Всероссийская научно-практическая конференция

Применение гибридных высокопроизводительных вычислительных систем для решения научных и инженерных задач

Нижегородский Государственный Университет им. Н.И. Лобачевского –
Национальный исследовательский университет

Нижний Новгород, 1 марта 2011

Оглавление

Доклады

Высокоуровневые средства программирования для GPU

Ю. Сердюк
Институт программных систем РАН, г.Переславль-Залесский
Yury@serdyuk.botik.ru

Представлен обзор систем программирования для GPU на основе .NET и языка C#: язык программирования MC#, проект Accelerator V2 компании Microsoft и система GPU.NET компании TidePowerd. Рассмотрен синтаксис средств языка MC# для программирования GPU и их реализация. Приведены результаты тестирования приложений, разработанных на языке MC#, на машинах с GPU Nvidia Tesla C1060и Tesla M2050, включая машины с несколькими GPU и кластерные системы.

Введение

В настоящий момент, для достижения более высокой производительности вычислительных устройств, всё более часто применяются их гибридные варианты, использующие различного типа ускорители: графические процессоры (GPU), процессоры Cell, программируемые логические матрицы (FPGA).

Использование такого рода гибридных параллельных архитектур дополнительно усложнило процесс разработки приложений для них, поскольку потребовало применения целой группы технологий/библиотек (таких как MPI, Pthreads, CUDA и т.п.) для создания одного приложения. Частично проблему унификации разнообразных программных средств для программирования такого рода архитектур решает библиотека OpenCL, но она достаточно сложна в изучении и использовании и имеет свои ограничения.

В данной работе представлено расширение языка MC# конструкциями и средствами для программирования графических процессоров [1].

Средства языка MC# для программирования графических процессоров

Язык программирования MC# является расширением языка C# и основан на платформе .NET. Он предназначен для разработки параллельных, распределенных приложений, предназначенных для исполнения на всех типах параллельных архитектур, включая гибридные системы и кластеры.

Базовая идея адаптации системы программирования MC# к вычислительным системам на базе GPU состоит во введении дополнительного модификатора *gpu*, который должен быть использован в определении метода объектно-ориентированной программы, предназначенного для исполнения на графическом процессоре.

Общая идеология программирования GPU на языке MC# совпадает с базовыми принципами технологии программирования CUDA. В частности, в системе программирования MC#, параметры конфигурации графического процессора (номер устройства, размеры решетки и размеры блока) устанавливаются путем задания значений для объекта GpuConfig, относительно которого *gpu*-методы вызываются как extension-методы.

Составной частью системы программирования MC#, поддерживающей графические процессоры, является библиотека GPU.NET, реализованная на языке C# и включающая в себя JIT-компилятор для GPU и набор функций, соответствующих функциям базовой библиотеки CUDA.

При программировании на языке MC# GPU, программист освобожден от необходимости явно программировать передачу данных из основной памяти в память графического устройства и обратно. Эта задача решается компилятором языка MC#, который генерирует вызовы соответствующих функций библиотеки GPU.NET, реализующих

копирование данных. Также, в *gpu*-методах возможно использование “разделяемой памяти” (shared memory) графического процессора. В языке MC#, массивы, размещаемые в разделяемой памяти, объявляются как параметризованные (generic) массивы.

Тестовые результаты

Тестирование программ на языке MC# для GPU проводилось в Межведомственном суперкомпьютерном центре (МСЦ) РАН, г.Москва, на 2-ух узловом кластере под управлением ОС Linux, где каждый узел был оснащен двумя GPU типа Nvidia Tesla C1060, и также на сервере с четырьмя устройствами Nvidia Tesla M2050.

В качестве тестовых приложений использовались программы перемножения матриц, решение задачи Дирихле для уравнения Пуассона, программа N-Queens, вычисляющая количество всех возможных различных расстановок ферзей на шахматной доске размером N x N, где ферзи попарно не атакуют друг друга.

Построены графики зависимости времени исполнения указанных программ как в зависимости от размера исходных данных, так и от количества использованных GPU (при фиксированном размере входных данных). Для задачи перемножения матриц, тестировались варианты как с элементами матриц типа float, так и с элементами типа double. Приведены сравнительные графики времени исполнения одних и тех же задач на GPU и на обычных (многоядерных) процессорах. Показано, что во многих случаях, программы, написанные полностью на MC# и исполняемые на GPU, превосходят по производительности программы, использующие функции библиотеки Intel MKL и исполняемые на узле с обычными многоядерными процессорами (а именно, на машине с двумя 6-ядерными процессорами Intel Xeon X5660).

Смежные работы и выводы.

Ключевая идея системы “Accelerator V2 for GPGPU and SIMD x86 multicore target” компании Microsoft состоит во введении специальных объектов − “параллельных массивов” и операций над ними, которые можно использовать в программах на языках C# и F#. Данные операции транслируются в код, исполняемый на графическом устройстве. При данном подходе, программист не имеет возможности написать собственную функцию, использующую произвольные операции и предназначенную для исполнения на GPU. А потому круг задач, решаемых на основе данного подхода, является достаточно узким.

Система GPU.NET компании TidePowerd по своей идеологии очень близка к подходу, принятому в языке MC#. Основное ее отличие от MC# состоит в том, что компоненты программы, относящиеся к выполнению на GPU, помечаются с помощью атрибутов языка C#. В частности, методы, предназначенные для исполнения на GPU, помечаются атрибутом *Kernel*, а массивы, размещаемые в разделяемой памяти, определяются с помощью атрибута *SharedMemory*. К недостаткам текущей реализации системы GPU.NET компании TidePowerd следует отнести невозможность работы с несколькими GPU в одной программе, отсутствие версии для ОС Linux, отсутствие библиотеки базовых математических функций типа sin, cos и др.

В данной работе продемонстрированы возможности использования высокоуровневых средств программирования для современных GPU. Приведены тестовые результаты для приложений, написанных на языке MC#.

Текущую версию системы MC# можно найти на сайте www.mcsharp.net.

Литература.

1. Ю. Сердюк. “Программирование графических процессоров на языке MC#”. - Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной суперкомпьютерной конференции (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, 2010.

Реализация алгоритма Viola-Jones для архитектуры NVIDIA CUDA

И.И. Зиновьев, Т.Е. Овчинникова, П.Ю. Шамин
Владимирский государственный университет имени А.Г. и Н.Г. Столетовых
izinoviev@vlsu.ru

Комплекс алгоритмов для обнаружения объектов, представленный в 2001 году Полом Виолой (Paul Viola) и Майклом Джонсом (Michael Jones) [1], логически состоит из двух компонентов: обучение классификатора и непосредственно обнаружение объектов на изображении. На практике основные требования к скорости работы предъявляются ко второму алгоритму. В данной работе рассматривается реализация этого алгоритма для архитектуры NVIDIA CUDA, которая позволяет в несколько раз ускорить его работу по сравнению с реализаций на центральном процессоре в библиотеке OpenCV [7].

Алгоритм Viola-Jones обнаружения объектов хорошо известен. Поэтому перечислим только наиболее значимые для рассматриваемой темы его особенности:

1. Большое количество обращений к памяти. Для вычисления отклика одного классификатора требуется от 11 до 15 обращений к памяти.
2. Вычисление интегрального изображения – особое представление исходного изображения, позволяющее быстрое вычисление характерных черт (особенностей), используемых при классификации изображения. Каждый пиксель интегрального изображения представляет собой сумму значений всех пикселей исходного изображения, расположенных выше и левее текущего пикселя
3. Применение каскада классификаторов, представляющего собой объединение в цепочку классификаторов по такому принципу, что выход одного является входом для другого. Поэтому, если участок изображения не проходит один классификатор, он сразу отбрасывается и не идет по всей цепочке.

Для эффективной реализации алгоритма необходимо также учесть особенности архитектуры NVIDIA CUDA, с которыми можно ознакомиться в [5].

Рассматриваемый алгоритм реализован как одно CUDA-ядро (рис. 1). Это позволило минимизировать пересылки данных через глобальную память графического ускорителя.

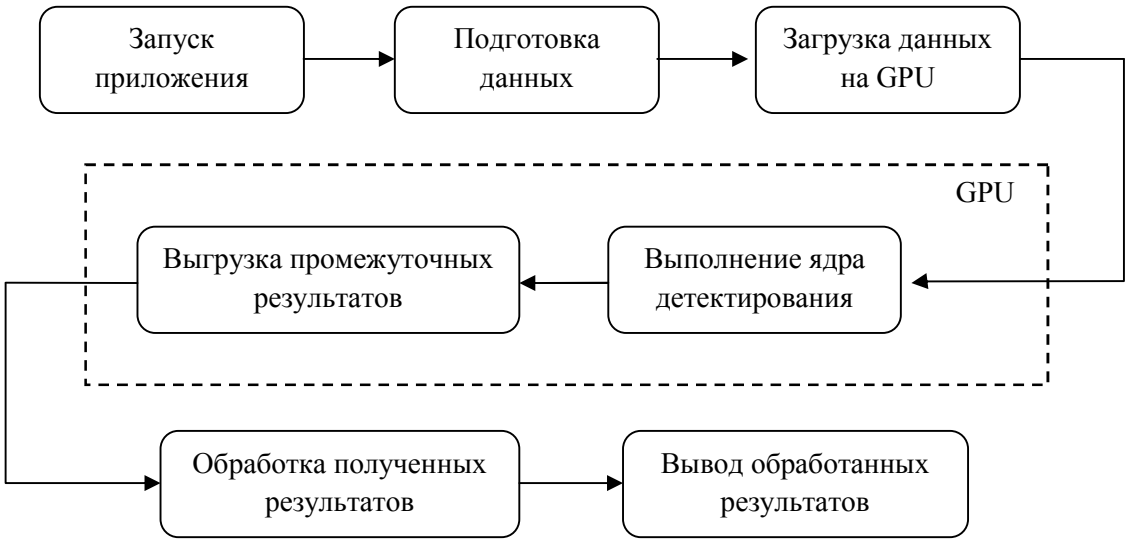


Рис. 1. Схема работы алгоритма Viola-Jones, реализованного с использованием графического ускорителя

Правильный выбор размера блока потоков и количества используемых регистров позволяет достичь значения коэффициента использования мультипроцессора порядка 2/3, что является достаточным для нивелирования влияния латентности при работе с памятью.

Исходные для алгоритма данные, а именно каскад классификаторов и анализируемое изображение, размещаются в текстурной и константной памяти графического ускорителя. Каскад классификаторов является неизменным, необходим всем потокам одновременно, что позволяет эффективно использовать особенности работы константной памяти. Размещение исходного изображения в текстурной памяти позволяет реализовать эффективный алгоритм его масштабирования путем выборки из текстуры с аппаратной билинейной фильтрацией.

Из-за большого числа обращений к памяти при работе рассматриваемого алгоритма необходимо использовать разделяемую память графического ускорителя как управляемый L1 кэш. Все потоки мультипроцессора анализируют частично перекрывающиеся участки изображения, поэтому в разделяемую память копируется часть исходного изображения, достаточная для обеспечения всех потоков данными для классификации.

Узким местом реализации алгоритма на графическом ускорителе является необходимость пересылки данных между хостом и ускорителем. Для поиска лиц разного размера алгоритм Viola-Jones предусматривает анализ изображения в различных масштабах. Полученные промежуточные результаты сразу передаются в оперативную память компьютера. Применение технологии «zero copy», реализованной во всех современных графических ускорителях, позволяет сократить накладные расходы при пересылке данных практически к нулю за счет их асинхронного выполнения.

Как упоминалось выше, масштабирование изображения реализовано, используя возможности текстурной памяти. Масштабированное изображение считывается непосредственно в разделяемую память, где и осуществляется его анализ. Это исключает дополнительные обращения в глобальную память, что положительно сказывается на времени работы алгоритма. Проведенное сравнение с алгоритмом масштабирования изображения, реализованным в библиотеке примитивов NPP, это подтверждает (рис.2).

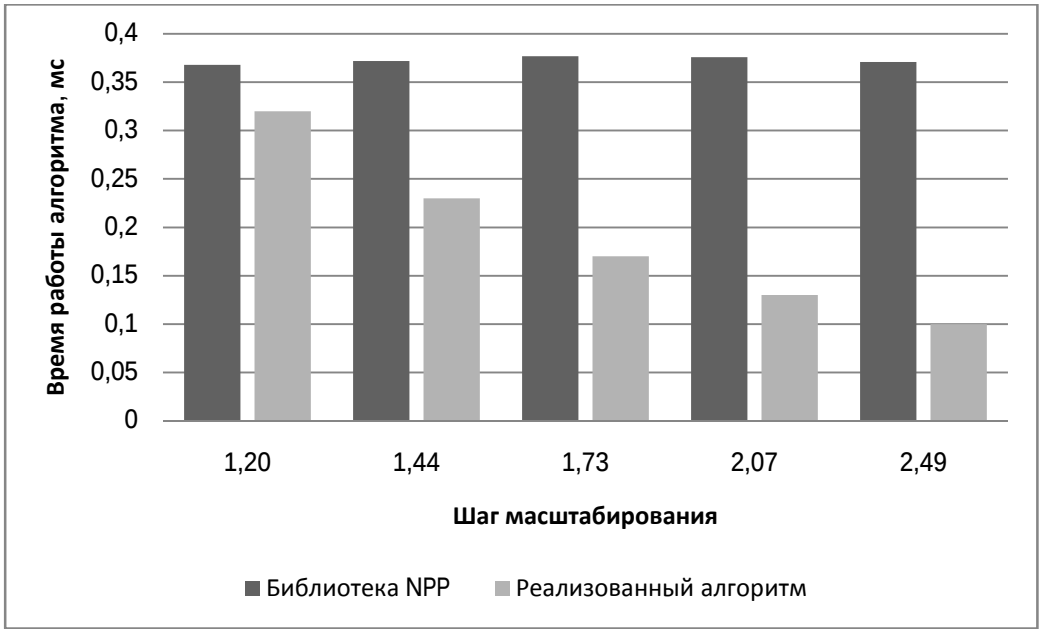


Рис. 2. Сравнение алгоритмов масштабирования изображения

Вычисление интегрального изображения является ключевым шагом в работе алгоритма Viola-Jones. Процесс вычисления интегрального изображения хорошо описан и изучен. Самым очевидным способом распараллеливания этого процесса является разбиение его на два этапа. На первом этапе проводится независимое суммирование по строкам изображения, на втором –

по столбцам. Однако для графического ускорителя реализация такого алгоритма усложняется из-за необходимости синхронизации результатов работы отдельных мультипроцессоров. Для облегчения процесса расчета интегрального изображения в данной работе предлагается использовать частичное интегральное изображение, которое вычисляется независимо для участков исходного изображения, уже размещенных в разделяемой памяти. Это становится возможным, так как в разделяемой памяти каждого мультипроцессора содержится достаточно данных для всех его потоков. Благодаря этому работа отдельных мультипроцессоров становится независимой, что исключает временные затраты на синхронизацию их работы. На рисунке 3 приводится сравнительная характеристика полученного алгоритма вычисления интегрального изображения с аналогичным алгоритмом, содержащимся в библиотеке NPP.

