УДК 658.512.22

Адаптация генетических алгоритмов для выполнения в эластичной вычислительной среде с учетом особенностей их применения в САПР

А.А. Шлепнев

Институт проблем проектирования в микроэлектронике РАН,

shleps@ippm.ru

***Аннотация* — В статье рассматриваются вопросы эффективного использования ресурсов облачных вычислительных систем для решения ряда задач из области автоматизированного проектирования, в основе которых лежат генетические алгоритмы. Предлагаются два способа организации параллельного выполнения генетических алгоритмов в облачной инфраструктуре: с помощью промышленных средств GRID и на базе собственной архитектуры эластичного приложения.**

***Ключевые слова* — эластичные вычисления, облачные вычисления, grid-технологии, генетический алгоритм, оптимизация, параллельные вычисления, инфраструктура.**

I. Введение

В области систем автоматизированного проектирования (далее - САПР) интегральных схем для таких задач, как оптимизация размещения блоков, широко применяются алгоритмы ограниченного перебора и, в частности, генетические алгоритмы (далее — ГА) [1], требующие значительного количества вычислительных ресурсов. В настоящее время крупным поставщиком вычислительных ресурсов для ресурсоёмких задач стали облачные центры обработки данных. Имеется тенденция использовать их не только для веб-приложений или суперкомпьютерных вычислений, но и для полного погружения маршрута проектирования изделий микроэлектроники [2], [3]. Одним из фундаментальных свойств облачных вычислительных систем является эластичность [4], которая позволяет использовать вычислительные ресурсы в экономически эффективной манере [5]. Однако, построенные по традиционным схемам приложения, в том числе промышленные средства САПР, не в состоянии воспользоваться этим свойством инфраструктуры, что резко ограничивает доступные приложению вычислительные мощности, усложняет и удорожает их использование. Таким образом, становится актуальной задача адаптации прикладного программного обеспечения для его эффективного выполнения в облачной инфраструктуре.

II. Архитектура облачных вычислений и эластичность

Национальным институтом по стандартам и технологиям США (НИСТ) свойство эластичности определяется, как возможность оперативно или автоматически масштабировать предоставляемые абоненту ресурсы, резервируя или освобождая компоненты инфраструктуры облака сообразно текущим потребностям. В случае облака типа IaaS такими ресурсами являются процессорные ядра, оперативная память и дисковое пространство, выделяемые в рамках одной или нескольких виртуальных машин. Автоматическое масштабирование в данном случае выражается в развертывании без вмешательства человека очередной виртуальной машины с базовыми настройками операционной системы. Дальнейшее приспособление вновь развернутой машины к потребностям абонента требует участия пользователя. Другой вид облаков — PaaS, предполагает полностью автоматическое конфигурирование компонентов информационного пространства среднего уровня: экземпляров операционных систем, сетей, систем хранения данных, систем управления базами данных (СУБД) и другого системного ПО. Наконец, в облаках типа SaaS пользователю предоставляется конечное приложение, которое обладает всеми ключевыми особенностями облака, в том числе эластичностью [5]. В настоящее время значительная часть SaaS-приложений относится к веб-приложениям: системы взаимодействия с клиентами, форумы, коммуникационные приложения для мобильных абонентов и т.д. Основная масса веб-приложений строится на базе типовых компонентов, таких как СУБД, интерпретаторы управляющего языка с библиотеками и шаблонами, веб-сервера и балансировщики нагрузки. В отличие от веб-приложений, приложения вычислительной направленности, полностью приспособленные к работе в облаке, должны разрабатываться с помощью специальных инструментов, привязанных к архитектуре конкретного облака (Microsoft Azure, RedHat OpenShift и др.) Проистекающие отсюда дополнительные сложности в разработке приводят к тому, что количество таких приложений невелико, несмотря на то, что мощности облачных центров

обработки данных как нельзя лучше соответствуют потребностям систем высокопроизводительных параллельных или распределенных вычислений.

