It may be also used for data errors detection and correction. For that an algorithm may be investigated. The method has been applied for temperature measurements and it can be extended for other weather parameters with small modification.

**REFERENCES**


1. Glahn, H. The Use of Model Output Statistics (MOS) in Objective Weather Forecasting/ H. Glahn and D. Lowry// Journal of Applied Meteorology, vol. 11, Issue 8, Dec. 1972, pp. 1203-1211.
2. Gringorten, I. Methods of Objective Weather Forecasting/ I. Gringorten// Advances in Geophysics Volume 2, 1955, pp 57–92.
3. Leith, C. Objective Methods for Weather Prediction/ C. Leith// Annual Review of Fluid Mechanics, Jan. 1978, Vol. 10, Pages 107-128.
4. Saloman, D. Data Compression the Complete Reference/ D. Saloman//Springer, 3rd Edition (2004).
5. Zhen, Ch. Design and Realization of Data Compression in Real-Time Database/ Ch. Zhen and B. Ren// IEEE International Conference on Computational Intelligence and Software Engineering, China, 11 – 13 Dec. 2009 pp. 340-243.
6. Yang, H. On the performance of data compression algorithms based upon string matching/ H. Yang// IEEE Transactions on Information Theory, Jan 1998, Vol. 44, Issue 1, pp.47- 65.
7. Lin, M. A New Architecture of a Two-Stage Lossless Data Compression and Decompression Algorithm/ M. Lin and Y. Yi Chang//IEEE transactions on VLSI, Vol. 17, Issue 9, Sept. 2009, pp. 1297- 1303.
8. Ray, G. Data Compression in Databases/ G. Ray//Master's Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, June 1995.
9. Tamrakar, A. A Compression Algorithm for Optimization of Storage Consumption of Non Oracle Database/A. Tamrakar and V. Nanda//A Compression Algorithm for Optimization of Storage Consumption of Non Oracle Database, July 2012, Vol. 1, Issue 5, pp. 39-43.
10. Всё о сжатии данных, изображений и видео [Electronic resource]/Access mode: http:// www. compression.ru.- Title from screen (last visit on 17-03-2013).


*Аспирант **Хуссейн М.Х.,** helphs@yahoo.com, Проф**. Якунин А.Г.,** тел. (8(3852) 29-07-86. e-mail: lis@agtu.secna.ru., Россия, 656038, г. Барнаул, пр-т Ленина, 46, Алтайский государственный технический университет им.И.И.Ползунова, Факультет информационных технологий, кафедра вычислительных систем и информационной безопасности,*

**УДК: 519.687.1**

# ИНСТРУМЕНТАЛЬНАЯ ПОДДЕРЖКА ЭФФЕКТИВНОГО ИСПОЛЬЗОВАНИЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ГЕТЕРОГЕННЫХ СИСТЕМ

С.А. Гризан, А.И. Легалов

Рассматриваются подходы к оптимизации программ, предназначенных для выполнения вычислений на графических ускорителях. Представлены алгоритмы балансировки нагрузки, зависящие от критических параметров решаемой задачи. Приведены примеры использования разработанной библиотеки, обеспечивающей подстройку решаемой задачи под используемую конфигурацию вычислительных средств.

**Ключевые слова:** GPU, CUDA, автоподстройка, оптимизация.

## Введение

В настоящее время наблюдается широкое использование суперкомпьютерных систем на основе неоднородной (гетерогенной) архитектуры. Основными вычислительными элементами в них являются специализированные графические ускорители (GPU), позволившие обойти кластерные системы как по пиковой, так и по реально достигнутой производительности. Вместе с тем, графические ускорители, являясь одной из перспективных вычислительных платформ, требуют для своего эффективного использования дополнительных затрат на анализ методов распараллеливания решаемой прикладной задачи и их программирование. Для сокращения этих затрат и создания эффективных параллельных программ необходимы методы автоматизации программирования, направленные на оптимизацию кода для ключевых вычислительных ядер, под которыми в прикладных задачах понимаются фрагменты кода, реализующие параллельные алгоритмы. Вычислительные ядра при этом могут быть автоматически подстроены под конкретное приложение как в контексте определенных входных и

выходных данных, так и в контексте используемого аппаратного обеспечения.