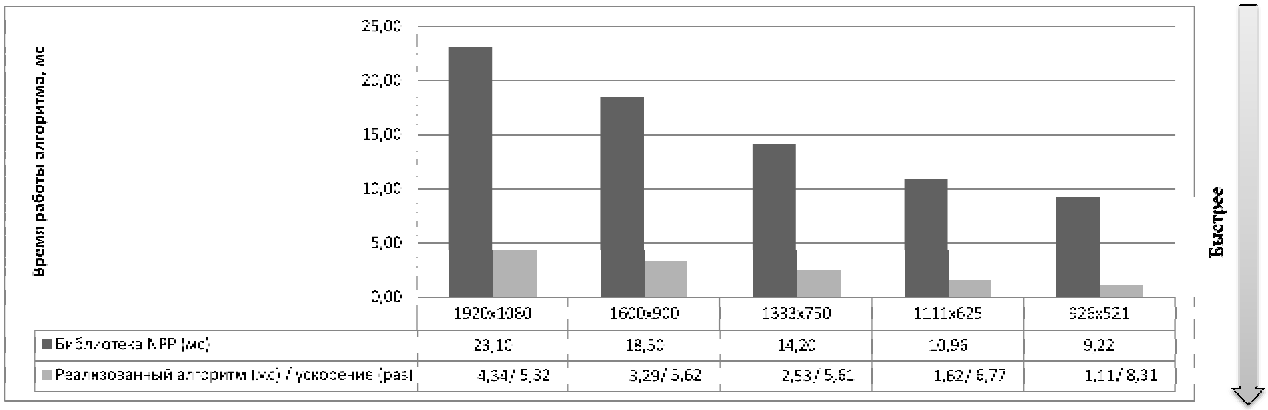


Рис. 3. Сравнение производительности алгоритмов вычисления интегрального изображения

Тестирование полученной реализации алгоритма Viola-Jones проводилось на изображениях трех размеров: 1920x1080, 1080x720 и 640x360. Для тестирования применялся каскад классификаторов, предложенный в работе [2]. Осуществлялся поиск лиц с минимальным размером 20x20 с шагом масштабирования исходного изображения 1,2. Рисунок 4 содержит результаты измерения производительности алгоритма в целом на различных графических ускорителях. Для сравнения приведена производительность алгоритма, реализованного в библиотеке OpenCV.

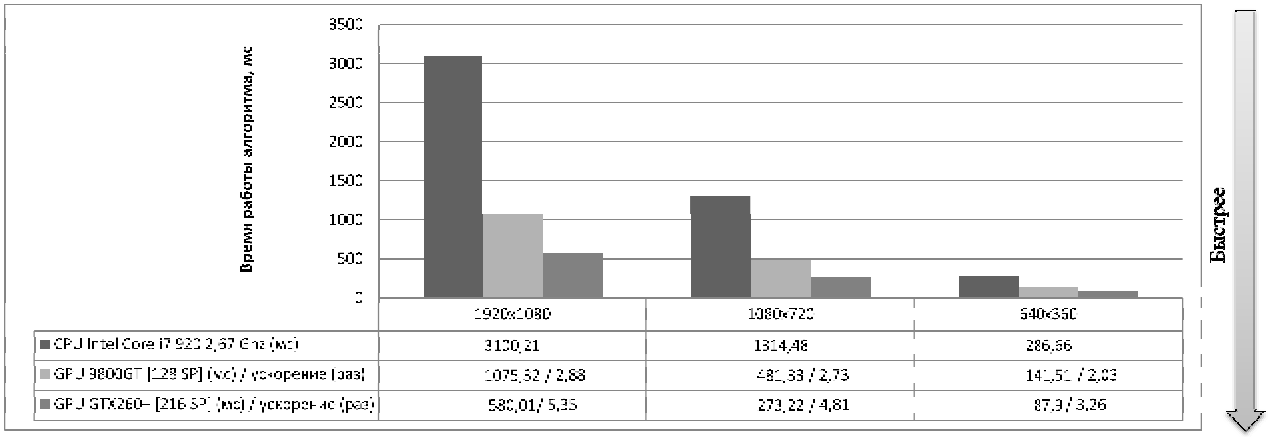


Рис. 4. Результаты измерения производительности алгоритма Viola-Jones

Можно отметить ускорение работы алгоритма на графическом ускорителе от 2 до 5 раз в сравнении с однопоточным вариантом, работающим на центральном процессоре. Такой результат в несколько раз превышает полученный автором работы [4]. Немаловажным является хорошая масштабируемость полученной реализации алгоритма. Это позволяет рассчитывать на еще большее ускорение на более современных графических ускорителях.

Выводы

В данной работе предложен оригинальный вариант эффективной реализации алгоритма Viola-Jones для архитектуры графических ускорителей производства NVIDIA. Следует отметить, что сам алгоритм не слишком приспособлен для реализации на графической карте. Большое число обращений к памяти, простои потоков при работе каскада классификаторов негативно сказываются на производительности. Но даже при этом полученная реализация алгоритма на современных графических ускорителях работает в несколько раз быстрее, чем аналоги на центральном процессоре.

Литература

Paul Viola, Michael J. Jones. Robust real-time face detection // International journal of computer vision 57(2), 137–154, 2004

Rainer Lienhart, Alexander Kuranov, Vadim Pisarevsky. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection.
URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.139.4825&rep=rep1&type =pdf (дата обращения: 21.07.2010)

Квасов Д.С., Зиновьев И.И., Кокорин И.Г., Коробко С.С. Обнаружение и идентификация лиц на растровом изображении с применением метода Виолы и Джонса // Труды XVII Всероссийской научно-методической конференции Телематика-2010, СПб, С. 253-254.

Jesse Patrick Harvey. GPU Acceleration of Object Classification Algorithms Using NVIDIA CUDA
URL: https://ritdml.rit.edu/bitstream/handle/1850/10894/35445_pdf_00B0B24A-DFD8-11DE-9A30-D21AD352ABB1.pdf?sequence=1. (дата обращения: 11.10.2010)

NVIDIA CUDA C Programming Guide, 2010. // CUDA Toolkit

CUDA C Best Practices Guide, 2010. // CUDA Toolkit

Документация открытой библиотеки OpenCV.
URL: http://opencv.willowgarage.com/wiki/ (дата обращения: 15.11.2010)

Программирование для нескольких GPU на примере задачи распространения света в многослойной среде

Багаутдинов Т.А.[1], Гергель В.П.[1], Горшков А.В.[1], Фикс И.И.[1,2], Кириллин М.Ю.[1,2]
[1]Нижегородский госуниверситет им. Н.И. Лобачевского
[2]Институт прикладной физики РАН

Введение

В настоящее время лазерные методы получили широкое распространение в бесконтактной неразрушающей диагностике внутренней структуры различных оптически неоднородных объектов, в частности, они находят применение в медицине, биофизике, науках о материалах, физике атмосферы, и других областях.

Для повышения эффективности современных методов лазерной диагностики, а также для разработки новых методов, необходимо подробное изучение особенностей процесса распространения света в различных средах, включая биоткани. Однако решение данной задачи затруднено тем, что на данный момент не существует точной теории для описания распространения света в структурно неоднородных средах, а экспериментальные исследования осложнены трудностями поддержания постоянства их структурно-динамических параметров. В связи с этим все большую роль приобретает компьютерное моделирование этого процесса (см., например, [1]). Данный подход позволяет более тщательно изучить особенности процесса распространения лазерного пучка в модельных средах, а также исследовать зависимость получаемых результатов от различных параметров измерительной системы и исследуемого объекта, что бывает весьма затруднительно в эксперименте.

Постановка задачи

В рамках данной работы ставится задача моделирования распространения света в многослойной среде. При этом считаем, что задача решается в трехмерном пространстве, а среда состоит из набора плоскопараллельных слоев. Каждый слой является бесконечно широким, описывается толщиной и рядом оптических характеристик.

Результатами моделирования являются значения следующих физических величин:

- пространственное распределение интенсивности рассеянного назад излучения на поверхности среды;
- пространственное распределение интенсивности рассеянного вперед излучения на задней границе среды;
- объемное распределение поглощенной интенсивности в среде.

Практическое решение поставленной задачи с использованием современных процессоров требует значительного времени вычислений. В связи с чем, актуальной является задача ускорения этого процесса.

Общее описание алгоритма решения задачи

Одним из подходов к решению задачи распространения света в многослойных средах является метод статистического моделирования Монте-Карло [1, 4]. Под методом Монте-Карло понимается совокупность приемов, позволяющих получать необходимые решения при помощи многократных случайных испытаний. Оценки искомой величины выводятся статистическим путем.

Применительно к задаче распространения излучения в многослойной среде метод Монте-Карло заключается в многократном повторении расчета траектории движения фотона в среде, исходя из задаваемых параметров среды.

Общий алгоритм решения задачи представлен на схеме [1] (рис. 1).

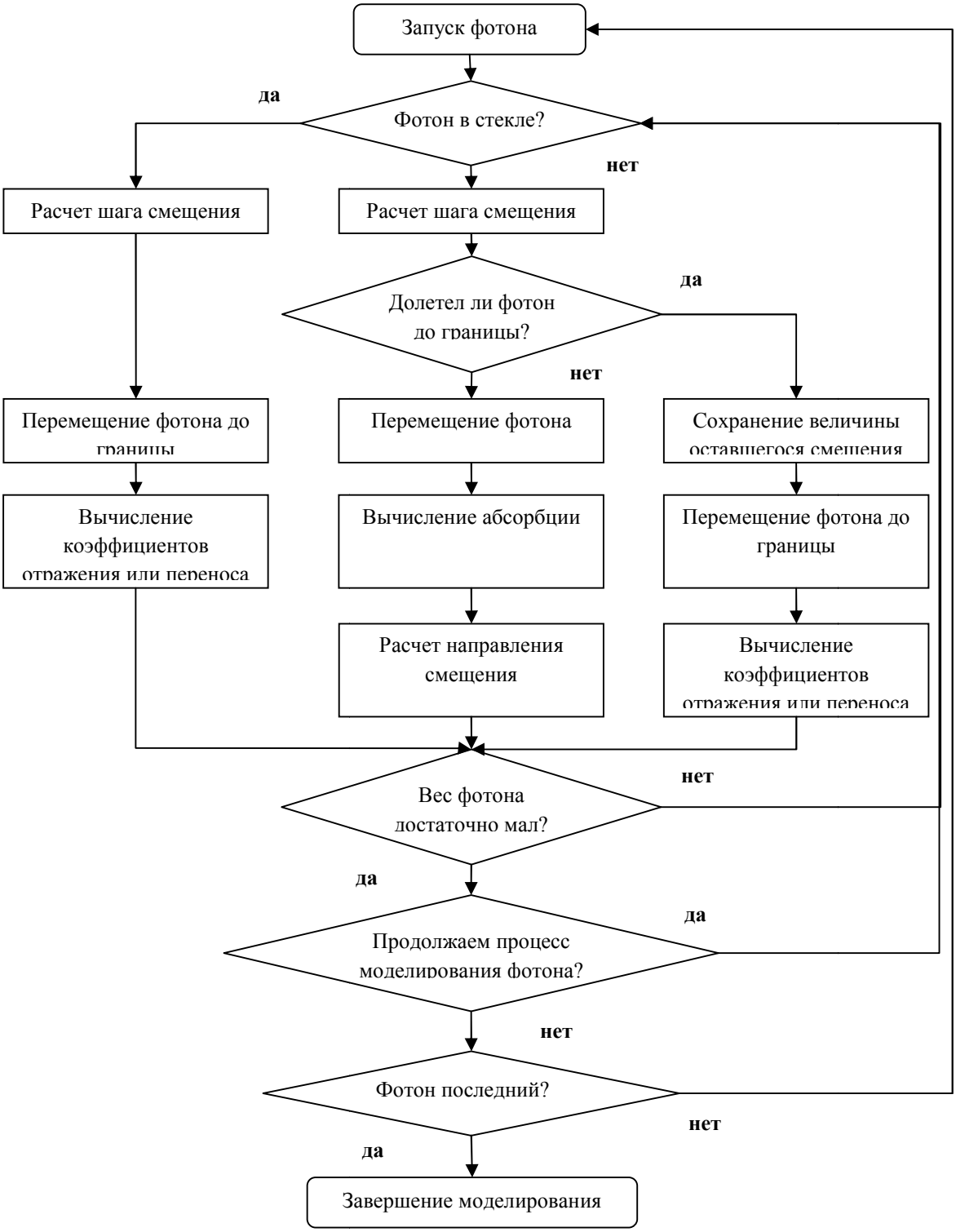


Рис. 1. Алгоритм моделирования распространения света в многослойной среде методом Монте-Карло

Для каждого фотона вышеуказанные действия выполняются независимо. Число фотонов в реальных экспериментах может быть достаточно большим (10^8 – 10^{10}), из чего можно предположить, что данная задача может быть эффективно реализована при использовании архитектуры графических адаптеров (GPU). В работе описывается реализация представленного выше алгоритма на графическом адаптере с использованием технологии CUDA [2].

Схема распараллеливания и оптимизация

В рассматриваемой задаче необходимые для моделирования действия производятся независимо над каждым фотоном, что позволяет эффективно применить схему распараллеливания, при которой каждый поток вычисляет траекторию движения своего фотона. Поскольку число фотонов велико, предложенная схема позволяет загрузить все доступные вычислительные ядра графического адаптера.

Следует отметить, что для повышения производительности каждому потоку имеет смысл проводить вычисления сразу над группой фотонов (в текущей реализации размер группы фотонов равен 1000).

Для обеспечения сходимости метода моделирования распространения света необходимо, во-первых, наличие длинной последовательности различных случайных чисел, во-вторых, организация работы каждого конкретного вычислительного потока со своей подпоследовательностью, причем все эти подпоследовательности не должны пересекаться.

Генератор случайных чисел, реализованный в стандартной библиотеке языка программирования Си (функция rand) порождает недостаточно длинную последовательность случайных чисел, и к тому же не может быть использован при программировании на GPU. Поэтому было принято решение адаптировать алгоритм генератора MCG59 для работы в несколько потоков на графическом адаптере. В текущей реализации все потоки работают с одной последовательностью случайных чисел, но при этом каждый поток использует свои ($id + i * N$) элементы этой последовательности (здесь id – номер потока, N – общее число потоков).

Важным условием эффективности программ на GPU является правильное использование имеющихся типов памяти. В частности для решения данной задачи необходимы следующие наборы данных: информация о среде (хранится в разделяемой памяти), текущие характеристики фотона (хранятся в регистрах) и результирующие данные (имеют большой объем, хранятся в глобальной памяти). Разделяемая память и регистры являются быстрыми типами памяти, но имеют небольшой объем, в связи с чем использовать их для работы с результирующими данными не представляется возможным. Глобальная память, напротив, позволяет хранить большие объемы данных, но является относительно медленной. Однако в данной задаче обращений к результирующим данным немного, поэтому существенного влияния на производительность использование медленной памяти не оказывает.

Данные, относящиеся к результатам, являются общими для всех фотонов, откуда возникает необходимость обращения к этим данным одновременно из разных потоков, а значит, нужна синхронизация. В текущей реализации для этого используются атомарные операции, работающие с 64-битными целыми числами (для их применения версия compute capability графического адаптера должна быть 1.2 и выше).

Использование механизмов синхронизации обычно сильно уменьшает производительность приложений. Рассматриваемая задача не является исключением, однако здесь все зависит от размерности результирующих массивов, которая является входным параметром алгоритма. При малой размерности (100 элементов в каждом массиве) одновременных обращений к одним и тем же данным много, скорость работы приложения низкая. Но если увеличить число элементов каждого массива до 10 000 (тем самым повысив разрешающую способность задачи), то влияние синхронизации на производительность программы будет минимально.

Дополнительно следует отметить, что для решения требуемой задачи достаточно одинарной точности. А использование операций с одинарной точностью на GPU так же существенно ускоряет приложение. К тому же в данном случае возможно использование более быстрых, но менее точных математических функций, предоставляемых встроенной библиотекой CUDA.

Использование нескольких GPU

Технология CUDA позволяет использовать одновременно несколько графических адаптеров, установленных в рамках одного вычислительного узла. Каждый GPU имеет свой уникальный идентификатор, с помощью которого можно запускать выполнение задачи на выбранном устройстве. При этом для одновременной работы нескольких GPU необходимо создать на CPU определенное число параллельных потоков, чтобы каждый поток работал с собственным графическим адаптером. Способ создания потоков на центральном процессоре может быть любым, в частности в данной работе использовалась технология OpenMP.

В целях организации параллельной работы нескольких GPU, установленных на разных вычислительных узлах (например, в рамках вычислительного кластера), приведенный выше подход может быть обобщен. Для этого необходимо дополнительно применить одну из технологий параллельного программирования для систем с распределенной памятью, например, MPI [3].

Для использования нескольких графических адаптеров нам потребовалось организовать корректную работу с имеющимися данными:

- перед началом выполнения алгоритма создаются управляющие структуры, которые хранят данные для работы генератора случайных чисел (начальное смещение в генерируемой последовательности для конкретного устройства и общие данные генератора);
- перед началом выполнения алгоритма делается несколько копий структуры с информацией о моделируемой среде (входные данные задачи), каждая копия используется на своем GPU;
- структура состояния фотона создается и используется только в рамках вычислительного потока на графическом адаптере, поэтому не требует выполнения дополнительных действий при переходе на несколько GPU;
- промежуточные результаты накапливаются в памяти каждого из используемых графических адаптеров, а после окончания основного алгоритма суммируются на центральном процессоре.