Существуют примеры использования традиционного инструментария высокопроизводительных вычислений в рамках облачных сервисов [6], [7], однако данные системы не обладают свойством эластичности, поскольку в классических библиотеках для распределенных вычислений, таких как OpenMP, MPI или PVM количество исполнителей, между которыми распараллеливается задача, за очень редким исключением фиксируется при запуске приложения. Помимо невозможности воспользоваться преимуществом по экономической эффективности в полном объеме, использование систем распределенных вычислений традиционной архитектуры, работающих независимо от инфраструктуры облачного провайдера, вынуждает пользователя заниматься вручную весьма далекими от предметной области специфическими задачами: настройкой операционной системы и программного окружения. Таким образом, построение эластичных приложений для распределенных вычислений в облачной инфраструктуре является целесообразным с экономической и временной точки зрения, но требует специализированного подхода к разработке приложения.

До некоторой степени воспользоваться преимуществами облачных технологий для вычислений можно в облачной модели PaaS, если в предоставляемую абонентам платформу включена какая-либо система GRID-вычислений [8]. Такая конфигурация позволяет достигать эластичности на уровне пакета задач — хотя каждая задача всё ещё ограничена ресурсами отдельного виртуального узла, для решения пакета задач доступны ресурсы всего кластера целиком. Также автоматизируется процесс развертывания новых виртуальных машин в облаке и введения их в вычислительный GRID-кластер. К недостаткам такого решения относится необходимость вручную регулировать размер вычислительной инфраструктуры, сложности с возвратом неиспользуемых мощностей в облако и отсутствие в GRID-инфраструктуре поддержки обмена данными между отдельными задачами, в отличие от механизмов, предусмотренных в классических реализациях параллельных вычислений типа MPI.

III. Декомпозиция генетического алгоритма

Классический генетический алгоритм включает в себя следующие этапы:

1) генерация начальной популяции;
2) выбор особей-родителей;
3) применение оператора скрещивания;
4) применение оператора мутаций;
5) вычисление фитнесс-функции;
6) замена текущей популяции поколением потомков;
7) проверка выполнения критерия остановки и переход к пункту 2.

Традиционно, распараллеливанию среди большого количества исполнителей подлежит вычисление фитнесс-функции, что особенно эффективно, если оно занимает значительное время по сравнению с эволюционными операциями.