**Факторы, влияющие на эффективность гетерогенных архитектур**

При создании прикладных программ для графических ускорителей разработчики решают две основные задачи: обеспечение работоспособности программы и достижение роста ее производительности. В большинстве случаев популярные инструментарии, такие как PGI Accelerator[1], HMPP[2] пытаются решить обе указанные задачи. Однако в первую очередь они предназначены для упрощения программирования, а вопросы оптимизации зачастую приходится решать самим разработчикам. В последнее время наблюдается процесс непрерывного упрощения архитектур ускорителей, в результате чего проблемы первого типа становятся менее актуальными. Например, программная модель NVidia CUDA, поддерживает всё больше и больше возможностей языка программирования C++. Поэтому оптимизация программ и необходимость достижения требуемого ускорения выступает на первый план. Ряд проблем оптимизации для графических ускорителей полностью или частично могут быть автоматизированы благодаря применению специальных средств динамической адаптации программ.

Одной из ключевых задач является **обеспечение равномерной загрузки вычислителя**. Графический ускоритель NVidia состоит из независимых вычислительных блоков, называемых мультипроцессорами, каждый из которых содержит несколько десятков вычислительных ядер, набор регистров и встроенную кэш-память. Обеспечение равномерной загрузки всех мультипроцессоров зачастую является одним из критериев, поскольку позволяет повысить производительность за счет уменьшения времени простоя отдельных узлов. Для этого программист должен разбить все нити вычислений на блоки, каждый из которых будет обрабатываться на одном мультипроцессоре. Практически во всех современных программах подобные разбиения делаются статически, исходя из априорной информации. Отсутствие дополнительной информации, которая обычно может быть получена только во время вычислений, ведет к потере производительности вычислений. Данная проблема и методы её статического решения описаны сотрудниками компании Nvidia. Благодаря удачному выбору топологии данных они смогли ускорить тестовую программу более чем вдвое [3, 4].

Важную роль в повышении эффективности играет **балансировка вычислительной нагрузки между вычислителями**. Графические ускорители обычно используются совместно с центральным процессором. Несмотря на то, что пиковые производительности этих двух классов вычислителей несоизмеримы, проведенные исследования показывают, что при соответствующей оптимизации несколько центральных процессоров, расположенных на одной плате и имеющих общую память, могут обеспечит производительность, соизмеримую с показателями современных графических ускорителей [5] даже на задаче, идеальной для графического ускорителя. При вычислениях с двойной точностью или с интенсивными операциями обращения к памяти, аналогичные результаты можно получить даже без оптимизации программы, выполняемой на центральном процессоре.

При распределении вычислительной нагрузки между графическим ускорителем и центральным процессором практически невозможно обеспечить эффективное статическое распределение. Она будет существенно меняться в зависимости от типов процессоров и ускорителей, пропускной способности каналов, операционной системы и обрабатываемых данных. Поэтому в общем случае более быстрым вычислителем попеременно могут быть то графический ускоритель, то многоядерный центральный процессор. Более того, даже если сделать замеры времени на начальных итерациях, а затем на их основе задать распределение нагрузки для последующих итераций, то эти оценки через некоторое время работы программы потеряют свою актуальность.