Таким образом, рассматриваемый нами алгоритм позволяет создать работающую на нескольких GPU программу за небольшое количество достаточно простых шагов.

Результаты

В табл. 1 приведено время работы программы (в секундах) в зависимости от числа фотонов и вычислительного устройства.

Процессор: Intel Xeon E5520 2.27 ГГц, 4 ядра.
Количество ядер в процессоре: 4.
Количество процессоров: 2.
Объем оперативной памяти: 12 Гб.
Графический адаптер: Nvidia Tesla C1060.

Таблица 1. Время работы алгоритма (в секундах) в зависимости от числа фотонов и вычислительного устройства

Вычислительное устройство	Число фотонов				
	10^4	10^5	10^6	10^7	10^8
Xeon E5520, 1 ядро (1 core)	0,50	4,90	49,3	494	4951
2 x Xeon E5520, 8 ядер (8 cores)	0,11	0,94	7,60	74	730
Tesla C1060 (1 GPU)	0,06	0,11	0,60	5,13	50,7

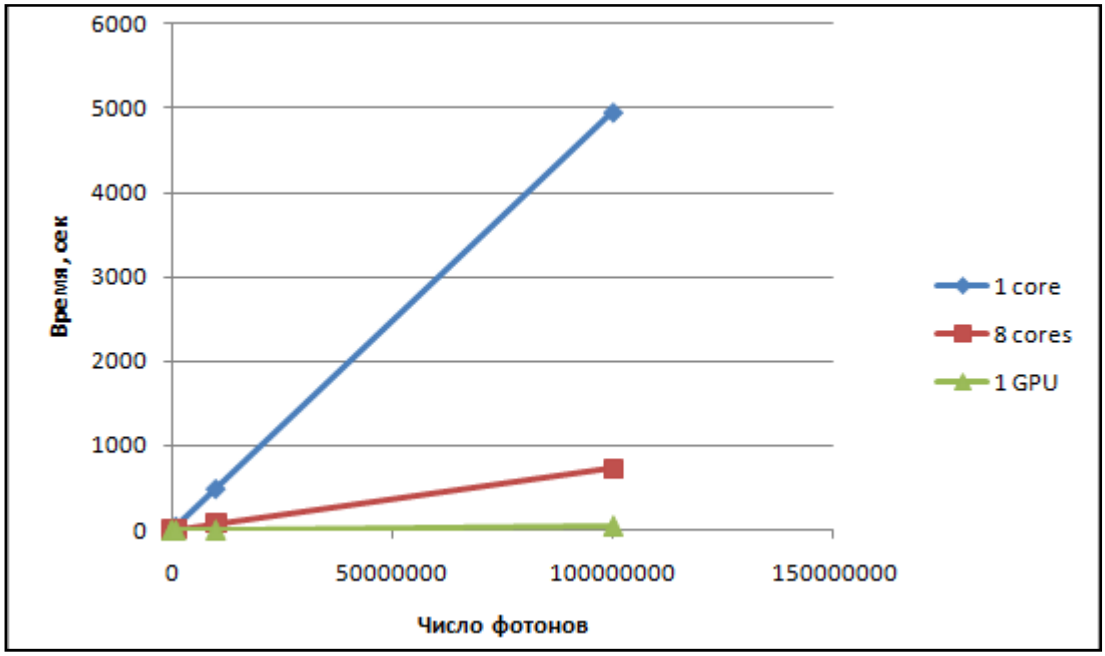


Рис. 2. Время работы алгоритма в зависимости от числа фотонов и вычислительного устройства

Таблица 2. Время работы алгоритма (в секундах) в зависимости от количества используемых GPU, число фотонов равно 10^8

Вычислительное устройство	Число используемых GPU			
	1	2	3	4
Tesla C1060	50.73	25.76	18.14	12.84

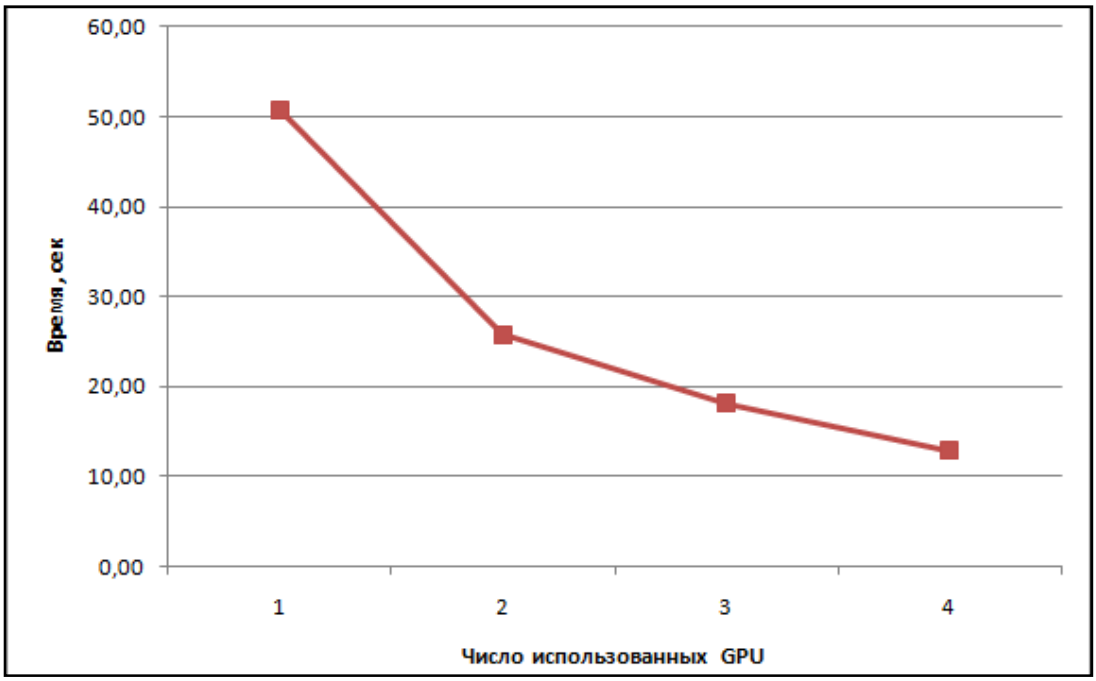


Рис. 3. Время работы алгоритма в зависимости от числа использованных GPU, число фотонов равно 10^8

Результаты проведенных экспериментов (табл. 1 и рис. 2) показывают, что использование графического адаптера и технологии CUDA существенно повышает производительность приложения, а потому является обоснованным.

А применение нескольких GPU для обработки данных (табл. 2 и рис. 3) позволяет дополнительно уменьшить время, необходимое для исполнения программы. Причем проведенные эксперименты говорят о хорошей масштабируемости задачи при одновременной работе на нескольких графических адаптерах.

Усовершенствование разработанного программного кода для расчета распространения излучения в средах со сложной геометрией позволит в перспективе эффективно использовать его как для моделирования работы различных систем оптической биомедицинской диагностики, так и для планирования лучевой терапии.

Авторы выражают благодарность Меерову Иосифу Борисовичу и Донченко Роману (Нижегородский госуниверситет им. Н.И. Лобачевского). Работа выполнена при поддержке ФЦП «Научные и научно-педагогические кадры инновационной России», проект № 02.740.11.0839.

Литература

1. LihongWang, Steven L. Jacques. Monte Carlo Modeling of Light Transport in Multi-layered Tissues in Standard C, 1992.

2. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. Издательство "ДМК Пресс", 2010.

3. Гергель В.П. Теория и практика параллельных вычислений. – М.: Интернет-Университет Информационных Технологий, Бином, 2007.

4. Соболь И.М. Численные методы Монте-Карло. Издательство "Наука", 1973.

Реализация метода Монте-Карло на CUDA для задач оптической биомедицинской диагностики

Фикс И.И., Кириллин М.Ю., Катичев А.Р.
Институт Прикладной Физики, РАН

В последнее время методы оптической биомедицинской диагностики получили широкое развитие, и многие из них переходят из класса экспериментальных в число применяющихся в клинической практике. Среди преимуществ методов оптической диагностики и визуализации следует отметить их неинвазивность, безопасность и экономичность (по сравнению с традиционными рентгеновскими методами и магнитно-резонасной томографией), и высокое пространственное разрешение (по сравнению с методами УЗИ). Ряд методов оптической диагностики при применении требуют восстановления трехмерного изображений по результатам регистрации рассеянного объектом излучения, а также интерпретации полученных изображений. Интегро-дифференциальное уравнение переноса излучения, описывающее распространение излучения в рассеивающих средах является довольно сложным, и в общем случае не имеет аналитического решения, что требует применения различных приближений или его решения численными методами.

Одним из эффективных методов численного моделирования распространения излучения является метод Монте-Карло [1]. Применительно к данной задаче он заключается в многократном повторении расчета случайной траектории движения фотона в среде исходя из задаваемых параметров среды (геометрии и оптических свойств) и последующем статистическом анализе полученных данных. Преимуществом данного метода перед различными аналитическими приближениями является возможность расчета интенсивности рассеянного излучения как вблизи источника, так и на больших расстояниях при любой заданной геометрии среды, в то время как областью применения аналитических приближений существенно ограничена.

Единственным недостатком метода Монте-Карло является его ресурсозатратность (так, для достижения удовлетворительной точности при моделировании распространения излучения в образце биоткани толщиной около 1 см обычно требуется порядка 10^8 случайных траекторий фотонов). Однако это ограничение может быть эффективно преодолено с помощью использования графических процессоров, позволяющих осуществлять многопотоковые параллельные вычисления [2].

Поскольку для вычисления случайные траектории фотонов являются независимыми друг от друга и определяются оптическими и геометрическими характеристиками объекта, их расчет может осуществляться параллельно без обмена данными между потоками. Этот факт делает реализацию ММК с применением графических процессоров очень выгодной.

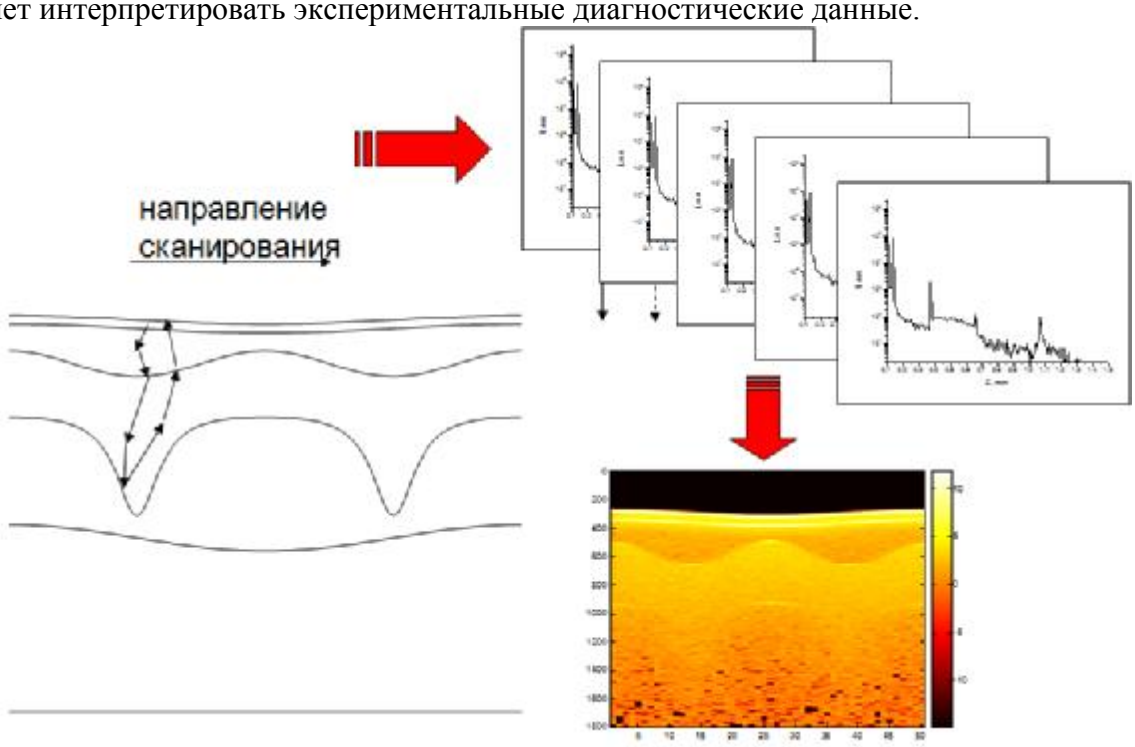
В настоящей работе метод Монте-Карло (ММК) применяется для:

- Расчета характеристик светового поля на длинах волн возбуждения и флуоресценции в задаче диффузионной флуоресцентной томографии (ДФТ).
- Моделирования изображений биотканей, получаемых методом оптической когерентной томографии.
- Расчета максимальной глубины визуализации структуры биотканей методом многофотонной флуоресцентной микроскопии.

Диффузионная флуоресцентная томография представляет собой современный оптический метод визуализации внутренней структуры оптически неоднородных сред и объектов с использованием флуоресцентных маркеров [3]. Флуоресцентные маркеры различной природы используются для диагностики онкологических заболеваний, исследований молекулярных процессов, типичных для канцерогенеза, исследования метастазирования и ответа на противоопухолевую терапию [4]. Для точного определения местоположения и реальных размеров флуоресцирующей опухоли, находящейся в

лабораторном животном на большой глубине (глубже кожного покрова), используют методы ДФТ [5]. В методе ДФТ облучение исследуемого объекта происходит на длине волны возбуждения флуоресцирующих веществ (маркеров), а детектирование сигнала - в спектре их флуоресценции. Для реконструкции распределения флуорофора в ткани в настоящее время используются специальные алгоритмы, учитывающие диффузное распространение света [6,7]. Метод Монте-Карло применительно к ДФТ может применяться для решения следующих задач: расчет распространения света внутри объекта со «сложной» геометрией; проведение численных экспериментов (моделирования результатов эксперимента) для апробации алгоритмов восстановления; восстановление оптических свойств среды (коэффициентов поглощения, рассеяния, фактора анизотропии).

Оптическая когерентная томография (ОКТ), основанная на принципах низкокогерентной интерферометрии [8], позволяет получать изображение поверхностных слоев биотканей на глубинах до 2 мм с более высоким по сравнению с традиционными (УЗИ, рентген) методами пространственным разрешением (до единиц микрон). Формирование ОКТ-изображений основано на визуализации двух- или трехмерного распределения оптических свойств внутри биоткани, что позволяет регистрировать не только морфологические, но и функциональные изменения в биотканях, связанные с локальным изменением их оптических свойств. Метод Монте-Карло применяется для моделирования ОКТ-изображений различных объектов [9], что позволяет интерпретировать экспериментальные диагностические данные.



Двухфотонная микроскопия (ДФМ) - вид лазерной сканирующей микроскопии, использующей двухфотонно возбуждаемую флуоресценцию для визуализации внутренних структур биологических тканей [10]. Данный вид микроскопии отличается высокой проникающей способностью и низкой фототоксичностью. МК моделирование применительно к ДФМ применялось для расчета распространения сфокусированных гауссовых пучков в рассеивающих средах. Такие расчеты использовались для проверки вновь разработанных аналитических моделей рассеяния пучка в рассеивающих средах, а так же для оценки эффективности возбуждения сигнала двухфотонной флуоресценции в зависимости от глубины фокусировки излучения накачки [11].

Ниже представлены характерные времена, требуемые для проведения расчета метода Монте-Карло на GPU (Nvidia GeForce 260) с использованием технологии CUDA и CPU (AMD Phenom II x4 920, 2.7 ГГц, однопотоковая реализация).