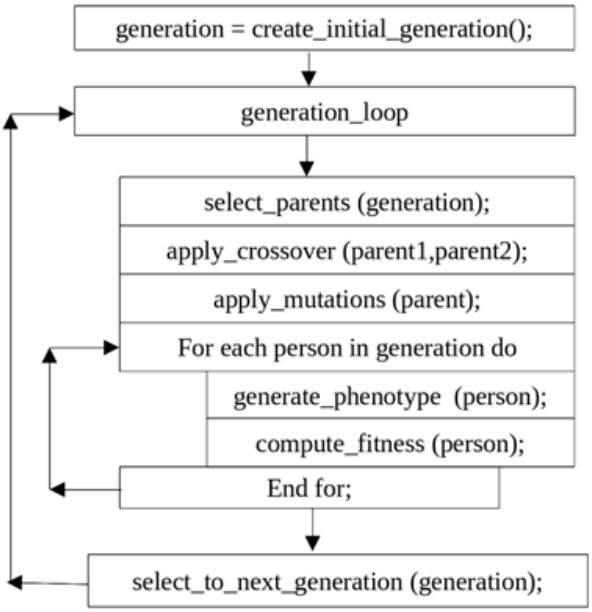

Рис. 1. Структура последовательного ГА

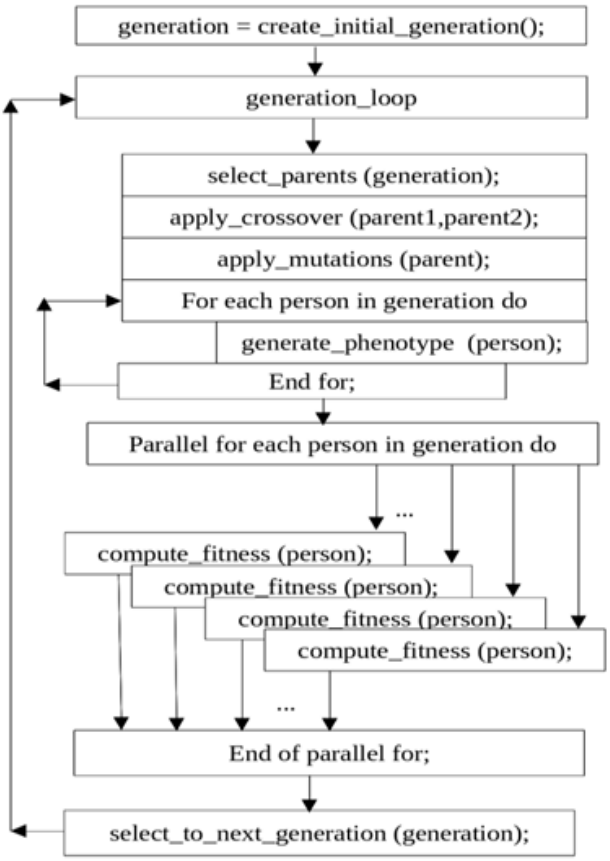

Рис. 2. Структура параллельного ГА

На рис. 1 и 2 с помощью псевдокода показана организация последовательных и параллельных фрагментов в различных вариантах алгоритма. Более сложные модификации, например, островные генетические алгоритмы, предусматривают одновременное эволюционирование нескольких популяций [9], [10], что позволяет распараллелить не только вычисление фитнесс-функции, но и эволюционные операции по количеству островов (рис. 3).

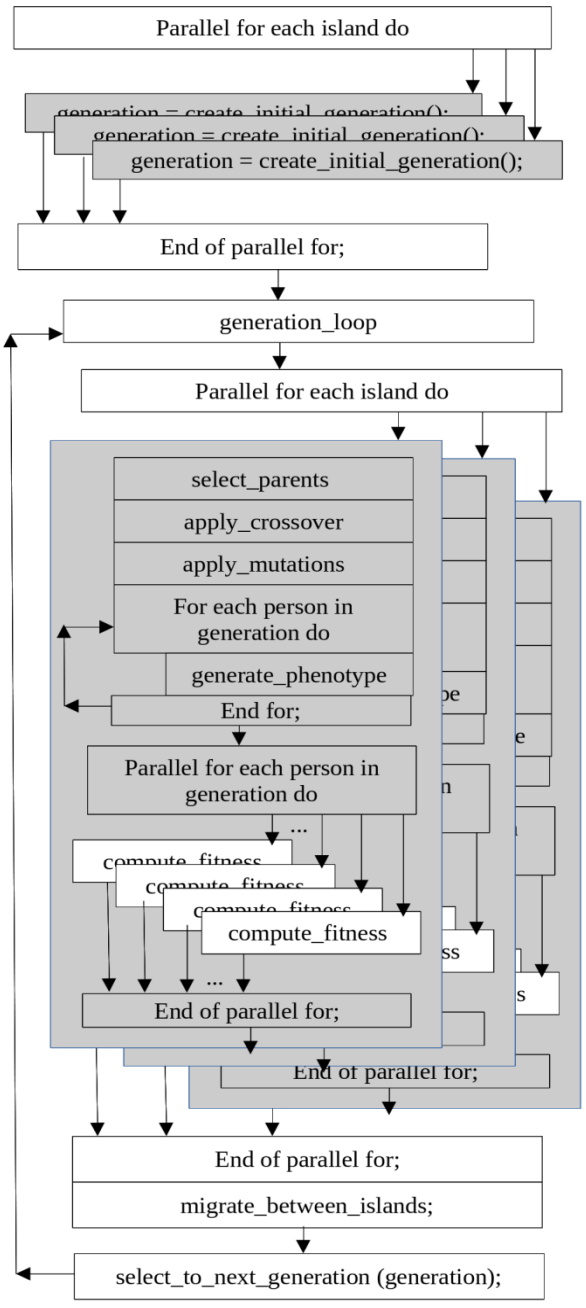

Рис. 3. Разновидность островного параллельного ГА

IV. Особенности ГА применительно к САПР

В области САПР микроэлектроники генетические алгоритмы обычно применяются для оптимизации проектных решений [1], что определяет следующие особенности реализации:

1) генотипом особи является описание параметров проектного решения;

2) для вычисления фитнесс-функции могут задействоваться промышленные средства САПР, внешние по отношению к собственно генетическому алгоритму: симуляторы, трассировщики и. т.д.;

3) как следствие - фенотипом особи является проектное решение в формате, доступном для используемых промышленных средств САПР;

4) как следствие — обязательно присутствует допускающий параллельную обработку этап формирования фенотипа на основе генотипа;

5) вычисление фитнесс-функции может быть очень сложным, длительным процессом, распараллеливание которого может быть ограничено наличием лицензий на используемые промышленные средства САПР;

6) как следствие - может присутствовать этап грубой, но быстрой оценки фитнесс-функции с помощью эвристик для отбраковывания явно дефектных особей, который также допускает распараллеливание.