Появление стандарта OpenCL и усложнение архитектуры ускорителей также повлияли на **выбор вычислительного ядра алгоритма, оптимального для конкретной системы**. При проведении специфических оптимизаций, когда существует несколько альтернативных методов повышения производительности, ответ на вопрос, какой же из путей окажется более перспективным, не всегда однозначен. Например, графические ускорители NVidia и AMD имеют свои специфические особенности. Для достижения максимальной производительности необходимо как минимум иметь две версии алгоритма, каждая из которых оптимизирована под ускорители соответствующей компании. Например, реализация одного из алгоритмов обработки фотографий с использованием OpenCL и локальной памяти в качестве программиру-

емого кэша замедлило программу на ускорителе от AMD примерно на 25%, в то время как на схожем по производительности ускорителе от NVidia было получено двукратное ускорение [6]. Более того, если с использованием технологии OpenCL предполагается разрабатывать вычислительные ядра не только для графического, но и для центрального процессора, то, скорее всего, придется разрабатывать отдельную ветку с кардинально переписанной структурой алгоритма, чтобы обеспечить возможность векторизации. В противном случае придется довольствоваться многократным снижением относительно реально достижимой производительности [7].

Зачастую бывает сложно определить требуемую степень параллелизма. Во многих алгоритмах число независимых вычислительных нитей заметно превышает количество доступных вычислительных ядер. С одной стороны, это полезное свойство алгоритма, поскольку оно позволяет гарантированно и равномерно загрузить все вычислители. С другой, выбор слишком легких нитей может обернуться повышенными накладными расходами при обработке, в результате чего суммарная производительность только уменьшится. В технологии NVidia механизмы автоматического укрупнения нитей отсутствуют, поэтому определение их вычислительной емкости лежит исключительно на программисте.

**Библиотека для систем с гетерогенной архитектурой**

Для инструментальной поддержки балансировки вычислительной нагрузки в гетерогенных высокопроизводительных вычислительных системах разработана библиотека ttgLib. Она предназначена для формирования специальной программной прослойки, обеспечивающей слежение за работой основной программы и ее динамическую подстройку под целевую систему и обрабатываемые данные. Подобная динамическая адаптация позволяет не только повысить производительность, но и автоматизировать процесс оптимизации. Ключевой особенностью библиотеки является система динамических параметров, встраиваемых в основную программу и позволяющих менеджеру оптимизации автоматически изменять их значения с целью повышения производительности. В случае, если информации оказывается недостаточно, разработчик может явно выбрать алгоритм и модель оптимизации или указать дополнительную априорную информацию. Для программиста это означает, что замена менее 5% исходного кода позволит на 30-50% повысить производительность программы, а дополнительная модификация 20-30% кода позволит многократно ускорить исходную программу.

Другими преимуществами динамически изменяемых параметров являются автоподстройка под произвольную систему с заранее неизвестной конфигурацией и возможность влиять на работу оптимизируемой программы, выполняющейся даже на удаленном компьютере. Это может оказаться полезным при проведении расчетов на специализированных высокопроизводительных серверах, построенных на базе графических ускорителей, так как позволяет не только обеспечить максимальную производительность вне зависимости от количества пользователей на целевой системе, но и следить за ходом вычислений через веб-браузер или специальные утилиты, влияя, при необходимости, на работу оптимизирующей системы.

Большая часть предлагаемой библиотеки реализована в виде **сервисов**, предоставляющих заданную функциональность посредством программных интерфейсов. В качестве примера можно привести: сервис, отвечающий за создание записей о работе программы; сервис таймера, выполняющий замеры времени. Для инициализации сервисов необходимо создать экземпляр выделенного класса загрузчика, в конструкторе которого указать одну из поддерживаемых схем работы. В тех случаях, когда требуется дополнительная настройка отдельного сервиса, имеется возможность переопределить базовую схему работы. Настройки отдельного сервиса, добавленные позднее, перекрывают предыдущие, тем самым позволяя изменять любые опции базовых схем работы.