Количество траекторий	Время счета на GPU, с	Время счета на CPU, с
10^7	1.2	354
10^8	10.2	3614
10^9	101.8	36895

Таблица 1. Сравнение времени расчета при реализации ММК на GPU и CPU

Особенностью реализации алгоритма является то, что при использовании технологии CUDA за один вызов ядра GPU просчитывается около $6 \cdot 10^6$ различных случайных траекторий фотонов.
Таким образом, было показано, что реализация метода Монте-Карло с использованием технологии CUDA позволяет существенно ускорить вычисления, что открывает возможности для более эффективного применения этого метода в задачах биомедицинской оптической диагностики.

Работа выполнена при финансовой поддержке грантов РФФИ 10-02-00744-а и 11-02-01129-а, гранта Президента Российской Федерации для государственной поддержки молодых российских ученых - кандидатов наук МК-1127.2010.2 и ФЦП "Научные и научно-педагогические кадры инновационной России" на 2009 - 2013 годы (ГК 02.740.11.0839).

Литература

1. Wang L., Jacques S. L., Zheng L. “MCML-Monte Carlo modeling of light transport in multi-layered tissues”, *Comput Methods Programs Biomed*, **47**(2), 131-146, (1995).
2. Alerstam E., Svensson T., Andersson-Engels S. Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration, J Biomed Opt, 13(6), 060504 (2008).
3. S.V. Patwardhan, S.R. Bloch, S. Achilefu, J.P. Culver, “Time-dependent whole-body fluorescence tomography of probe bio-distribution in mice”, *Optics Express*, **13**, 2564-2577 (2005).
4. V. Ntziachristos, E.A. Schellenberger, J. Ripoll, D. Yessayan, E. Graves, A. Bogdanov, L. Josephson, and R. Weissleder “Visualization of antitumor treatment by means of fluorescence molecular tomography with an annexin V–Cy5.5 conjugate”, *PNAS*, **101**, 12294–12299 (2004).
5. V. Ntziachristos, R. Weissleder “Experimental three-dimensional fluorescence reconstruction of diffuse media by use of a normalized Born approximation”, *Optics Letters*, **26**, 893-895 (2001).
6. Ф. Натеррер, *Математические аспекты компьютерной томографии*. М.: Мир, 1990.
7. A. Soubret, V. Ntziachristos “Fluorescence molecular tomography in the presence of background fluorescence”, *Physics in Medicine Biology*, **51**, 3983-4001 (2006).
8. B. E. Bouma, G. J. Tearney (eds.), *Handbook of optical coherence tomography* (Marcel Dekker,New York, 2002).
9. M. Kirillin, I. Meglinski, V. Kuzmin, E. Sergeeva and R. Myllylä “Simulation of optical coherence tomography images by Monte Carlo modeling based on polarization vector approach”, *Optics Express*, **18**(21), 21714-21724 (2010).
10. F. Helmchen, W. Denk. “Deep-tissue two-photon microscopy”, *Nature Methods*, **2**, (12), 932-940 (2005).
11. Е.А. Сергеева, А.Р. Катичев, М.Ю. Кириллин, “Формирование сигнала двухфотонной флуоресцентной микроскопии в условиях сильного рассеяния: теоретическое и численное моделирование”, *Квантовая Электроника*, **40**(12), 1053–1061 (2010).

Сравнение вычислительных возможностей графических ускорителей NVidia при решении различных классов задач

Кривов М.А.[(1)], Казеннов А.М.[(2)],
[(1)] *Издательство SCR-Media, журнал «Суперкомпьютеры»,*
[(2)] *Московский Физико-Технический Университет*

Введение

Современные графические ускорители обладают не только колоссальными вычислительными возможностями, но и достаточно сложной архитектурой, затрудняющей их эффективное использование при решении задач из некоторых областей. При адаптации алгоритмов под графические ускорители, достаточно часто узким местом оказывается не производительность GPU, а недостаточный объём какого-либо типа памяти, затраты на подготовку данных центральным процессором или неэффективность использования потоковых процессоров.

В данной работе, являющейся расширенной версией статьи [1], проведено тестирование четырёх видеокарт от Nvidia из различных сегментов с целью оценки реально достижимой производительности при решении разных классов задач. В качестве тестов были выбраны GPGPU-бенчмарки SHOC и Rodinia, программы, использующие библиотеки MAGMA и CUBLAS, а также примеры из Nvidia CUDA SDK. В частности, были проведены сравнения производительности при использовании технологий OpenCL и NVidia CUDA, вычислениях с двойной и одинарной точностью и выполнении различных реализаций одного и того же алгоритма.

Тестовые системы

Тестирование проводилось на графическом кластере МФТИ, каждый узел которого оснащён двумя 6-ядерными процессорами AMD Opteron 2427 @2.2 GHz и 16 гигабайтами памяти типа DDR3. В качестве графических ускорителей использовались следующие видеокарты NVidia:

GPU	Пиковая производительность (float / double), GFlops	Объём памяти, GB	Количество CUDA-ядер	Частота CUDA-ядер, MHz
Tesla C2050	1030,4 / 515,2	3	448	1147
Tesla C1060	933 / 78	4	240	1296
GeForce 480 GTX	1344,96 / 168	1,5	480	1401
GeForce 295 GTX	2x894,2 / 2x74,5	1,75	2x240	1242

Таблица 1. Описание тестируемых ускорителей

Графические ускорители Tesla C2050 и GeForce 480 являются представителями современной архитектуры GPU под названием Fermi, результатом чего является их более высокая производительность при операциях с двойной точностью и поддержка дополнительной функциональности (архитектурных расширений 2.0). Две другие видеокарты (Tesla C1060 и GeForce 295) являются модификациями более старой архитектуры GT200, представленной в 2008 году и предназначенной для проведения расчётов с одинарной точностью.

Стоит также отметить, что графический ускоритель GeForce 295 состоит из двух относительно независимых процессоров и программно распознаётся как две видеокарты, поэтому при тестировании использовалась только половина доступных вычислительных возможностей данного ускорителя.

Тесты SHOC

Пакет тестов Scalable HeterOgeneous Computing (или SHOC) [2], разрабатывается известной американской лабораторией ORNL как независимый бенчмарк для гетерогенных систем, оснащённых графическими ускорителями. Все тесты логически разбиты на три уровня, определяющие их приближенность к реальности. Так, приложения из уровня 0 являются синтетическими тестами и нацелены на получение максимально возможной производительности, в то время как тесты из уровня 2 реализуют решения более реальных задач и, соответственно, показывают более скромные результаты.

Стоит отметить, что SHOC поддерживает как технологию NVidia CUDA, так и OpenCL, а начиная с версии 1.01 появилась возможность запуска бенчмарка на гетерогенных кластерах (с помощью библиотеки MPI). Результаты запуска некоторых тестов из CUDA-версии для одного узла приведены в следующей таблице:

GPU	**MaxFlops** (float/double) GFlops	**FFT** (float/double) GFlops	**GEMM** (float/double) GFlops	**Bandwidth** GB/s			**MD** (float/double) GB/s	**Reduction** (float/double) GB/s
				Global	Texture	Shared		
Tesla C2050	1002 / 501	69 / 34,3	301,5 / 68	93,1	73,6	368,5	74 / 84,5	61,2 / 65
Tesla C1060	721,9 / 77	94 / 26,7	202 / 38,1	77,5	66	229,2	59,1 / 27	46 / 42,4
GeForce 480 GTX	1313 / 168	202 / 63,5	366,6 / 84	152,3	91,6	489,9	91,5 / 75	83 / 86,6
GeForce 295 GTX	691 / 74,2	93 / 25,7	200,2 / 38	87,9	54,4	219,6	51,5 / 26	50 / 42,2

Таблица 2. Результаты тестов SHOC (CUDA-версия)

В тестах *MaxFlops* и *Bandwidth* (принадлежащих уровню 0) подбирается такое вычислительное ядро, которое позволяет получить максимальные значения соответствующих характеристик. Если сравнить достигнутые значения с теоретической пиковой производительностью, то можно заметить, что они практически идентичны — в большинстве случаев разность не превосходит нескольких процентов.

Тесты *FFT* (быстрое преобразование Фурье) и *GEMM* (перемножение матриц) оценивают производительность GPU на достаточно вычислительноёмких задачах, в то время как в тестах *MD* (задача N тел) и *Reduction* (свёртка массива) узким местом является память.

При запуске OpenCL-версий тестов были получены следующие результаты:

GPU	**MaxFlops** (float/double) GFlops	**FFT** (float/double) GFlops	**GEMM** (float/double) GFlops	**Bandwidth** GB/s			**MD** (float/double) GB/s	**Reduction** (float/double) GB/s
				Global	Image	Local		
Tesla C2050	1005 / 503	40,4 / 17	274 / 51,6	95	72,4	371,8	25 / 27,7	34 / 36,6
Tesla C1060	727 / 77,6	23 / 7, 1	133 / 20,5	81,9	66,2	262,6	23 / 22,6	26 / 24,9
GeForce 480 GTX	1317 / 168	52,1 / 17	334 / 52,9	164	98,2	486,5	30,7 / 34	45 / 47,2
GeForce 295 GTX	696 / 74,3	26 / 6, 9	136,6 / 20	94,3	54,7	251,7	22 / 21,7	26 / 23,8

Таблица 3. Результаты тестов SHOC (OpenCL-версия)

Как видно, OpenCL-тесты в сравнении с CUDA-аналогами показывают либо схожую производительность, либо в несколько раз меньшую. Однозначно объяснить данное явление достаточно сложно — можно лишь заметить о меньшей «зрелости» технологии и специфики бенчмарка, который, возможно, ещё не достаточно оптимизирован (отметим, что первая стабильная версия появилось в декабре 2010 года).

Тесты MAGMA и CUBLAS

Библиотеки MAGMA [3] и CUBLAS являются реализациями ставшего фактически стандартом интерфейса BLAS, содержащего основные операции линейной алгебры. Обе эти библиотеки используют технологию NVidia CUDA и хорошо оптимизированы под графические процессоры. Ниже приведены результаты сравнения для четырёх операций:

GPU	**GEMM** (float / double), GFlops		**GEMV** (float / double), GFlops		**GESV** (float / double), GFlops		**SYMV** (float / double), GFlops	
	MAGMA	CUBLAS	MAGMA	CUBLAS	MAGMA	CUBLAS	MAGMA	CUBLAS
Tesla C2050	562 / 172	561,6 / 174	42,7 / 22	45,7 / 20	320 / 142,5	- / -	50 / 31	15 / 12, 4
Tesla C1060	365 / 74	193 / 74	39 / 21,3	40 / 16,5	288,3 / 66	- / -	60 / 23,4	16,4 / 3,7
GeForce 480 GTX	740,2 / 160	740,6 / 159	75 / 30	64 / 32	408,5 / 140	- / -	67 / 43,1	22, 5 / 15,2
GeForce 295 GTX	302 / 71,4	168 / 71,3	49 / 23	48 / 18	280,9 / 64	- / -	67 / 22,8	16,2 / 3,6

Таблица 4. Результаты тестирования библиотек MAGMA и CUBLAS

Стоит сразу отметить, что размер матриц и векторов выбирался с целью получения максимальной производительности. Так, в тесте *GEMM* проводилось перемножение двух матриц размером 1600 на 1600. Аналогично, тест *GEMV* заключался в умножении вектора размерности 9949 на соответствующую матрицу. В тесте *GESV* решалась СЛАУ с квадратной матрицей размерностью 10112, а в тесте *SYMV* проводилось перемножение симметричной матрицы размерности 8221 на вектор и последующее сложение с другим вектором.

Если сравнивать конкурирующие библиотеки MAGMA и CUBLAS, то однозначно определить более эффективную реализацию BLAS для технологии CUDA невозможно — в зависимости от размера данных, более быстрой является то одна библиотека, то другая. Стоит заметить, что обычно MAGMA более эффективно работает на больших данных (размером около 1 гигабайта), в то время как CUBLAS чаще выигрывает при работе с данными порядка 100 мегабайт, а также на ускорителях с архитектурой Fermi.

Тесты RODINIA

Бенчмарк Rodinia [4] был разработан группой исследователей из Вирджинского университета и является сборкой вычислительноёмких CUDA- и OpenMP-ядер, созданных раннее авторами бенчмарка в рамках различных исследований. В связи с его внутренней разнородностью, для данного тестирования была переписана система замеров времени для унификации результатов. Ниже приведено время работы некоторых ядер бенчмарка (затраты на копирование данных не учитываются):

	BFS msec	**CFD** sec	**HeartWall** msec	**HotSpot** msec	**LUD** msec	**NW** sec	**StreamCluster** sec
Предметная область	*Графы*	*Аэро-динамика*	*Медицина*	*Физика*	*Линейная алгебра*	*Био-информатика*	*Поиск данных*
Tesla C2050	15,4	6,6	201	6,64	2,1	2,2	16,4
Tesla C1060	18,8	6,6	360	6,6	2	2,1	12,4
GeForce 480 GTX	10,1	5,05	131,2	6,5	2,2	2,3	12,27
GeForce 295 GTX	18,26	8,9	268	6,4	2	2,2	13,7

Таблица 5. Результаты тестов Rodinia

Подробное описание каждого теста может быть найдено в [5]. Как видно из результатов, время работы большинства ядер составляет миллисекунды (хотя тестирование проводилось на

доступных данных максимального размера). Поэтому выигрыш от использования более новых графических ускорителей незаметен — узким местом уже становится копирование памяти и накладные расходы на запуск ядер. Возможно, это является следствием того, что бенчмарк разрабатывался в 2007-2009 годах и «затачивался» под менее производительные ускорители. В любом случае, самой производительной видеокартой оказалась GeForce 480 GTX, на одном тесте даже обходящая специализированную Tesla C1060 в 2.74 раза.

Примеры из NVidia CUDA SDK

Последним тестом, проведённом в данной работе, является запуск некоторых примеров из NVidia GPU Computing SDK. Будучи разработанными сотрудниками NVidia, они достаточно хорошо оптимизированы и большинстве случаев учитывают особенности используемого поколения архитектуры GPU.

GPU	**Black-Scholes**		**Monte Carlo** msec	**Fast Walsh Transform** msec	**Radix sort**		**Sorting networks** msec
	Time, msec	Bandwidth GB/s			Int, msec	Float, msec	
Tesla C2050	2,8	27,6	3,4	17,5	47,5	47,1	14,8
Tesla C1060	1	80,5	3,78	26,2	49,7	48,4	22
GeForce 480 GTX	1,9	40,6	2,18	12,3	47,7	47,4	10,6
GeForce 295 GTX	0,9	86,5	3,2	22,6	50,1	48,6	20,5

Таблица 5. Результаты тестирования примеров из NVidia CUDA SDK

Результаты данных тестов соответствуют синтетическим бенчмаркам — новые графические ускорители оказываются существенно быстрее предшественников. Исключением является пример Black-Scholes, при выполнении которого старые Tesla C1060 и GeForce 295 GTX обходят более мощные видеокарты, что вызвано «излишней» оптимизацией.

Результаты

Если обобщать полученные результаты, то в первую очередь стоит сравнить эффективность конкурирующих API для программирования под графические ускорители. Ниже приведено сравнение технологий NVidia CUDA и OpenCL, полученное усреднением схожих тестов из бенчмарка SHOC (одинарная точность):

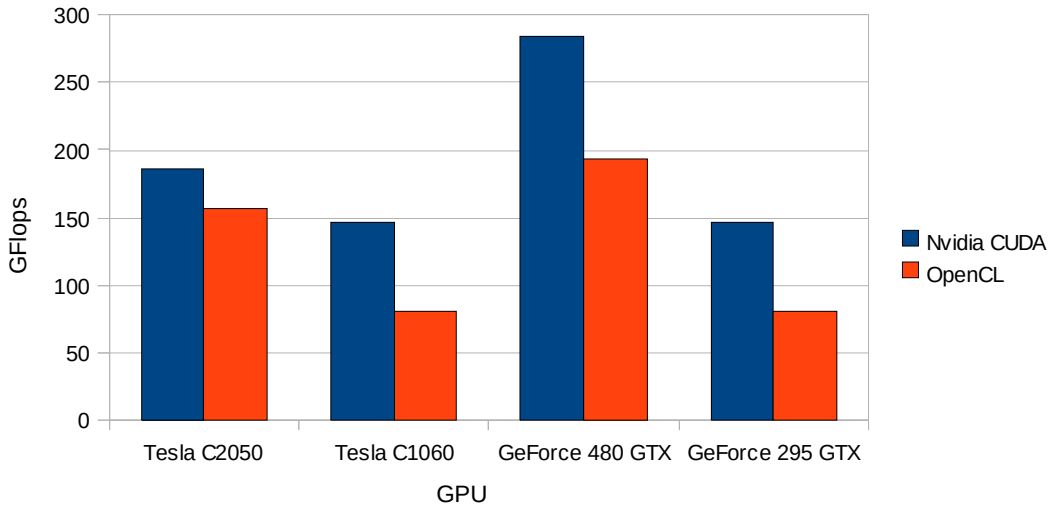


График 6. Сравнение технологий NVidia CUDA и OpenCL

Как видно из графика, NVidia CUDA стабильно обходит ещё молодой стандарт OpenCL, причём отрыв (в процентном соотношении) слабо зависит от используемой видеокарты.