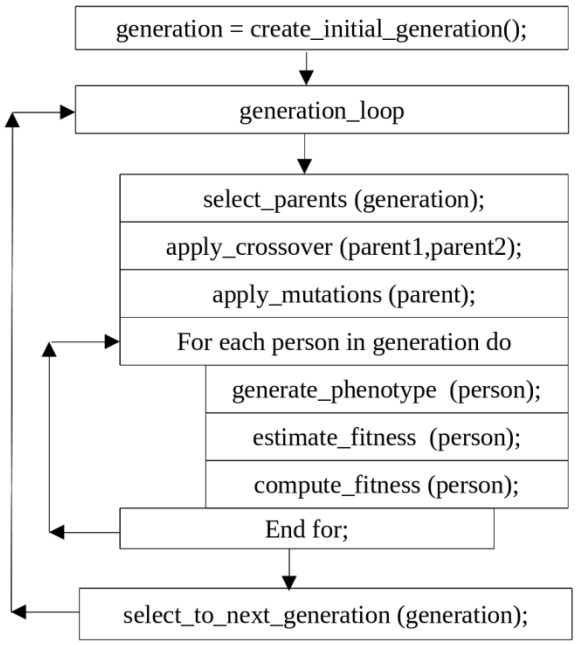

Рис. 4. Этапы ГА с учетом особенностей применения в САПР

V. Реализация параллельного ГА на базе инфраструктуры PaaS с GRID

В парадигме GRID единицей диспетчеризации является процесс ОС, поэтому параллельный алгоритм приходится разбивать на два или более приложений, одно из которых (менеджер) выполняет подготовку исходных данных для параллельной части, сбор

результатов параллельных вычислений и другую последовательную обработку, а остальные (подпрограммы) - отвечают за функции, подлежащие распараллеливанию. На рис. 5 и 6 показаны некоторые возможные варианты распределения псевдокода параллельного ГА по приложениям и маршруты передачи управления между ними. Передача управления от последовательной части алгоритма к параллельной осуществляется вызовом клиентской части диспетчера GRID в виде команды ОС bsub или qsub с соответствующими параметрами командной строки или аналогичным вызовом функции API.