Основным средством встраивания возможностей библиотеки ttgLib в оптимизируемые приложения является **механизм динамических параметров**. Он реализован в виде шаблонных классов вида Parameter<T>, которые с точки зрения программиста эквивалентны обычным переменным. Однако их использование позволяет подсистеме оптимизации собирать статистику и незаметно влиять на программу. Благодаря внешним утилитам пользователь получает возможность не только наблюдать за значениями подобных динамических переменных, но и изменять их

Для упрощения реализации различных паттернов программирования [8], предназначенных для эффективной реализации типо-

вых параллельных конструкций, библиотека ttgLib предоставляет набор **гибридных примитивов**, позволяющих реализовать алгоритм для различных вычислительных систем. С помощью данных примитивов, являющихся обёртками поверх динамических параметров, можно сгруппировать несколько альтернативных вычислительных ядер, написанных с использованием технологий NVidia CUDA, OpenCL и/или расширений SSE/AVX. В результате при работе для каждого вычислителя будет выбрано оптимальное ядро, а вычислительная нагрузка равномерно распределится между всеми устройствами. Разработаны два примитива: HybridFor и HybridTask.

Примитив HybridFor реализует паттерн параллельного обхода по итераторам коллекции, что позволяет заменить произвольный одномерный цикл несколькими независимыми циклами, каждый из которых будет обрабатывать свою часть данных на целевом устройстве. Для этого создается экземпляр класса примитива, регистрирующий доступные вычислительные ядра, который в дальнейшем будет использоваться для запуска вычислений. Из специфичных особенностей данного примитива следует отметить возможность задать выравнивание передаваемых в ядро интервалов цикла, что, например, необходимо для корректной работы расширений SSE.

Примитив HybridTask предназначен для выполнения набора неделимых неоднородных задач, каждая из которых может быть выполнена на одном из доступных устройств, возможно, с использованием альтернативных вычислительных ядер. В частности, это позволяет равномерно загрузить все вычислители при выполнении пакетных заданий. При этом для каждого вычислителя будет подобрано не только оптимальное ядро, но и преимущественно ему будут передаваться те задачи, на которых он показывает максимальную производительность.

Ключевым механизмом ttgLib является **подсистема оптимизации**, позволяющая для произвольной группы динамических параметров и/или гибридных примитивов подобрать значения, обеспечивающие максимальную производительность. Для этого предполагается, что вычисления производятся итерационно, и на каждом шаге вычислительная сложность линейно зависит от размера обрабатываемых данных.

При использовании ttgLib все критические по производительности величины должны быть заменены динамическими параметрами вида Parameter<int>, являющимися шаблонными классами языка C++. В результате этого ttgLib получает возможность прозрачно для программы подменять значения параметров при поиске более оптимальных. Время выполнения GPGPU-ядра задается минимизируемой функцией, аргументами которой служат значения констант, заменённых динамическими параметрами. Таким образом, выполнение GPGPU-ядра с использованием ttgLib является лишь одним замером данной функции времени. При этом предполагается, что специфика вычислений не сильно изменяется от запуска к запуску одного и того же ядра, для решения которых применяются итерационные методы.

Благодаря этому имеется возможность применять различные классические методы оптимизации функционалов с учетом ряда ограничений:

- число доступных замеров достаточно мало, так как необходимо обеспечить приемлемое время настройки оптимального значения параметров;
- функционал времени может изменяться от замера к замеру, в результате чего старые значения могут устаревать;
- функционал определён не на всей области и может иметь разрывы.

В большинстве случаев об оптимизируемом приложении известна дополнительная априорная информация: ожидаемое количество итераций, примерное время одной итерации, требуется ли время для стабилизации производительности и другая. Эту информацию можно использовать для уточнения стратегии, что повышает качество оптимизации и скорость обучения. Для использования этих данных применяются стратегии оптимизации, в которых задаются подходящие настройки и алгоритмы. По умолчанию в библиотеке используется стратегия, рассчитанная не менее чем на 200 итераций, среднее время которых превышает 5 миллисекунд, что позволяет с одной стороны подобрать достаточно оптимальную конфигурацию параметров, и с другой стороны проводить дальнейшие вычисления без дополнительных накладных расходов.

В настоящий момент реализовано два базовых алгоритма оптимизации вычислений: это алгоритм интеллектуального перебора и генетический алгоритм.