Другим интересным результатом является сравнение производительности различных графических ускорителей при операциях с одинарной и двойной точностью, приведённое ниже:

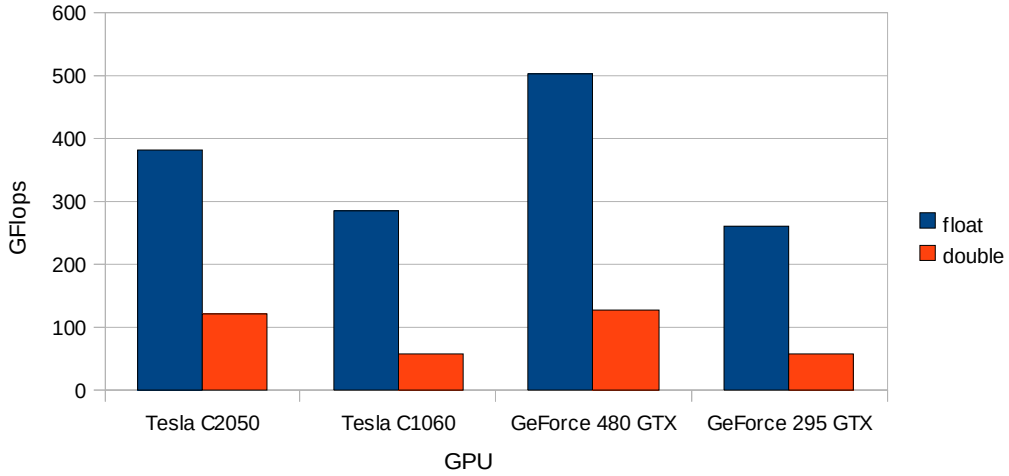


График 7. Сравнение производительности при операциях с двойной и одинарной точностью

Даже у графических ускорителей с архитектурой Fermi, для которой заявлена полноценная поддержка двойной точностью, производительность при выполнении double-операций падает более чем в 3-4 раза. Однако стоит заметить, что при этом они в разы превосходят представителей устаревающей архитектуры GT200.

Последнем сравнением является запуск различных реализаций одного и того же алгоритма (SGEMM из BLAS), содержащихся в разных библиотеках. Результаты работы при перемножении матриц сопоставимой размерности приведены ниже:

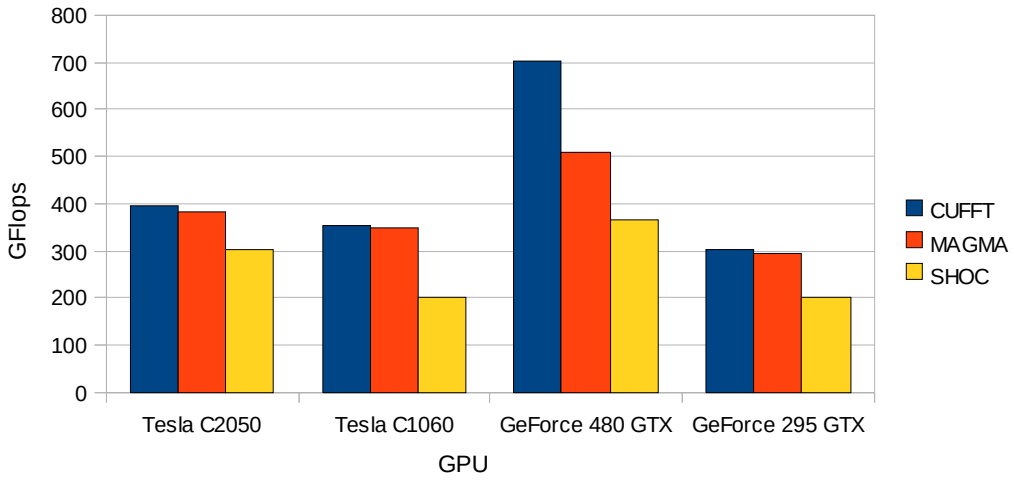


График 8. Сравнение производительности различных реализаций SGEMM

В данном случае оказалось, что производительность всех реализаций практически сопоставима, а разрыв между ними зависит больше от обрабатываемых данных.

Список литературы

1) Кривов М.А., Казеннов А.М., GeForce или Tesla? // Журнал «Суперкомпьютеры», Зима 2010, с. 42-43.
2) Danalis и др., The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite // The Third Workshop on General-Purpose Computation on Graphics Processors (GPGPU 2010). Март 2010.
3) Электронный ресурс http://magma.maths.usyd.edu.au/magma.
4) S. Che и др., Rodinia: A Benchmark Suite for Heterogeneous Computing // IEEE International Symposium on Workload Characterization (IISWC), Октябрь 2009, С. 44-54,
5) Электронный ресурс https://www.cs.virginia.edu/~skadron/wiki/rodinia.

Использование GPU для решения задач компьютерного зрения в библиотеке OpenCV

Анатолий Бакшеев, Владислав Виноградов, Виктор Ерухимов,
Кирилл Корняков, Алексей Спижевой
Itseez Ltd,

anatoly.baksheev@itseez.com,vlad.vinogradov@itseez.com, victor.eruhimov@itseez.com,
kirill.kornyakov@itseez.com, alexey.spizhevoy@itseez.com

Аннотация

В данной статье рассматривается рассказывается об использовании GPU для решения задач компьютерного зренияв библиотеке OpenCV. Описывается текущее состояние разработки, некоторые технические подробности и примеры практического применения. Для отдельных алгоритмов приводятся данные об ускорении по сравнению с центральным процессором. В качестве примера приведено описание реализации алгоритма стереозрения на GPU.

Введение

В последнее время вычислительные мощности графических процессоров (GPU)значительно выросли, в связи с чем возможность использования этих мощностей для вычислительных задач общего назначения становится все более заманчивой. Сегодня мы можем наблюдать большую активность в данном направлении[1]. Компьютерное зрение является одной из областей, в которых можно эффективно использовать GPU для широкого круга актуальных задач, включая детектирование лиц и пешеходов, восстановление трёхмерных поверхностей, распознавание объектов для робототехники и визуальной одометрии. Многие алгоритмы такого рода можно эффективно распараллеливать по данным и выполнять на GPU с высокой скоростью.

Область компьютерного зрения достаточно обширна, в ней действует большое число научных и промышленных коллективов, решающих самые разнообразные задачи на различных уровнях. И ключевым фактором, определяющим то, насколько активно будут использоваться графические процессоры в будущем, является наличие удобных программных инструментов. Необходима программная инфраструктура, на основе которой создавались бы новые разработки в области компьютерного зрения. В качестве такого инструмента предлагается библиотека OpenCV с поддержкой GPU.

Ранее уже предпринимались попытки использовать видеокарты для решения задач в области компьютерного зрения. Примером могут служить библиотеки CUVI Lib[2], GPU4VISION[3], GPUCV[4], openVidia[5], NPP[6]. Кроме того, существует некоторое количество независимых реализаций отдельных алгоритмов. В большинстве своем эти разработки ведутся энтузиастами, ресурсы которых ограничены.Многим из них приходится выполнять дублирующую работу.

Поэтому существует потребность в профессиональном инструменте, который объединил бы в себе лучшие в своем классе алгоритмы. OpenCV – это кроссплатформенная библиотека алгоритмов компьютерного зрения, являющаяся стандартом де-факто в индустрии [7]. В 2010 году компанией Itseez при поддержке NVidia проект разработки GPU модуля для OpenCV. Этот модуль представляет собой набор функций и классов, позволяющих специалистам в области компьютерного зрения быстро начать использовать возможностей видеокарт.

Поддержка GPU в OpenCV

Общее описание

GPU модуль OpenCV содержит в себе несколько уровней функциональности. На нижнем уровне находятся реализации служебных операций, таких как инициализация и управление GPU, работа с памятью, механизма синхронных вызовов. Уровнем выше реализован широкий набор базовых функций обработки изображений: различные методы фильтрации, поиск максимума, аффинные преобразования, вычисление разности изображений в различных нормах и т.д.Список этих алгоритмов постоянно обновляется, в целом они призваны облегчить разработку алгоритмов компьютерного зрения самого высокого уровня. К последним можно отнести несколько существующих в OpenCV реализаций:

- Стереозрение:
  - BlockMatching[8]
  - BeliefPropagation[9]
  - Constant Space BeliefPropagation[10]
- HistogramsofOrientedGradients и линейныйSVM[11] – детектирование пешеходов и других объектов.
- Viola-Jonescascadeclassifier[12] – детектирование человеческих лиц, логотипов и т.д.
- Speed-uprobustfeatures (SURF)[13] – поиск ключевых точек на изображении для склеивания изображений, распознавания текстурированных объектов и т.д.

Использование GPU модуля

ВOpenCVконтейнером для хранения данных (в том числе изображений) служит класс `Mat`. По аналогии, в GPU модуле реализован класс `GpuMat`, практически с той же функциональностью, но хранящий данные в видеопамяти. Библиотека OpenCVпредоставляет удобные средства обмена информацией между GPU и CPU.

Основную часть GPU модуля составляют функции, имеющие интерфейс, идентичный CPU части OpenCV, с тем лишь отличием, что они принимают на вход объект типа `GpuMat`.

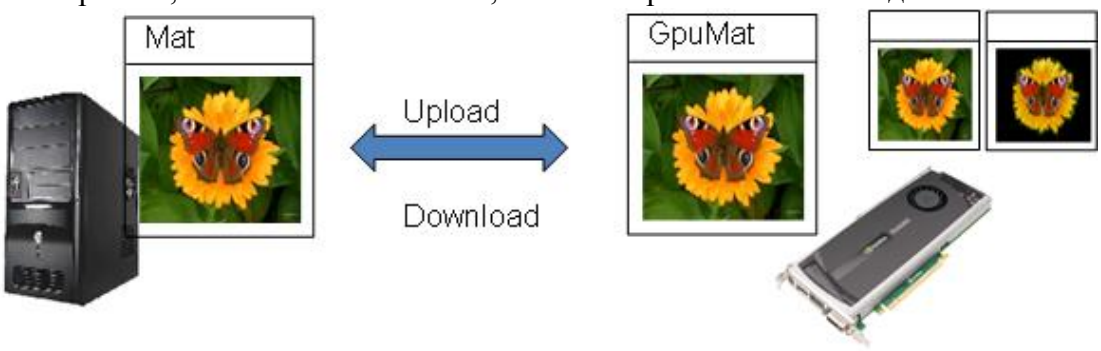


Рис. 1 Хранение изображений на видеокарте.

Программный интерфейс модуля (API) создан максимально близким к интерфейсу CPU части OpenCV, что облегчает перенос существующего кода на GPU. Однажды загрузив изображение на видеокарту, можно использовать различные функции обработки, которые будут производиться целиком на видеокарте.

Ниже приводится пример кода, который получает изображение с камеры и запускает на нем алгоритм поиска пешеходов. В качестве алгоритма используется HistogramsofOrientedGradients (HOG) [11]. Различия между GPU и CPU версиями выделены полужирным начертанием:

```
Mat frame;
VideoCapturecapture(camera);
cv::HOGDescriptor hog;
hog.setSVMDetector(cv::HOGDescriptor::
getDefaultPeopleDetectorector());
capture>> frame;


vector<Rect> found;
hog.detectMultiScale(frame, found,
    1.4,  Size(8, 8), Size(0, 0), 1.05, 8);
```

```
Mat frame;
VideoCapturecapture(camera);
cv::gpu::HOGDescriptor hog;
hog.setSVMDetector(cv::HOGDescriptor::
getDefaultPeopleDetectorector());
capture>> frame;

GpuMatgpu_frame;
gpu_frame.upload(frame);
vector<Rect> found;
hog.detectMultiScale(gpu_frame, found,
    1.4, Size(8, 8), Size(0, 0), 1.05, 8);
```

Рис. 2 Пример исходного кода.

Сравнение производительности

Для тестов производительности использовалась следующая конфигурация: четырехядерный Intel Core i5-750 2.66ГГц и NVidiaGeForce GTX470 с 448 ядрами на архитектуре Fermi. Полученные ускоренияна GPUпо сравнению с CPUверсиями представлены в таблице ниже.

Таблица1. Ускорение на GPUпо сравнению с CPU.

Stereo Block Matching (BM)	7x
Stereo Belief Propagation (BP)	10-20x
Stereo Constant Space BP (CSBP)	50-100x
HOG	3.5-7x
SURF	12x
CascadeClassifier	6-10x

При сравнении использовались GPU и CPU версии алгоритмов из OpenCV, кроме BP и CSBP,для которых сравнение велось с авторским кодом.OpenCV содержит реализации алгоритмов, которые длительное время оптимизировались профессиональными разработчиками, что обеспечивает корректность сравнения. Библиотека OpenCVбыла собрана с поддержкой многопоточности (на основе Intel TBB), поэтому все ядра процессора были задействованы.

Следует отметить, что прирост производительности базовых функций составляет в среднем ~50x. Объяснение этому состоит в том, что простые алгоритмы могут быть более эффективно распараллелены по данным. Для получения подробной информации о скорости работы примитивов и некоторых функций высокого уровня, следует запустить приложение `performance_gpu` из набора примеров OpenCV.

Гибридные системы

Известно, что не все вычисления могут быть успешно распараллелены. Существуют чисто последовательные алгоритмы, эффективная реализация которых на GPU невозможна. Примером может служить фильтрациямалых связанных компонент на изображении (speckle filtering). Подобные операции приходится производитьна центральном процессоре. Однако,в этом случае возникают накладные расходы при передаче данных между CPU и GPU.В некоторых случаях, доля накладных расходов может быть весьма существенной.

Для минимизации подобных издержек в OpenCV предлагается механизмасинхронных вызовов. При использовании ассинхронной функции, она возвращает управление сразу, в то времякак результат может быть готов лишь через некоторое время. Чтобы узнать о готовности результата следует использовать класс `Stream`.Подобные средства позволяют достаточно просто использовать GPU и CPU одновременно.

Использование нескольких GPU

Возможность использования нескольких видеокарт также является достаточно востребованной.Однако,в OpenCV реализации алгоритмов ориентированы на использование одного GPU. Чтобы использовать несколько GPU, пользователь должен сам распределять работу между ними, используя для этих целей класс `MultiGpuManager`.

Если имеется большая база изображений, то легко можно разбить всю работу по нескольким видеокартам. В случае же одного изображения распараллеливание работы не всегда возможно. Почти для всех примитивных функций время работы алгоритма достаточно мало, и накладные расходы на пересылку данных будут составлять существенную долю времени, что делает такое распараллеливание неэффективным. В случае высокоуровневых алгоритмов, использовать несколько GPU для работы над одним изображением – вполне допустимо, поскольку время обработки много больше времени пересылки.

Например, для алгоритма Stereo Block Matching можно разрезать картинку пополам с небольшим перекрытием, вычислить результат на каждом GPU независимо, а затем склеить результаты. Производительность для двух GPU по результатам запусков составляет 180% от

Рис. 3 Использование двух GPU в алгоритме стереозрения.

варианта с одним GPU.

Еще один пример –задача детектирования пешеходов. Поскольку люди на кадре могут быть различного размера в зависимости от расстояния до камеры, алгоритм запускается на различных масштабах изображения. Каждый масштаб обрабатывается независимо, после чего результаты объединяются. Таким образом, обработку набора масштабов можно разделить между несколькими GPU, получаемое ускорение пропорционально количеству используемых GPU.