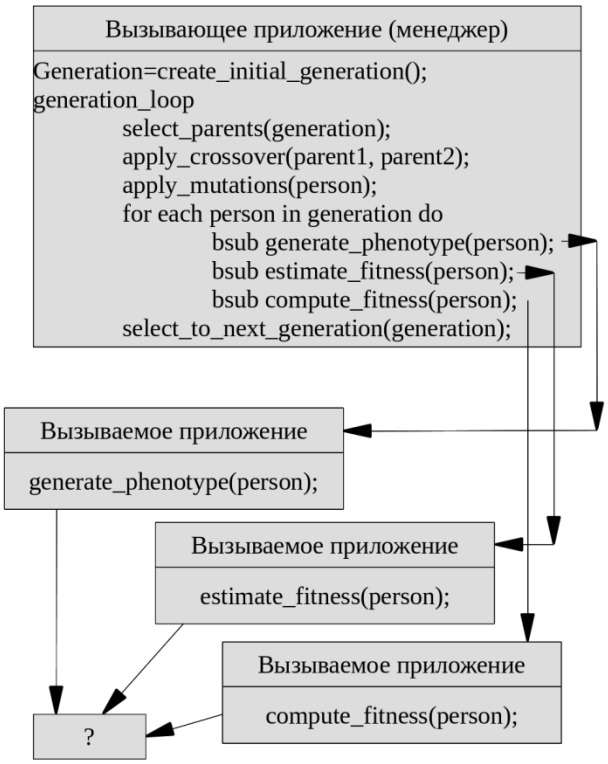

Рис. 5. Наивная реализация ГА в GRID

Далее, диспетчер GRID асинхронно запускает на доступных вычислительных узлах приложения, указанные в качестве параметра sub-команды. При этом очередность запуска приложений-подпрограмм и тем более очередность их завершения носят в значительной степени случайный характер. Это порождает ситуации, когда последующая часть алгоритма, например, вычисление фитнесс-функции для некоторой особи, получит управление прежде, чем закончится генерация её фенотипа, что приведёт к краху алгоритма. Таким образом приложение-менеджер должно иметь средства, чтобы избежать ситуации гонок и обеспечить контроль за возвратом управления от параллельной части к последовательной. Штатные средства GRID предусматривают только простейшие механизмы уведомления о завершении подзадач, такие как электронная почта или опрос состояния очереди (через API или командами типа qstat или qmon), что крайне усложняет реализацию контроля над очередью задач. Другое решение заключается в объединении нескольких этапов алгоритма в маршрут, представляющий собой командный файл ОС, и дальнейшее использование маршрута в качестве единицы диспетчеризации. Это гарантирует, что операции генерации фенотипа, грубой оценки фитнесс-функции и точного вычисления фитнесс-функции будут выполняться в правильном порядке без усложнения приложения-менеджера.

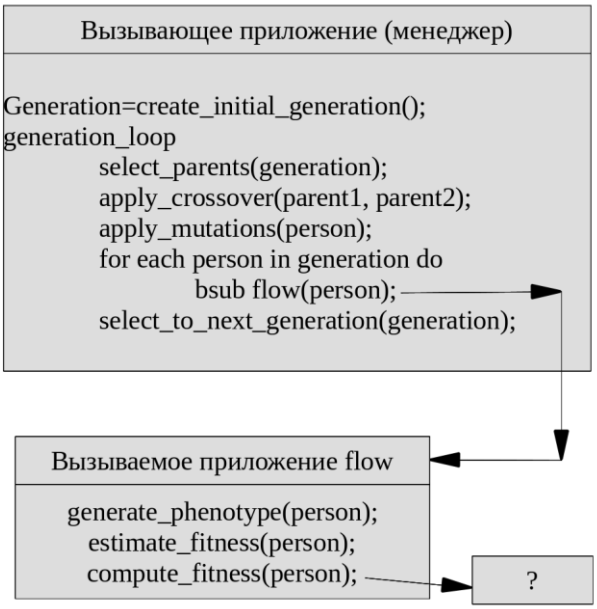

Рис. 6. Реализация ГА в GRID с использованием маршрута

К сожалению, полностью избежать проблемы синхронизации не удастся — для осуществления следующего шага цикла смены поколений необходимо полностью закончить этап вычисления фитнесс-функции для всех особей, поскольку эти значения будут необходимы для операции выбора родителей. Для этого приложение-менеджер должно дождаться завершения выполнения сторонних программ и получить от них результаты, при том, что диспетчер GRID, как правило, не предоставляет функций по обмену данными даже между единицами диспетчеризации, не говоря уже о составных частях маршрута. В нашем случае подпрограммам генетического алгоритма и внешним средствам САПР необходимо обмениваться следующими данными об эволюционирующих особях: генотип, фенотип, значение фитнесс-функции, а также служебной информацией для селективной части ГА (возраст, принадлежность к острову). Использование внешних средств САПР заставляет организовывать обмен посредством работы с файлами, находящимися на системе хранения данных (далее - СХД), общей для всех вычислительных узлов GRID, то есть сетевой файловой системе типа NFS, или, в редких случаях, на кластерной файловой системе типа Ceph, Lustre или GlusterFS. Реализовать поверх файловой системы стандартные примитивы синхронизации типа

семафоров и мьютексов, которые были бы независимы от типа применяемой файловой системы, устойчивы к гонкам, и не вызывали бы серьезную деградацию производительности невозможно, поскольку требование устойчивости к гонкам означает использование синхронных режимов работы файловой системы и отключение кешей на всем пути следования данных, т. е. и на вычислительных узлах и на файл-сервере. Кроме того, выполнение этих условий невозможно проконтролировать при работе в облачной инфраструктуре.