Алгоритм интеллектуального перебора является модифицированной версией покоординатного спуска. В агрессивном режиме

он перебирает все комбинации параметров в порядке убывания их производительности. В основном режиме он пытается удержаться в найденном экстремуме. Предназначен для случаев, когда имеется небольшое количество оптимизируемых параметров (от 1 до 4), или же число агрессивных итераций не превышает 200-500. В этом случае им обеспечивается лучшая скорость обучения и требуется минимальное количество замеров. В стратегии оптимизации можно задать параметр, отвечающий за качество оптимизации. Чем выше качество, тем ниже скорость обучения, и наоборот.

Генетический алгоритм в агрессивном режиме проводит активные мутации и скрещивания лучших наборов параметров. В обычном режиме выбирает лучше наборы из оставшейся популяции. Предназначен для длительных вычислений с количеством агрессивных итераций более 1000, не имеющих резких изменений производительности. Он также полезен при оптимизации более чем по 10 параметрам.

Помимо этого в библиотеке имеются алгоритмы балансировки вычислительной нагрузки между различными устройствами. В частности с их использованием оптимизируется примитив HybridFor. В настоящий момент реализованы алгоритм следования тренду и LUT-алгоритм.

**Следование тренду** является основным алгоритмом, распределяющим нагрузку и выбор вычислительных ядер в зависимости от текущей производительности каждого устройства. Он имеет два порога, первый из которых определяет минимальную порцию данных, которую имеет смысл отправлять на отдельный вычислитель, а второй - максимальную порцию, при превышении которой задача будет обрабатываться только одним устройством. Используется для случаев, когда имеется несколько вычислителей со схожими архитектурными особенностями и/или нет времени на проведение дополнительного обучения.

При работе **алгоритма LUT** (Lookup Table, таблица поиска) производится построение функции производительности каждого отдельного устройства в зависимости от размера обрабатываемых данных, что позволяет учитывать и использовать кэш-эффекты и прочие резкие изменения производительности. Предназначен, когда используются вычислители с разной архитектурой и/или имеется достаточное количество альтернативных вычислительных ядер (более 2 веток на одно устройство).

**Тестирование библиотеки**

Апробация разработанной библиотеки проводилась на трех модельных задачах аэродинамики, в каждой из которых использовались рассмотренные выше автоматически подбираемые параметры: это трёхмерное уравнение Лапласа в постановке задачи Дирихле на структурированной сетке методом Якоби, уравнение теплопроводности, заданное на двумерной области с граничными условиями Дирихле с использованием явной разностной схемы, и уравнение Эйлера на двумерной области со смешанными граничными условиями с использованием метода HLL [9].

В первой задаче использовался только один GPU NVidia Tesla C2050, в то время как во второй и третьей расчёты проводились сразу на кластере, каждый узел которого содержал по три GPU NVidia Tesla C2070, что породило дополнительные параметры для балансировки нагрузки.

Проведенные эксперименты показали, что при решении уравнения Лапласа достигнуто повышение производительности в среднем на 20% (при этом наилучший результат составил 37%), для уравнения тепломассопереноса – 80-90%, и для уравнения Эйлера – 60%.

**Заключение**

Рассмотренный подход к оптимизации GPGPU-программ путём встраивания в них разработанной технологии автоподстройки оказался эффективным при решении трех модельных задач, используемых в аэродинамике. При подстройке программы к связке «обрабатываемые данные + аппаратная платформа» за счёт подбора оптимальных значений всего для 4-5 констант удалось повысить производительность в среднем на 20-60% относительно исходных версий без внесения в них каких-либо существенных изменений. Таким образом, реализация перечисленных принципов и решение рассмотренных проблем при помощи предложенной библиотеки ttgLib позволяет повысить производительность приложения при меньших затратах на разработку программного обеспечения, что достигается за счет автоматизации процесса создания ключевых вычислительных ядер.