Использование GPUв задаче стереозрения

Постановка задачи и описание алгоритма

Задача стерео сопоставления состоит в поиске соответствий между точками на двух

Рис. 4 Использование нескольких GPU в алгоритме детектирования пешеходов.

изображениях (левого и правого), полученных при помощи двух камер или одной стереокамеры. Найдя соответствие для каждого пикселя в виде вектора смещения, можно вычислить расстояние до объектов на сцене, при условии, что известно взаимное расположение камер и их параметры. Задача поиска пары для точки левого изображения может быть сведена к задаче поиска вдоль горизонтальной прямой на правом изображении. Тогда для каждого пикселя достаточно знать лишь величину смещения вдоль оси абсцисс (disparity). Оптимальное соответствие между точками определяется на основе сравнения окрестностей пикселов левого и правого изображений. Классическим методом решения данной задачи является алгоритм Stereo Block Matching (block-matching). Данный алгоритм достаточно хорошо изучен, подробная информация может быть найдена в работах [8], [15].

Важно отметить, что алгоритм block-matchingбыл уже неоднократно портирован на GPU[14][16][17][18]. Настоящая реализация основывается на одной из ранних работ и представляет собой модифицированную версию алгоритма, описанного в [13]. Детали и общее описание алгоритма приводятся в оригинальном источнике, поэтому здесь в основном будут рассмотрены отличия от оригинальной версии. Главными усовершенствованиями являются следующие моменты: более эффективное вычисление корреляционных сумм, обеспечение кеш-эффективности, фильтрация низкотекстурированных областей. Незначительные усовершенствования, такие как более экономичный формат хранения, в данной статье не описываются, однако могут быть найдены и исходных кодах библиотеки OpenCV.

Далее рассмотрим каждое из усовершенствований отдельно.

Вычисление корреляционных сумм

Вычисление корреляционных сумм – это центральная часть алгоритма, целью которой является определение похожести двух блоков пикселей. В оригинальной версии алгоритма вычисления, производимые различными потоками, значительно перекрывались. Фактически, соседние потоки на GPU суммировали элементы двух одинаковых векторов, отличающиеся лишь одним элементом. В предложенной реализации потоки делят вектор на несколько частей, и каждый суммирует лишь свой участок, оставшуюся же часть получает у других потоков.

Обеспечение кеш-эффективности

Алгоритм поиска соответствий на GPU устроен таким образом, что блоки потоков читают области изображения, сильно пересекающиеся друг с другом. Данное обстоятельство наводит на мысль о том, что повторные чтения должны производиться уже из кеша L1 на GPU. Однако, согласно показаниям CUDAVisualProfiler, доступ к глобальной памяти являлся главным узким местом описанной реализации. Подобная ситуация объясняется тем, что мы вынуждены хранить в кеше одновременно несколько строк изображения. Это приводит к тому, по мере добавления новых строк ранее прочитанные оказываются выброшенными из кеша. Таким образом, переход к новым итерациям в цикле по значению смещения (d)означает повторное чтение строк изображения из глобальной памяти. Тривиальные расчеты подтверждают данную гипотезу (Рис. 5).Размер части карты смещений, вычисляемой блоком потоков, равен blockWidth x RowPerThread=128x21. Тогда для вычисления корреляционных сумм пикселей этой части, размер кусков входных изображений, необходимых при каждом значении счетчика цикла по d, равен:

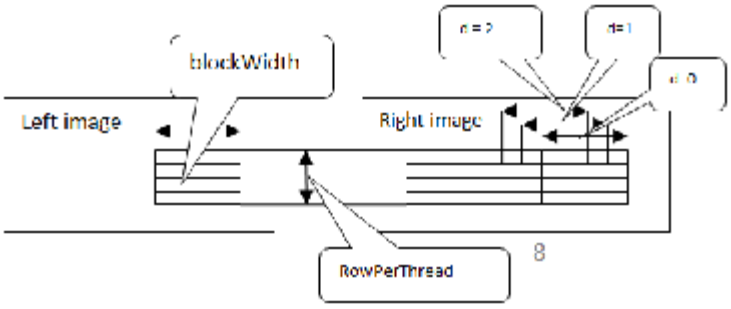


Рис. 5 Вычислительная схема алгоритма.

(blockWidth +winSize– 1) RowPerThread * 2 * 8 blocks permultiprocessor =*

*= (128 + 18) * 21 *2 * 8 = 49kb > 48kb of cache per multiprocessor*

Данные расчеты приведены для видеокарты NVidia GeForce GTX470, построенной на архитектуре Fermi. Для улучшения кеш-эффективности было использовано развертывание цикла по смещениям, в котором определяется оптимальное соответствие пикселов левого и правого изображений. Была использована глубина развертки, равная 8, таким образом, происходил одновременных расчет 8 корреляционных сумм, после чего выбиралось лучшее значениеи происходило сравнение с рекордом, хранящимся в глобальной памяти.

Глубина развертки 8 была получена экспериментально, ее уменьшение ухудшает кеш-эффективность, а следовательно производительность. Увеличение же ведет к снижению количества запускаемых блоков из-за нехватки регистровой памяти. Естественно, что выбранные значения могут быть не оптимальны для других моделей GPU.

Фильтрация низкотекстурированных областей

Алгоритм StereoBlockMatching работает лучше всего на сценах, в которых объекты имеют текстуру, то есть содержат множество мелких деталей. Детализация и уникальность рисунка позволяет лучше сопоставлять пикселы левого и правого кадров. В областях же, где текстуры мало, мы можем видеть частые ошибки алгоритма. Так, например, проблемы возникают при определении расстояния до однородных стен (Рис. 6).

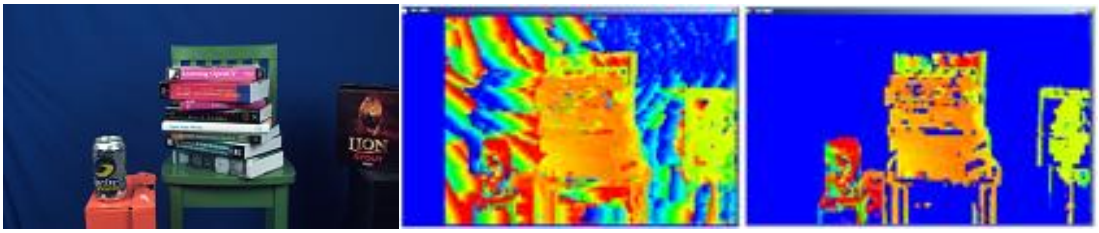


Рис. 6 Фильтрация областей со слабой текстурой.

Идея алгоритма фильтрации достаточно проста: для окрестности каждого пиксеоя изображения определяется мера ее текстурированности. Для этого к изображению применяется свертка с ядром:

$$\begin{matrix} -2 & 0 & 2 \\ -1 & 0 & 1 \\ -2 & 0 & 2 \end{matrix}$$

Данный оператор представляет собой производную по X. Затем происходит суммирование этих производных в окрестности каждого пикселя. После этого, если суммарное значение меньше, чем заданный порог, то область считается низкотекстурированной и величина смещения выставляется в «не определено». Для эффективного вычисления меры текстурированности, здесь используется идея бегущих сумм по Y.

Итоговая производительность

Финальная производительность алгоритма StereoBlockMatchingсоставила 18 кадров в секундуна видеокарте GeForce GTX470 (Fermi) и разрешении FullHD (1920x1080) для диапазона поиска смещения 0..256. Это приблизительно в 7 раз быстрее оптимизированной CPU версии алгоритма на четырехядерном IntelCore i5-750 2.66Ghz.На сегодняшний день это самая быстрая реализация алгоритма block-matchingиз известных авторам.

Заключение

Подводя итоги, еще раз отметим, чтоиспользование графических процессоров хорошо подходит для задач компьютерного зрения, позволяя значительно ускорить существующие алгоритмы и разрабатывать новые, ориентированные на массивно-параллельные архитектуры.

Для того, чтобы дать толчок разработкам в этой области, был начат настоящий проект по поддержке GPUв библиотеке OpenCV. Разработанный модуль достаточно прост в использовании, позволяет получать прирост производительности в 50-100 раз для примитивных функций, и в 5-10 раз для сложных высокоуровневых алгоритмов.

Важно отметить, что библиотека OpenCVраспространяется по свободной лицензии типа BSD, поэтому каждый может ознакомиться и воспользоваться результатами данной работы. Проект будет интенсивно развиваться, планируется расширение существующей базы примитивов и разработка новых высокоуровневых алгоритмов.

Ссылки

1. http://gpgpu.org
2. CUVI Lib, http://www.cuvilib.com
3. GPU4VISION, http://www.gpu4vision.org
4. GPUCV, https://picoforge.int-evry.fr/cgi-bin/twiki/view/Gpucv/Web
5. openVidia, http://openvidia.sourceforge.net/index.php/OpenVIDIA
6. NPP, http://developer.nvidia.com/object/npp_home.html
7. OpenCV,http://opencv.willowgarage.com/wiki
8. G.Bradski, A.Kaegler,*Computer Vision with the OpenCV library*.2008
9. P.Felzenszwalb, D.Huttenlocher,*Efficient Belief Propagation for Early Vision*. International Journal of Computer Vision, Vol. 70, No. 1, October 2006.
10. Q. Yang, L. Wang, and N. Ahuja, *A constant-space belief propagation algorithm for stereo matching.* CVPR 2010.
11. N.Dalal, B.Triggs, *Histograms of Oriented Gradients for Human Detection.*CVPR2005.
12. A.Obukhov,*Face detection with cuda*. GTC2009.
13. H. Bay, T.Tuytelaars, Luc Van Gool, *SURF: Speeded Up Robust Features.*
14. J.Stam (NVIDIA), D. Gallup and J. M. Frahm (UNC),*CUDA Stereo Imaging* http://openvidia.sourceforge.net/index.php/OpenVIDIA
15. D. Scharstein and R. Szeliski, *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms.*International Journal of Computer Vision, 47(1/2/3):7-42, April-June 2002.
16. J. Gibson, O. Marques, Stereo depth with a Unified Architecture GPU, CVPR2008 http://www.computer.org/portal/web/csdl/doi/10.1109/CVPRW.2008.4563092
17. I. Rosenberg, P. Davidson, C. Muller, J. Han (New York University), *Real-time stereo vision using semi-global matching on programmable graphics hardware*.SIGGRAPH2006
18. J. Woetzel , R. Koch, *Real-time Multi-stereo Depth Estimation on GPU with Approximative Discontinuity Handling*, 2004 http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=FE85B8890FFE28137C211F35343080DE?doi=10.1.1.1.3748&rep=rep1&type=pdf

Тезисы докладов

Программно-аппаратные комплексы на базе вычислительных систем с арифметическими ускорителями для моделирования методом Монте-Карло и методом молекулярной динамики

Воронин Б.Л., Грушин С.А., Житник А.К., Залялов А.Н., Копкин С.В., Крючков И.А., Малькин А.Г., Огнев С.П., Рослов В.И., Рыбкин А.С., Степаненко С.А., Шагалиев Р.М., Южаков В.В.

Российский федеральный ядерный центр — Всероссийский научно-исследовательский институт экспериментальной физики. Институт теоретической и математической физики (РФЯЦ-ВНИИЭФ ИТМФ)

Тезисы

Одним из современных направлений развития средств математического моделирования является применение вычислительных систем с арифметическими ускорителями, называемых также гибридными системами [1]. Вследствие конструктивных особенностей ускорителей эти системы, по сравнению с обычными универсальными микропроцессорами, позволяют на определенных классах задач существенно ускорить вычислительный процесс.

Такими задачами, в частности, являются задачи расчета нейтронно физических характеристик ядерных энергетических установок методом Монте-Карло [2] и задачи моделирования свойств ядерного топлива и конструкционных материалов реакторов, а также транспортных упаковочных комплектов методами молекулярной (наноуровень) и кластерной (микроуровень) динамики [3].

В этой работе представлены результаты исследования возможностей создания программно-аппаратных комплексов на базе вычислительных систем, интегрирующих процессоры архитектуры x86 и арифметические ускорители, предназначенных:

− для расчета методом Монте-Карло критических параметров активных зон ядерных реакторов, мест хранения ядерного топлива, транспортно-упаковочных комплектов для перевозки топлива и других средств обращения с топливом на АЭС и проектирования технических объектов атомной энергетики;

− для моделирования свойств материалов методом молекулярной динамики при решении специальных классов задач атомной энергетики и других наукоемких отраслей промышленности.

В работе приведены структура и параметры вычислительных систем с арифметическими ускорителями, особенности реализации аппаратно-программных комплексов.

Экономическая эффективность результатов подтверждается не только техническими параметрами (производительность, мощность), конкурентно способной ценой создаваемых вычислительных систем, не требующих специально обустроенных систем жизнеобеспечения, но и созданием комплексных программно-аппаратных средств моделирования сложных физических процессов.

Литература

1. Sim L.C., Schroder H., Leedham G. MIMD-SIMD hybrid system – towards a new low cost parallel system // Parallel Computing . 2003. Vol. 29. P. 21-36.
2. Zhitnik A.K., Tarasov V.A., Ognev S.P. , Taiwo T.A., Yang W.S. Code TDMCC for Monte Carlo Computations of Spatial Reactor Cores Kinetics // Monte Carlo 2005 Topical Meeting. Chattanooga, Tennesse, April 17 – 21, 2005.
3. Воронин Б.Л., Ерофеев А.М., Копкин С.В., Крючков И.А., Рыбкин А.С., Степаненко С.А., Южаков В.В. Применение графических арифметических ускорителей для расчета задач молекулярной динамики по программному комплексу МД // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2009. Вып. 2. С. 55-61.

Использование NVIDIA PhysX для моделирования взаимодействия корабля со льдом в реальном времени.

Метрикин И. А., Løset Sveinung
Marine Civil Engineering Group, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Аннотация.

В настоящее время наблюдается существенный всплеск интереса ведущих мировых держав к Арктике. В первую очередь это связано с тем, что до 30% мировых запасов природного газа и до 13% мировых запасов нефти сосредоточены в пределах Северного полярного круга [1, 2]. Большинство арктических месторождений расположены в акваториях Баренцева и Карского морей, а также в море Бофорта. Основной преградой для освоения нефтегазовых месторождений в Арктике являются тяжелейшие погодные условия: экстремально низкие температуры, полярная ночь, сильнейшие ветра, и самое главное – морской лёд. Именно ледовые нагрузки составляют до 90% общих нагрузок на судно, оперирующее в Арктике [3].

На ранних этапах проектирования новых ледоколов и арктических танкеров, для оценки ледовых нагрузок широко используется численное моделирование. Это связано с исключительной дороговизной лабораторных и тем более полномасштабных испытаний [4]. Однако, процесс взаимодействия корабля со льдом является настолько сложным, что производительность большинства численных моделей далека от рельного времени. В численной модели необходимо реализовать разрушение льдин при их взаимодействиях друг с другом и с корпусом корабля, притапливание сломанных льдин, трение объектов и взаимные соударения. Вычислительные затраты большинства современных моделей взаимодействия корабля со льдом существенно превосходят мощность вычислительных систем и для выполнения расчетов требуются дни, а иногда недели [5].

В данном докладе приводится новый метод численного моделирования взаимодействия корабля со льдом, основывающийся на использовании технологии NVIDIA PhysX. Основное преимущество данного метода – существенное повышение вычислительной эффективности по отношению к предшественникам, позволившее реализовать симуляции в реальном времени. В докладе освещены физические механизмы, учитываемые в математической модели и способ их реализации с помощью NVIDIA PhysX. Приведены результаты численного моделирования и их сравнение с лабораторными и полномасштабными испытаниями. Выделены основные достоинства и недостатки NVIDIA PhysX как платформы для реализации крупномасштабных симуляторов физических систем.

Список литературы.