В нашем случае устойчивости к гонкам не требуется, поскольку имеет место однонаправленная передача данных от параллельных участков алгоритма к последовательному, а параллельные фрагменты не обмениваются информацией между собой. Реализовать ожидание завершения обработки особей можно с помощью распространенного механизма символических ссылок. Перед отправкой в GRID задания по обработке особи с идентификатором PersonID создается флаг с помощью команды «ln -s NextGen PersonID» или ее API-аналога. В ходе выполнения маршрута флаг сбрасывается с помощью команды удаления файла «rm PersonID». Чтобы приступить к следующему шагу эволюционного цикла менеджер ожидает завершения обработки всех особей, проверяя условие отсутствия символических ссылок на несуществующий файл NextGen (аналог команды ls -l | grep NextGen). Следует отметить, что в наиболее распространенном случае, когда для обмена данными используется сетевая файловая система NFS, приложение-менеджер, выполняющееся на одном вычислительном узле, узнает о том, что файл-флаг был удален маршрутом обработки особи, выполнявшимся на другом вычислительном узле, только после того, как будет обновлен кеш NFS-клиента. В случае традиционной для САПР Linux-среды при настройках по умолчанию это произойдет через 60 секунд. Такие задержки резко ограничивают эффективность работы с множеством небольших задач. Для преодоления этого недостатка может применяться либо тюнинг режимов работы NFS, оказывающий соответствующее негативное влияние на производительность СХД, либо создание отдельного программного средства, отвечающего за блокировки и обмен, которое может быть построено на СУБД, или непосредственно на сетевых протоколах более низкого уровня.

VI. Реализация параллельного ГА в виде эластичного распределенного приложения

В [12] описана архитектура информационной системы, предназначенной для передачи данных, а также организации распределенных вычислений в различных типах инфраструктуры, включая облако, GRID, их комбинацию, либо независимые вычислительные узлы. Отличительной особенностью данной архитектуры является возможность динамического изменения количества параллельно работающих исполнителей, что позволяет отказываться от избыточной вычислительной мощности с её возвращением в облако для достижения максимальной эластичности. Текущая реализация обеспечивает асинхронный обмен командами, параметрами команд и результатами выполнения команд между несколькими экземплярами одного приложения или несколькими различными приложениями, для чего в распоряжении программиста имеются примитивы QueueCommand и QueueFunction. Сами приложения реализуются в виде интерпретатора очереди команд, при этом прикладная часть приложения размещается в обработчиках команд.

Для реализации параллельных ГА данная архитектура была дополнена примитивами синхронизации, функционально подобными токенам, применяющимся в параллельных потоковых вычислительных системах [13]. Операция CreateWaiter регистрирует в приложении примитив-«официант» (англ. waiter), который можно связывать с одной или несколькими блокировками и который выполняет «заказ», т. е. активирует заданный набор команд (обработчик) при снятии всех связанных с ним блокировок, после чего самоудаляется. Операция CreateLock регистрирует в приложении блокировку с указанным идентификатором, которая может быть применена к одному или нескольким «официантам». Операция LockWaiter связывает указанного «официанта» с указанной блокировкой. Операция Unlock снимает указанную блокировку со всех «официантов», имеющихся в приложении и удаляет её из приложения.

При выполнении распределенного приложения, структура которого приведена на рис. 7, в конфигурации с более чем одним исполнителем, операция QueueCommand(*::RunFlow:PersonID) ставит задачу на запуск маршрута обработки особи в общую очередь приложения посредством встроенного механизма передачи сообщений. Далее задача переадресуется одному из исполнителей согласно выбранной политике балансировки нагрузки: например, «наименьшая длина очереди команд», «наименьшее расчетное время ожидания в очереди» и т.п. Вызов Unlock(PersonID) по завершении обработки особи также порождает сообщения между исполнителями, в результате чего снимаются блокировки, препятствующие переходу к следующему шагу цикла смены поколений. В текущей реализации накладные расходы, обусловленные интерпретацией элементов очереди команд, замером производительности исполнения, балансировкой загрузки исполнителей и т.д. составляют 0.6-2 мсек на команду для простых команд. В случае команд с большим количеством параметров время может увеличиваться до десятков и сотен миллисекунд.

