых накопителя HDD объёмом 1 Тбайт с интерфейсом SATA III, организованных в виде массива RAID 1. Вычислительный узел 2 содержит две видеокарты на базе GPU AMD.

Узел управления содержит два 8-ядерных процессора Intel Xeon E5-2620 v4, оперативную память объемом 64 Гбайт стандарта DDR4, девять дисковых накопителей: SSD-накопитель объёмом 256 Гбайт с интерфейсом PCIe 3.0×4 в стандарте M.2; два SSD-накопителя объёмом 250 Гбайт с интерфейсом SATA III в массиве RAID 1; два HDD-накопителя объёмом 2 Тбайт с интерфейсом SATA III в массиве RAID 1; четыре HDD-накопителя объёмом 3 Тбайт с интерфейсом SATA III, размещенных в массиве RAID 10 с объёмом 6 Тбайт. Наличие нескольких дисковых накопителей позволяет, в том числе, создавать на управляющем узле требуемую систему виртуальных машин [2].

Обработка данных может осуществляться посредством как центральных процессоров на основе технологий MPI и Open MP, так и графических на основе технологий OpenCL и CUDA.

После реализации библиотеки для работы с генетическими алгоритмами с применением технологий OpenMP и OpenCL [3] для локального распараллеливания отдельных этапов алгоритма в ней был реализован параллельный генетический алгоритм на основе модели «клиент-сервер». В качестве технологий параллелизма для реализации модели «клиент-сервер» были использованы POSIX threads и Open MPI. Основным этапом перед использованием данной библиотеки в вычислительном эксперименте является ее сборка.

Для сборки библиотеки используется Makefile, представленный в листинге. Данный Makefile адаптирован под сборку проекта с использованием OpenCL и OpenMPI. Файл помещается в папку с исходниками библиотеки, а сборка осуществляется выполнением команды *make*.

Листинг1 - Текст Makefile

```
TARGET=$(shell basename `pwd`)
SOURCES=$(wildcard *.cpp)
OBJECTS=$(SOURCES:%.cpp=%.o)

CXXFLAGS=-std=gnu++11
# Check for 32-bit vs 64-bit
PROC_TYPE = $(strip $(shell uname -m | grep 64))

# Linux OS
LIBS= -I/usr/local/cuda-8.0/targets/x86_64-linux/include/ -lOpenCL
CFLAGS+= -I/usr/local/cuda-8.0/targets/x86_64-linux/include
LDFLAGS += -L/usr/local/cuda-8.0/targets/x86_64-linux/lib/x86
LDLIBS += -lOpenCL
ifeq ($(PROC_TYPE),)
        CFLAGS+=-m32
else
        CFLAGS+=-m64
endif

all: $(TARGET)

$(OBJECTS): $(SOURCES)
$(TARGET): $(OBJECTS)
        $(CXX) -o $(TARGET) $(LDFLAGS) $(OBJECTS) $(LOADLIBES) $(LDLIBS) $(CFLAGS)
clean:
        $(RM) $(OBJECTS) $(TARGET)
.PHONY: all clean
```

Данный файл определяет следующую конфигурацию:

- в качестве целевой используется текущая папка;
- исходные файлы в данной папке имеют расширение cpp;
- компилятор использует поддержку стандарта C++11;
- заданы пути для использования и поддержки OpenCL;
- заданы цели для сборки проекта.

В результате выполнения будет собрана библиотека, которую можно подключать к пользовательским проектам в качестве статической библиотеки.

В статье предложен способ отображения генетического алгоритма на структуру параллельной гетерогенной вычислительной системы, отличающийся сочетанием модели параллельной реализации генетического алгоритма «клиент-сервер» с использованием шаблона объектно-ориентированного проектирования «активный объект» и разработанной ранее модели параллельного выполнения генетических операторов с использованием технологии OpenCL. Предложенный способ позволяет одновременно задействовать в ходе работы генетического алгоритма как технологии, задействующие многопроцессорные вычисления (POSIX threads и OpenMPI) и графические вычисления (OpenCL), получая реализацию генетического алгоритма результат которого соответствует последовательной версии с точностью до реализации применяемых случайных последовательностей.

Способ предусматривает наличие планировщика потоков на основе паттерна ActiveObject, управляющего общим ходом вычислений и отслеживающего динамику популяции, управляемых потоков соответствующих отобранным парам родительских особей работающих на основе

технологий POSIX threads и OpenMPI, а также параллельную реализацию самих генетических операторов на основе SIMD-модели на базе технологий OpenMP или OpenCL.

Способ нашёл отражение в разработанной библиотеке классов для реализации генетических алгоритмов, которая была собрана для дальнейших исследований на гибридном вычислительном кластере СФ МЭИ и ориентирована на его структуру.

Список литературы

1. Скобцов Ю.А., Сперанский Д.В. Эволюционные вычисления: учебное пособие.-М.: Национальный Открытый Университет “ИНТУИТ”, 2015. –331с
2. Борисов В. В., Зернов М. М., Федулов А. С., Якушевский К. А. Исследование характеристик гибридного вычислительного кластера // Системы управления, связи и безопасности. 2016. №4. С. 129-146.
3. Воевода П.Е., Зернов М.М. Применение технологий параллельного программирования в библиотеке для реализации генетических алгоритмов // Информационные технологии, энергетика и экономика XIII международная научно-техническая конференция студентов и аспирантов. - Смоленск: 2016 г., стр. 225-232.
4. Курейчик В.М., Кныш Д.С. Параллельный генетический алгоритм. Модели и проблемы построения // V Международная научно-практическая конференция "Интегрированные модели и мягкие вычисления в искусственном интеллекте" - Коломна: 2009 г.

References

1. Skobcov YU.A., Speranskij D.V. EHvolyucionnye vychisleniya: uchebnoe posobie.-M.: Nacional'nyj Otkrytyj Universitet “INTUIT”, 2015. –331 p.
2. Borisov V. V., Zernov M. M., Fedulov A. S., YAkushevskij K. A. Issledovanie harakteristik gibridnogo vychislitel'nogo klastera // Sistemy upravleniya, svyazi i bezopasnosti. 2016. №4. pp. 129-146.
3. Voevoda P.E., Zernov M.M. Primenenie tekhnologij parallel'nogo programmirovaniya v biblioteke dlya realizacii geneticheskih algoritmov // Informacionnye tekhnologii, ehnergetika i ehkonomika XIII mezhdunarodnaya nauchno-tekhnicheskaya konferenciya studentov i aspirantov. - Smolensk: 2016, pp. 225-232.
4. Kurejchik V.M., Knysh D.S. Parallel'nyj geneticheskij algoritm. Modeli i problemy postroeniya // V Mezhdunarodnaya nauchno-prakticheskaya konferenciya "Integrirovannye modeli i myagkie vychisleniya v iskusstvennom intellekte" - Kolomna: 2009.