1. Основы государственной политики Российской Федерации в Арктике на период до 2020 года и дальнейшую перспективу. Документ, утвержденный Президентом Российской Федерации Дмитрием Медведевым 18 сентября 2008 г. (Пр - 1969).
2. D. L. Gautier, K. J. Bird, R. R. Charpentier, A. Grantz, D. W. Houseknecht, T. R. Klett, T. E. Moore, J. K. Pitman, C. J. Schenk, J. H. Schuenemeyer, K. Sørensen, M. E. Tennyson, Z. C. Valin, C. J. Wandrey. Assessment of undiscovered oil and gas in the arctic. Science, 324:1175–1179, 2009.
3. P. Valanto. On the cause and distribution of resistance forces on ship hulls moving in level ice, Proceedings of POAC 2001 (2001), pp. 803–816.
4. B. Su, K. Riska and T. Moan, A numerical method for the prediction of ship performance in level ice, Cold Regions Science and Technology 60 (2010), pp. 177–188.
5. Lubbad, R., Løset, S., A numerical model for real-time simulation of ship–ice interaction, Cold Reg. Sci. Technol. (2010), doi:10.1016/j.coldregions.2010.09.004.

Стендовые доклады

Адаптация геостатистического симулятора к современным гибридным вычислительным системам на основе графических процессоров NVIDIA

М.В. Андреев, Р.К. Газизов, А.Л. Штангеев, А.В. Юлдашев, А.А. Яковлев
Уфимский государственный авиационный технический университет

В настоящем докладе представлены новые результаты по адаптации геостатистического симулятора, реализующего построение обусловленных геостохастических геологических моделей по скважинным данным с учетом анизотропности и нестационарности пластовых свойств, для эффективного выполнения на гибридных вычислительных системах на основе графических процессоров (GPU) компании NVIDIA.

Нами была разработана параллельная версия геостатистического симулятора [1], способная выполняться на гибридных вычислительных системах с общей памятью и графическими процессорами NVIDIA с поддержкой CUDA. Распараллеливание в модели общей памяти реализовано средствами интерфейса OpenMP, а с помощью технологии PGI Accelerator [2] произведена адаптация для выполнения на GPU наиболее трудоемких участков кода программы, выделенных в виде функций, решающих следующие вычислительные задачи: построение матрицы ковариации, выполнение быстрого преобразования Фурье, а также обусловленного стохастического моделирования.

Было проведено тестирование производительности полученной версии симулятора при расчете реальной модели на двухпроцессорной рабочей станции FSC CELSIUS V840 на базе процессоров AMD Opteron 2214 с различными графическими ускорителями NVIDIA: GeForce 295, 460, 470, 480 GTX и Tesla C1060. Отслеживалось ускорение суммарного времени выполнения адаптированных участков кода при расчете на CPU+GPU относительно суммарного времени их выполнения при расчете на 4 ядрах центральных процессоров (CPU). В результате максимальное ускорение было достигнуто при использовании GeForce 480 GTX и составило более 8 раз.

Далее возникла задача адаптации симулятора к более современным гибридным вычислительным системам на основе графических процессоров NVIDIA. Благодаря предоставленному нам доступу к вычислительным ресурсам Межведомственного суперкомпьютерного центра РАН, были проведены работы по тестированию производительности и оптимизации симулятора на высокопроизводительном вычислительном сервере Kraftway Science KT25. Данный сервер имеет в своем составе два современных шестиядерных процессора Intel Xeon X5660, а также четыре профессиональных графических ускорителя NVIDIA Tesla M2050 последнего поколения, что позволяет говорить о его пиковой производительности более двух терафлопс на вычислениях двойной точности. Необходимо отметить несомненное достоинство установленных в сервере ускорителей Tesla серии 20: большой объем видеопамяти с поддержкой технологии коррекции ошибок ECC.

Первые тесты, проведенные на сервере Kraftway, выявили ряд особенностей. Во-первых, при тестировании на моделях, время расчета которых достаточно мало, было отмечено ощутимое (около 4-10 секунд) время инициализации при первом обращении к GPU в программе. В связи с этим инициализация в программе была вынесена отдельно от адаптированных функций и далее при замерах времени выполнения не учитывалась. Во-вторых, ускорение при выполнении на CPU+GPU относительно CPU оказалось существенно более низким, чем на ранее рассмотренной рабочей станции CELSIUS V840. Одной из причин снижения ускорения, безусловно, явилось использование более производительных процессоров Xeon X5660. Кроме того, сравнительный анализ характеристик карт GeForce 480 GTX и Tesla M2050 с сайта производителя, показал, что первая имеет в своем составе большее

количество ядер и ее аппаратные блоки работают на более высоких частотах, что приводит к более высокой пиковой производительности GeForce 480 GTX в сравнении с Tesla M2050.

Профилирование процесса выполнения симулятора на различных графических процессорах, проведенное средствами NVIDIA Visual Profiler из CUDA Toolkit, показало, что при выполнении некоторых адаптированных функций, ресурсы GPU используются не более чем на десять процентов. Это и послужило толчком для поиска путей оптимизации программного кода в целях более эффективного использования графических процессоров.

Оказалось, что в ряде рассмотренных функций имел место неэффективный порядок доступа к данным в памяти GPU. Например, в функции обусловленного стохастического моделирования было замечено, что при выборке элементов трехмерного массива данных во вложенном цикле, нити, выполняющиеся на графическом процессоре в рамках одного блока, осуществляют выборку разнесенных в памяти элементов массива. Проблема была устранена за счет перегруппировки данных в массиве и изменения порядка обращения к элементам массива, что позволило сократить время выполнения данного участка кода более чем в три раза.

Ниже приведены результаты тестирования симулятора после проведенных оптимизаций при расчете реальной модели. Отметим, что сборка симулятора для CPU осуществлялась компилятором Intel C++ Compiler, а для CPU+GPU – компиляторами PGI C/C++.

В таблице 1 представлены суммарные времена выполнения адаптированных функций в зависимости от числа потоков, порождаемых в OpenMP-программе при выполнении на CPU, а также CPU+GPU. Причем при запуске на CPU число потоков варьировалось от 1 до 12 в соответствии с числом процессорных ядер в системе, а при запуске на CPU+GPU от 1 до 4 для того, чтобы каждый поток, выполняющийся на одном из процессорных ядер, имел возможность монопольно использовать один из четырех графических ускорителей.

Таблица 1. Время выполнения адаптированных функций на CPU и CPU+GPU.

	Время выполнения, с				
Платформа/Количество потоков OpenMP	1	2	4	8	12
CPU	1 285,19	644,11	324,82	170,75	112,49
CPU+GPU	48,30	24,54	12,39	-	-

Для наглядности на рисунке 1 представлены зависимости ускорения расчета на CPU, а также CPU+GPU в зависимости от числа OpenMP-потоков по сравнению со временем расчета одним потоков на одном ядре центрального процессора. Несложно увидеть, что время расчета адаптированных функций снижается линейно с увеличением количества потоков. В случае одного потока, подключение к расчету графического ускорителя позволяет сократить время в 26 раз. Кроме того сравнение минимальных времен выполнения адаптированных функций на CPU и CPU+GPU показывает, что за счет использования графических процессоров минимальное время выполнения (расчет на всех 12 процессорных ядрах) может быть сокращено более чем в 9 раз (расчет на 4 процессорных ядрах при поддержке 4 графических ускорителей).

Таким образом, применение современных гибридных вычислительных систем на основе графических процессоров позволяет значительно ускорить процесс построения обусловленных геостохастических геологических моделей.

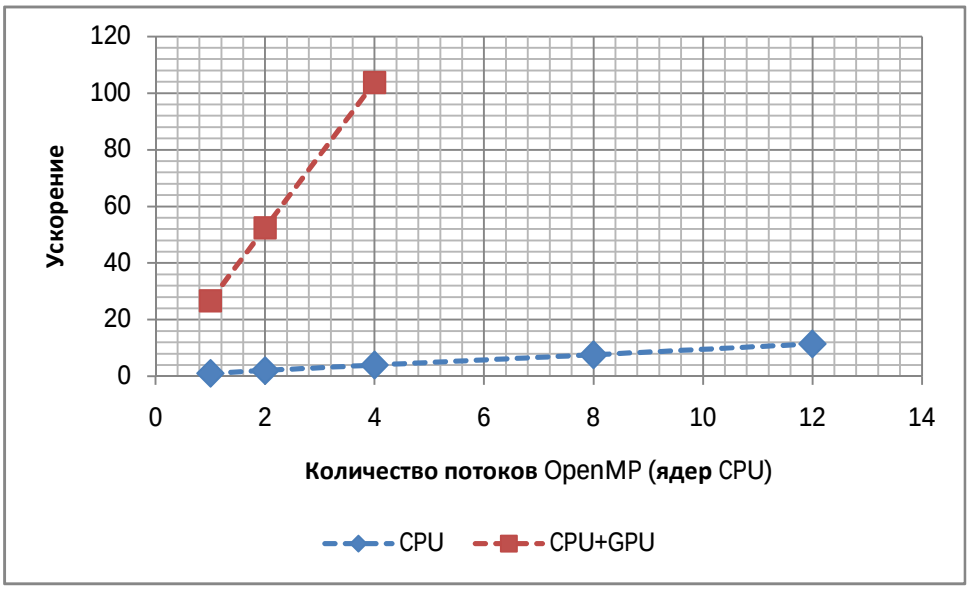


Рис. 1. Зависимости ускорения от количества потоков OpenMP при использовании CPU и CPU+GPU относительно времени расчета одним потоком на CPU

В дальнейшем планируется рассмотреть возможность перехода на вычисления с одинарной точностью в целях дальнейшего ускорения времени расчетов. Кроме того планируется оценить эффективность реализации алгоритмов решения задач геостатистического моделирования с помощью высокоуровневой технологии PGI Accelerator в сравнении с наиболее распространенной технологией программирования графических процессоров – CUDA.

Литература

1. Андреев М.В., Галеев Э.Р., Штангеев А.Л., Юлдашев А.В., Яковлев А.А. Опыт портирования геостатистического симулятора на архитектуру GPU средствами технологии PGI Accelerator // Высокопроизводительные параллельные вычисления на кластерных системах (HPC-2010): Материалы X международной конференции (Пермь, 1 – 3 ноября 2010 г.), в двух томах – Пермь: Издательство ПГТУ, 2010. – Том 1. С. 37-41.
2. PGI Fortran & C Accelerator Programming Model white paper. URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf (дата обращения: 11.02.2011).

Исследование ускорения решения термо-структурной задачи в пакете ANSYS средствами GPU

Р.К. Газизов, А.А. Касаткин, А.В. Юлдашев
Уфимский государственный авиационный технический университет

В настоящее время расширяется применение гибридных вычислительных систем на основе графических процессоров (GPU) для решения различных научных и прикладных вычислительных задач. Однако, пока еще немногие инженерные программные пакеты конечно-элементного анализа поддерживают использование GPU для ускорения проводимых вычислений. Недавно такая поддержка появилась в широко распространенном коммерческом программном комплексе ANSYS в составе модуля термо-прочностных расчетов ANSYS Mechanical.

В новой версии 13.0 модулем ANSYS Mechanical в многопоточной (SMP) версии поддерживаются расчеты на профессиональных графических ускорителях NVIDIA Tesla при использовании различных методов решения возникающих систем линейных уравнений: прямого SPARSE (на основе LU-разложения) и итерационных PCG и JCG (методы сопряженных градиентов с различными предобуславливателями) [1]. Одним из существенных ограничений в текущей версии является поддержка не более одного GPU. Однако, в последующих версиях планируется поддержать в SMP-версии расчеты на нескольких GPU, а также ввести поддержку GPU при вычислениях на многопроцессорных системах с распределенной памятью в составе Distributed ANSYS.

В нашей работе проведено исследование возможностей ANSYS Mechanical по ускорению расчетов средствами GPU при решении термо-структурной задачи, типичной для моделирования процесса линейной сварки трением (ЛСТ) [2]. Данная технология применяется, в частности, при производстве блисков для авиационных двигателей.

Сварка трением – это разновидность сварки давлением, при которой нагрев осуществляется трением, вызванным перемещением друг относительно друга соединяемых частей свариваемого изделия. Процесс ЛСТ осуществляется возвратно-поступательным движением частей, подлежащих свариванию, с частотой порядка 60 Гц и амплитудой до 3-х мм, сжимаемых с большим прижимным усилием для образования плотного контакта.

Большие напряжения в области контакта приводят к интенсивному тепловыделению и резким перепадам температур, вследствие чего при моделировании требуется мелкая неравномерная сетка. Кроме того, быстротечность процесса обуславливает необходимость выбора маленького ($10^{-4}...10^{-5}$ сек.) шага по времени для сходимости расчетных методов. Также задача усложняется существенной зависимостью механических и теплофизических свойств материала от температуры. В связи с этим компьютерное моделирование процесса ЛСТ является вычислительно трудоемкой задачей.

Исследование возможностей ANSYS Mechanical по ускорению решения данной задачи средствами GPU было проведено на двухпроцессорной рабочей станции FSC V840 (процессоры Opteron 2214 Dual-Core, 2.2 GHz, объем оперативной памяти 8 GB, PC2-5300 FB-DIMM ECC) с графическим ускорителем NVIDIA Tesla C1060.

В рамках исследования решалась термо-структурная задача (трение брусков с разогревом) с различным числом 20-узловых конечных элементов (SOLID226): 3200, 9600 и 19200 элементов. В целях сокращения времени тестирования расчет проводился на малом количестве временных шагов: 10 шагов для моделей из 3200 и 9600 элементов, 5 шагов для модели из 19200 элементов. Расчет осуществлялся в SMP-версии с использованием различных методов, поддерживающих вычисления на GPU: SPARSE, PCG и JCG. Ниже представлены результаты тестирования отдельно для каждого метода. В таблицах 1-3 приведены времена выполнения, а на рисунках 1-3 зависимости ускорений времени расчета от числа

задействованных процессорных ядер для запусков с включенной поддержкой GPU (CPU+GPU) и без нее (CPU) относительно времени расчета на одном процессорном ядре без поддержки GPU. Отметим, что все вычисления проводились с двойной точностью.

Таблица 1. Время расчета при использовании метода SPARSE с поддержкой GPU и без.

SPARSE	Время расчета, с.					
Число ядер / Количество элементов в модели, поддержка GPU	3200	3200, GPU	9600	9600, GPU	19200	19200, GPU
1	2618	1117	12936	3803	17111	4420
2	1822	1080	8427	3518	10837	4291
3	1525	1033	6637	3492	8606	4170
4	1372	1024	5781	3483	7535	4094

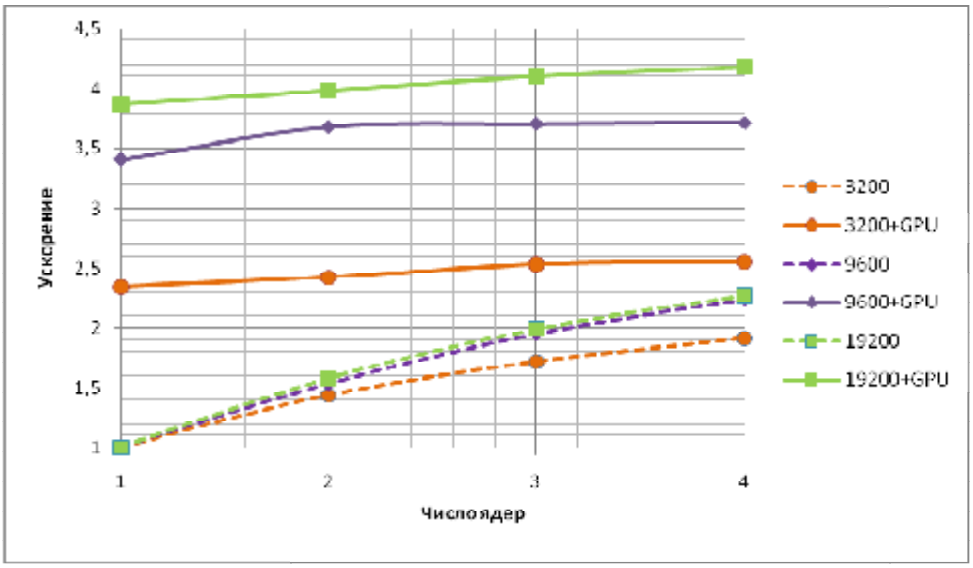


Рис. 1. Зависимости ускорения времени расчета от числа ядер CPU при использовании метода SPARSE относительно времени расчета на одном ядре без поддержки GPU

Из рисунка 1 видно, что при использовании прямого метода SPARSE SMP-версия ускоряется в 1,9-2,2 раза на 4 ядрах CPU в зависимости от размера модели. Заметно ускорение, достигнутое за счет поддержки GPU: при запуске на 4 процессорных ядрах в зависимости от размера модели время расчета при подключении GPU сокращается на 25-45%.