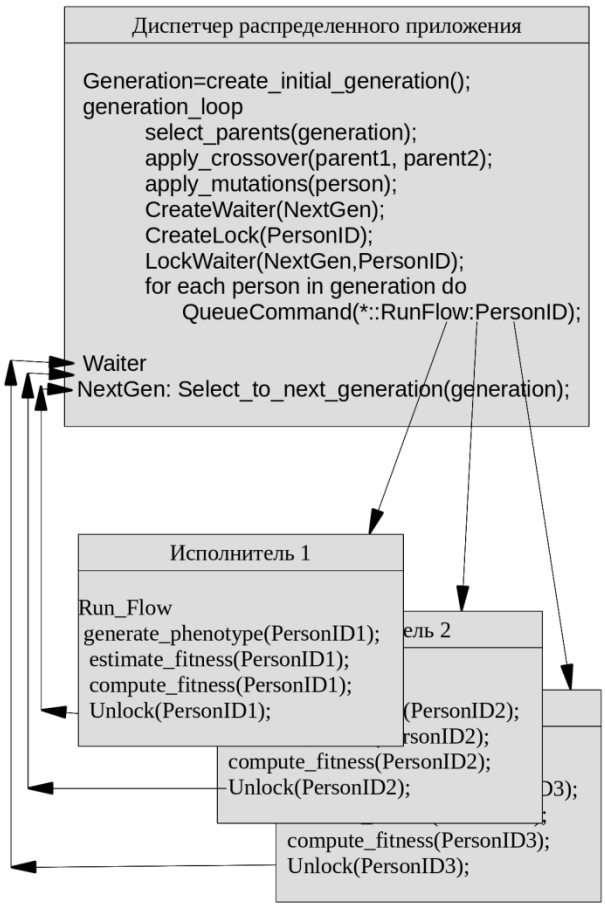

Рис. 7. Использование примитивов синхронизации для реализации параллельного ГА

Латентность встроенного механизма обмена сообщениями между исполнителями сильно зависит от загрузки узла, отвечающего за балансировку загрузки. Для информационной инфраструктуры, построенной на базе типовых компонентов (локальная коммутируемая сеть Ethernet 1ГБит/с) время отправки сообщения стабильно ниже 1 мсек, время эхо-ответа же, при тестах варьировалось 10 до 300 мсек. Это на порядки лучше результатов, достигаемых с помощью NFS и сравнимо с производительностью промышленных средств GRID, но далеко от значений, характерных для телекоммуникационных приложений, работающих в такой же инфраструктуре.

VII. Заключение

Для реализации параллельных ГА в современной облачной инфраструктуре предложены два метода. Более универсальный метод основывается на использовании только серийного управляющего ПО, но создает неоправданно высокую нагрузку на СХД и сопряжен со значительными задержками, что ведет к ограничению эффективности. Второй способ потенциально позволяет использовать свойство эластичности в полном объеме и не создает паразитной нагрузки, но требует особой архитектуры построения приложения и специализированного окружения (специально настроенного диспетчера GRID-вычислений).

В настоящее время идет работа над совершенствованием механизма управления очередью команд эластичного приложения, что должно снизить латентность обмена сообщениями до значений, характерных для применяемого оборудования. В перспективе планируется разработка GRID-диспетчера, реализующего возврат неиспользуемых ресурсов в облако, и реального приложения, использующего ГА для синтеза.

Литература

[1] Ерошенко И.Н. Планирование кристалла СБИС с учетом энергопотребления // Сб. трудов МЭС-2012. МЭС-2012. Россия, Москва, октябрь 2012. с. 257-262.

[2] URL: http://www.idexpert.ru/news/4972/ (дата обращения: 26.02.2016).

[3] URL: http://www.pcweek.ru/its/article/detail.php?ID=175179 (дата обращения: 26.02.2016).

[4] S. Dustdar et al. Principles of Elastic Processes // IEEE Internet Computing, vol. 15, no. 5, 2011.

[5] Mell P., Grance T. The NIST Definition of CloudComputing // NIST, Special Publication 800-145.

[6] URL: http://buildacloud.org/blog/181-high-performance-computing-and-cloudstack.html (дата обращения: 26.02.2016).

[7] URL: http://cyclecomputing.com/blog/cyclecloud-50000-core-utility-supercomputing/ (дата обращения: 26.02.2016).

[8] URL: http://www.teraproc.com/front-page-posts/cluster-as-a-service-for-hpc/ (дата обращения: 26.02.2016).

[9] Иванов Д.Е., Чебанов П.А. Взаимодействие компонент в распределенных генетических алгоритмах генерации тестов // Наукові праці Донецького національного технічного університету. Серія: “Обчислювальна техніка та автоматизація”. Випуск 16(147) Донецьк: ДонНТУ 2009. с.121-127.