**СПИСОК ЛИТЕРАТУРЫ**

1. PGI Accelerator Programming Model for Fortran & C [электронный ресурс] / The Portland Group, 2010 – Режим доступа: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf
2. Romain Dolbeau. HMPP: A Hybrid Multi-core Parallel Programming Environment / Romain Dolbeau, Stephane Bihan, Francois Bodin – CAPS entreprise, 2007.
3. Nyland, L. Fast N-Body simulation with CUDA / L.Nyland, M. Harris // GPU Gems 3 – 2007 – с. 677.
4. CUDA C Best Practices Guide [электронный ресурс] / NVidia corporation, version 3.2 – с. 50 – режим доступа: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
5. Кривов, М.А. Портируем на GPU и оптимизируем под CPU / М.А. Кривов, А.М. Казеннов // Журнал «Суперкомпьютеры», Весна 2011 – с. 43-45.
6. Кривов, М.А. Опыт портирования среды для HDR-обработки изображений на GPU и APU. / М.А. Кривов, М.Н. Притула, С.Г. Елизаров // Параллельные вычислительные технологии (ПаВТ'2012): труды международной научной конференции (Новосибирск, 26 – 30 марта 2012 г.), Челябинск: Издательский центр ЮУрГУ, 2012 – 774 с.
7. Кривов, М.А., Сине-зелено-красная OpenCL/ М.А. Кривов// Журнал «Суперкомпьютеры», Осень 2011 - с. 47-50.
8. Timothy, G. Mattson. Patterns for Parallel Programming / G. Mattson Timothy; Beverly A. Sanders; Berna L. Massingill – Addison-Wesley Professional, NY: 2009.
9. Janhunen, P. A positive conservative method for magnetohydrodynamics based on HLL and Roe methods / P. Janhunen – J. Comp. Phys., 1999.

*Аспирант **С.А. Гризан**, д.т.н., проф., зав. каф. А.И. Легалов - каф. вычислительной техники Сибирского федерального университета*

**УДК: 004.81:159.9**
**УДК: 004.93'1; 004.932**
**УДК: 615.471; 681.32(075)**

# КОГНИТИВНОЕ МАШИННОЕ ЗРЕНИЕ НА ОСНОВЕ ПАРАМЕТРИЧЕСКОГО АНАЛИЗА СТРУКТУРНЫХ ПРИМИТИВОВ ИЗОБРАЖЕНИЙ

К.Б. Саниев

Проблема автоматического распознавания изображений ставится в общем виде как информационная задача извлечения из видеосигнала данных об априорно неопределённых сценах и объектах. Решение строится на операциях обнаружения визуальных структурных примитивов, определении достаточного набора их признаков, безэталонной классификации по характеристическим признаковым гистограммам и различению по форме. Приводятся примеры апробации элементов когнитивной технологии машинного зрения при распознавании изображений объектов различных классов и динамических сцен.

**Ключевые слова:** когнитивное машинное зрение, визуальные примитивы, безэталонная классификация.

## Введение

Практические задачи автоматизации и информатизации производств, процессов и научных исследований требуют повышения эффективности всех видов систем технического зрения (СТЗ). Одной из основных целей этих разработок по-прежнему остаётся воспроизведение в распознающих автоматах информационных функций биологического зрительного восприятия. Качественное сравнение его информативности и методов автоматического анализа сигналов изображений показывает следующее.

- Современные технологии автоматического распознавания изображений, исходящие из методологии «распознавания образов», реализуют [1-3] частную информационную функцию классификации в виде проверки гипотез о наличии заранее заданных объектов. Априорная неопределённость множества (классов) объектов, которые могут быть различимы и опознаны, снимается частично посредством «ручного» задания $M$ эталонных описаний, построенных на $N$ эвристически сформированных признаках. Возможность расширения и коррекции множества эталонов в автоматическом режиме отсутствует. Информационная эффективность функции распознавания в этом случае принципиально ограничена количеством $M$ заданных эталонов.