[10] Tahseen A. Al-Ramadin Reduction Technique for Instance-based Learning Using Distributed Genetic Algorithms // International Journal of Grid and Distributed Computing Vol. 4, No. 3, September, 2011.

[11] Родзин С.И. Организация параллельных эволюционных вычислений // Известия ЮФУ, 07.2011.

[12] URL: http://chaoswarehouse.org/index.php/home/cwh-tech-platform/net/32-hpc-htc-cloud(дата обращения: 26.02.2016)

[13] Цветков В.В. Анализ и моделирование работы конвейера параллельной потоковой вычислительной системы // Материалы Международной научно-технической конференции «Многопроцессорные вычислительные и управляющие системы» (МВУС-2009), Таганрог, ТТИ ЮФУ. Т. 1, с. 116-119.

Adoption of Genetic Algorithms for running in elastic compute environment concerning CAD applications

A.A. Shlepnev

Institute for Design Problems in Microelectronics of RAS (IPPM RAS),
shleps@ippm.ru

***Keywords* — elastic computing, cloud computing, grid computing, genetic algorithm, optimization, parallel computing, infrastructure.**

ABSTRACT

Genetic algorithms are widely used for optimization purposes in EDA CAD. This application usually requires a lot of computing power so resources of cloud providers could be engaged. GA are well known subject for parallel computing, but traditional approaches for parallelizing does not allow to take all advantages of Clouds, especially elasticity. In this article two ways of adoption of Genetic Algorithms for running in Cloud infrastructure are proposed. First way requires any kind of commercially available grid engine like LSF, OpenLAVA, SGE or Univa to run parallel parts of algorithm such as fitness function computing and uses file operations on common storage to provide synchronization among algorithm stages. Second way is building GA application using elastic architecture, similar to architecture of GRID engines for running parallel parts of algorithm, and using the proposed «waiter» synchronization primitive similar to tokens used in parallel dataflow computing systems for synchronization. This way allows to increase/reduce the number of computing workers on-the-fly depending on current workload to provide full elasticity of application and decrease latency of interprocess communications by hundreds times (comparing to the first way).

REFERENCES

[1] Yeroshenko I.N. Planirovanie kristalla SBIS s uchetom energopotrebleniya // Sb. trudov MES-2012. MES-2012. Russia, Moscow, October 2012. pp. 257-262 (in Russian)

[2] URL: http://www.idexpert.ru/news/4972/ (accessed: 26.02.2016, in Russian).

[3] URL: http://www.pcweek.ru/its/article/detail.php?ID=175179 (accessed: 26.02.2016, in Russian).

[4] S. Dustdar et al. Principles of Elastic Processes // IEEE Internet Computing, vol. 15, no. 5, 2011.

[5] Mell P., Grance T. The NIST Definition of Cloud Computing // NIST, Special Publication 800-145.

[6] URL: http://buildacloud.org/blog/181-high-performance-computing-and-cloudstack.html (accessed: 26.02.2016).

[7] URL: http://cyclecomputing.com/blog/cyclecloud-50000-core-utility-supercomputing/ (accessed: 26.02.2016).

[8] URL: http://www.teraproc.com/front-page-posts/cluster-as-a-service-for-hpc/ (accessed: 26.02.2016)

[9] Ivanov D.E., Chebanov P.A. Vzaimodeejstvie komponent v raspredelennyh geneticheskih algoritmah generatsii testov (in Russian).

[10] Tahseen A. Al-Ramadin Reduction Technique for Instance-based Learning Using Distributed Genetic Algorithms // International Journal of Grid and Distributed Computing Vol. 4, No. 3, September, 2011

[11] Rodzin S.I.. Organizatsiya parallelnyh evolutsionnyh vychislenij // Izvestia UFU, 07.2011 (in Russian).

[12] URL: http://chaoswarehouse.org/index.php/home/cwh-tech-platform/net/32-hpc-htc-cloud (accessed: 26.02.2016)

[13] Tsvetkov V.V. Analiz i modelirovanie raboty konvejera parallelnoj potokovoj vychislitelnoj sistemy // Materialy Mezhdunarodnoj nauchno-tekhnicheskoj konferencii “mngoprocessornye vychislitelnye I upravlyaushhie sistemy” (MVUS-2009), Taganrog, TTI UFU. vol. 1, pp. 116-119 (in Russian).