Таблица 2. Время расчета при использовании метода JCG с поддержкой GPU и без.

JCG	Время расчета, с.					
Число ядер / Количество элементов в модели, поддержка GPU	3200	3200, GPU	9600	9600, GPU	19200	19200, GPU
1	1701	851	9169	3380	15024	4537
2	1304	752	6547	3190	10582	4424
3	1180	755	5713	2967	8991	4327
4	1259	798	5580	2938	9411	4349

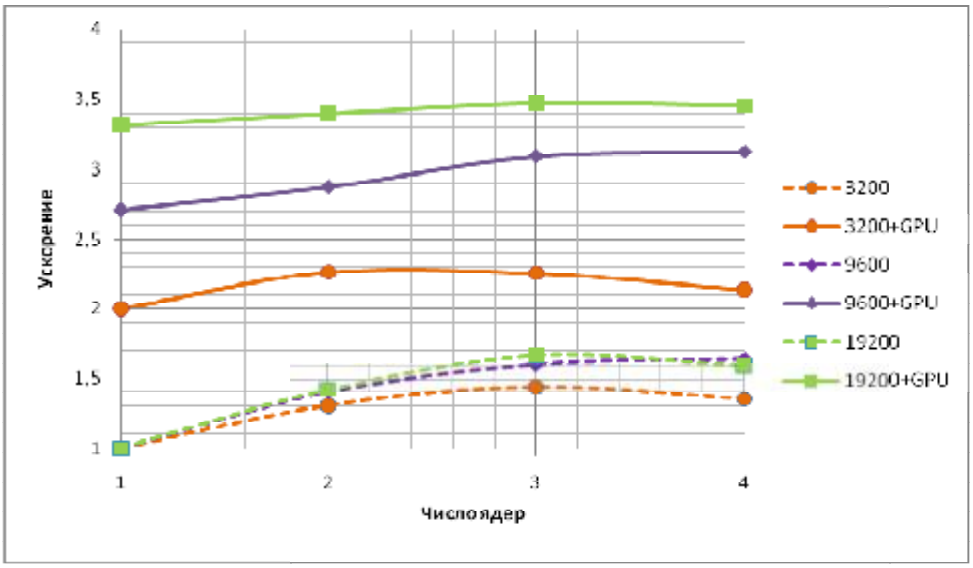


Рис. 2. Зависимости ускорения времени расчета от числа ядер CPU при использовании метода JCG относительно времени расчета на одном ядре без поддержки GPU

При использовании итерационного метода JCG (метод сопряженных градиентов с предобуславливателем Якоби) SMP-версия ускоряется несколько хуже, чем в случае предыдущего метода (рис. 2): в 1,4-1,6 раз на 3-4 ядрах CPU в зависимости от размера модели. Причем, при расчете модели с числом элементов равным 3200 и 19200 наблюдается замедление расчета на 4 ядрах. В тоже время, для метода JCG отмечается больший выигрыш от использования GPU. Минимальное время выполнения при поддержке GPU в зависимости от размера модели меньше на 36-51%, чем в случае оптимального запуска на ядрах CPU.

Таблица 3. Время расчета при использовании метода PCG с поддержкой GPU и без.

PCG	Время расчета, с.					
Число ядер / Количество элементов в модели, поддержка GPU	3200	3200, GPU	9600	9600, GPU	19200	19200, GPU
1	1434	1143	4444	3255	6091	3602
2	1239	964	3773	2995	4453	3452
3	1169	998	3543	2853	4120	3435
4	1234	995	3524	2931	4289	3514

Итерационный метод PCG (метод сопряженных градиентов со специализированным предобуславливателем), показывает наименьшее время выполнения при расчете рассматриваемой термо-структурной задачи на CPU по сравнению с методами SPARSE и JCG. Однако, при использовании PCG ускорение SMP-версии заметно ниже, чем в случае двух ранее рассмотренных методов (рис. 3), и составляет 1,2-1,4 раза на 3-4 ядрах CPU в зависимости от размера модели. Кроме того, снижение времени выполнения при расчете с поддержкой GPU составляет не более 19% относительно оптимального запуска на CPU. Таким образом, SMP-версия с методом PCG, как правило, позволяет получить решение нашей задачи за минимальное время, но при этом не получает большого преимущества от использования нескольких процессорных ядер, а также GPU.

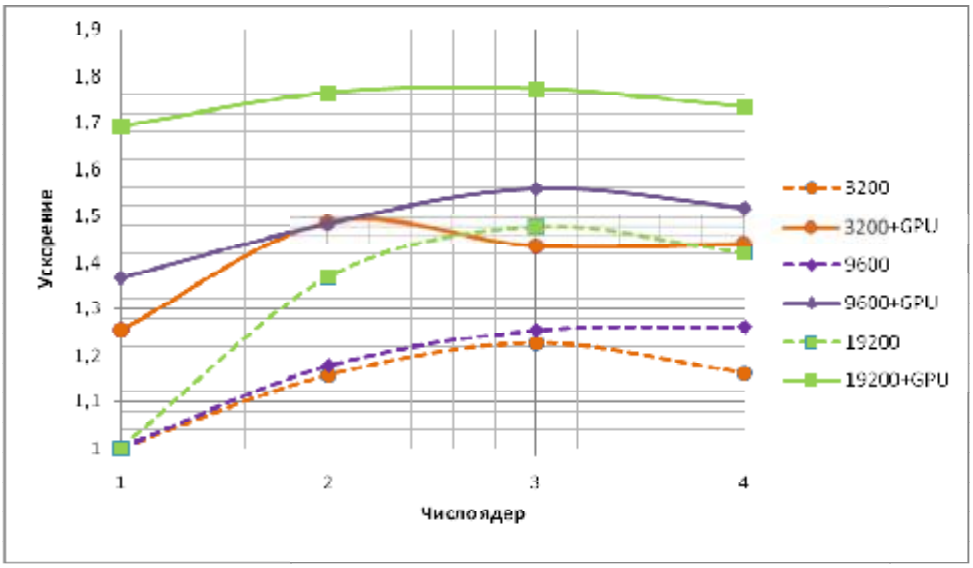


Рис. 3. Зависимости ускорения времени расчета от числа ядер CPU при использовании метода PCG относительно времени расчета на одном ядре без поддержки GPU

Отметим, что положительный эффект при расчете с включенной поддержкой GPU преимущественно возрастает при увеличении количества элементов в модели независимо от используемого метода.

Проведенные эксперименты позволили исследовать возможности ANSYS Mechanical по ускорению расчетов средствами GPU при решении термо-структурной задачи, типичной для моделирования процесса линейной сварки трением. Полученное ускорение оказалось не столь высоким, что, скорее всего, связано с использованием графического процессора предыдущего поколения. Тем не менее, при включенной поддержке GPU достигается стабильное снижение времени расчета на 19-51%. В связи с этим представляет интерес проведение аналогичного тестирования с графическим процессором Tesla серии 20.

Благодарности

Работа выполнена в рамках проекта «Создание технологий и промышленного производства узлов и лопаток ГТД с облегченными высокопрочными конструкциями для авиационных двигателей новых поколений» (шифр 2010-218-01-133) в рамках реализации постановления № 218 от 9.04.2010 г. «О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства».

Литература

1. Speed Up Simulations with a GPU // ANSYS Advantage Magazine. URL: http://www.ansys.com/staticassets/ANSYS/staticassets/resourcelibrary/article/AA-V4-I2-Speed-Up-Simulation-with-GPU.pdf (дата обращения: 11.02.2011).
2. Газизов Р.К., Иванов В.Ю., Касаткин А.А.,Лукащук С.Ю., Насибуллаев И.Ш., Ямилева А.М. Моделирование процесса линейной сварки трением в пакетах компьютерного моделирования – Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.) – Челябинск: Изд. ЮУрГУ, 2010. – С. 442-447.

Программный комплекс моделирования методом молекулярной динамики для гибридных вычислительных систем

И.А. Крючков, С.В. Копкин
ФГУП «РФЯЦ-ВНИИЭФ», г. Саров Нижегородской обл.

Введение

В проведенных ранее работах [1,2,3] получены результаты по адаптации различных частей комплекса МД для расчетов на гибридных вычислительных системах.

Проведение молекулярно-динамического моделирования процессов определяющих свойства конструкционных материалов является очень трудоемким и требует значительных вычислительных ресурсов. Часть расчетов по комплексу МД уже ведется на гибридных вычислительных системах.

В настоящей работе приведено обобщение по основным видам взаимодействий частиц, показаны особенности данного подхода.

Реализованы дополнительные возможности в ускорительной версии комплекса МД, а также исследуется возможность применения различных гибридных вычислительных систем для решения задач молекулярной динамики.

В частности:

- выполнена модернизация функций, реализующих межчастичное взаимодействие;
- реализована возможность расчета взаимодействия для нескольких материалов;
- получены длительности выполнения на однопроцессорной персональной системе, специализированной компактной вычислительной системе ГВС-10 «Кубань», гибридной мультипроцессорной системе;
- выполнено сравнение с универсальными составляющими приведенных систем;
- проведено предварительное сравнение ускорителей на основе графических процессоров NVIDIA GT200 и GF100 на задачах молекулярной динамики.

Модернизация программы

Табулирование значений потенциалов и их производных

В предыдущих работах были представлены результаты разработки ускорительной версии программы расчета сил взаимодействия для парных, многочастичных EAM и MEAM потенциалов в комплексе МД. Реализованные алгоритмы были узкоспециализированными, расчеты проводились для конкретных типов потенциалов: парный - Морзе, многочастичный EAM - Ackland et al. и MEAM по модели Baskes et al. Для вычислений с другими типами потенциалов программный код необходимо модифицировать, т.к. используется аналитический вид функций потенциала. При вводе новых модельных функций меняется арифметическая интенсивность программы, что требует проведение дополнительных исследований и настройки программы для оптимальной работы на арифметических ускорителях.

В комплексе МД используется метод табулирования значений потенциала и его производной, а при вычислении значений функции потенциала или его производной, конкретное значение получается интерполяцией табличных значений. Поэтому данный алгоритм адаптирован для работы на арифметических ускорителях. Это позволяет унифицировать работу программы для потенциалов различных видов и использовать библиотеку межчастичных потенциалов комплекса МД для расчетов на арифметических ускорителях.

При работе комплекса на гибридных вычислительных системах на универсальной части вызывается программа подготовки начальных данных, которая заполняет массивы данных параметрами задачи, строит начальную геометрию, вычисляет по аналитическим функциям значения потенциалов и их производных и заполняет массивы таблиц. Далее таблицы значений копируются в память ускорителей и используются во время работы молекулярно-

динамического решателя.

Программы молекулярно-динамического решателя реализованные для выполнения на ускорительном сегменте переработаны в соответствии с использованием табличных значений потенциалов и их производных. Добавлены ускорительные функции интерполяции парных и многочастичных потенциалов. Переработана структура программы вычисления сил — для парных и многочастичных потенциалов EAM разработан алгоритм распараллеливания на арифметических ускорителях по локальному списку частиц. Каждой частице ставится в соответствие номер ячейки в которой она находится и каждый поток ускорителя при обработке своей частицы производит перебор всех частиц в соответствующей ячейке, начиная с первой и по всем частицам из соседних ячеек. Это позволило отказаться от дорогостоящего списка соседей и использовать уже существующие в комплексе МД векторные массивы - цепочки связанных частиц.

Использование нескольких материалов

Первые версии программ комплекса МД для гибридных вычислительных систем обладали ограниченными возможностями, т.к. были рассчитаны на использование только одного типа материала и соответственно одного потенциала. В комплексе МД для универсальных вычислительных систем существует возможность проведения расчетов с несколькими типами материалов одновременно и соответственно несколькими потенциалами. Разработка ускорительной версии программы, использующей табулированные значения потенциалов и их производных, позволила адаптировать её для использования нескольких типов материалов в задаче и соответственно нескольких потенциалов.

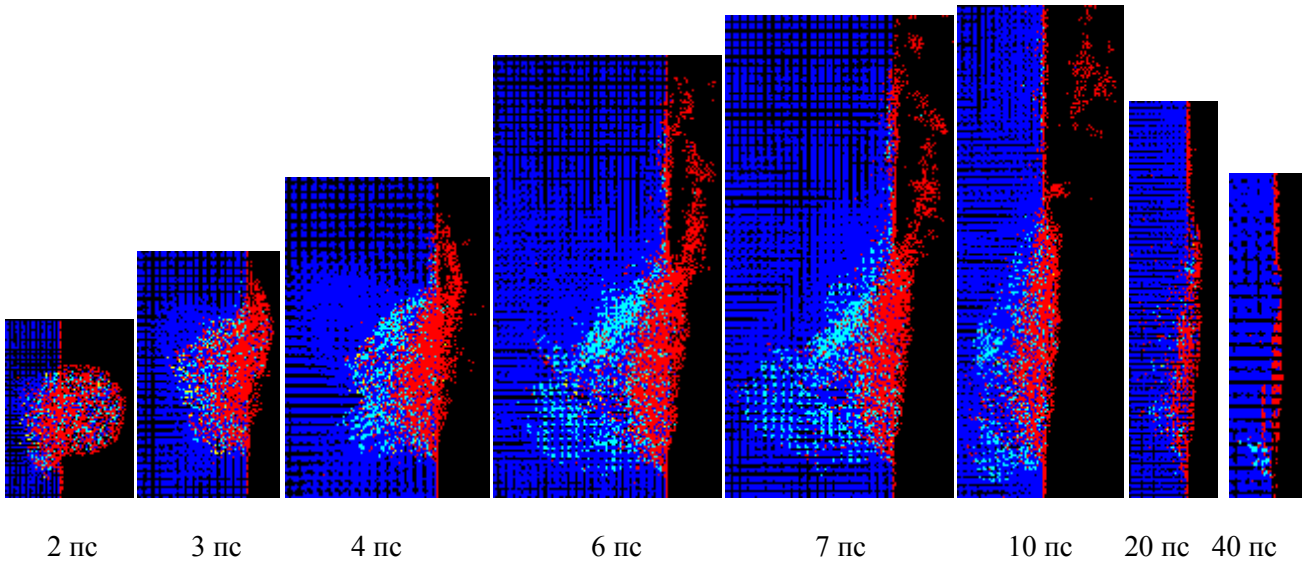


Рисунок 1- Процесс взаимодействия капли с [100] мишенью при косом ударе 60° к нормали, скорость капли 5 км/с. Под картинками время в пс. Синим цветом помечены атомы, принадлежащие ОЦК решетке, циан соответствует ГПУ, желтый – ГЦК, красным цветом помечены атомы, принадлежащие к дефектным структурам.

При использовании двух материалов в задаче для взаимодействия частиц одинакового и разного типа соответственно задается три потенциала. При этом задается таблица указателей на номер потенциала.

В программе подготовки начальных данных строится несколько таблиц по числу используемых потенциалов. Все таблицы копируются в память ускорителя и используются во время работы молекулярно-динамического решателя, реализованного для использования арифметических ускорителей. Добавлены ускорительные функции для интерполирования значений потенциалов и их производных с учетом типа взаимодействующих частиц.

Введенные изменения позволяют моделировать взаимодействие расплавленных металлически наночастиц с поверхностями ОЦК и ГЦК металлов[4]. Проведен расчет взаимодействия расплавленной капли меди (Cu) с поверхностью тантал (Ta) ОЦК (рисунок 1).

Потенциал MEAM для сплава Pu-Ga

Программа расчета взаимодействия частиц для потенциала MEAM отличается от аналогичных программ для парных и многочастичных EAM потенциалов. В ней используется аналитический вид функций. Для сплавов используется несколько наборов параметров, причем для взаимодействия пары однотипных частиц применяется полный набор параметров, а для взаимодействия частиц разного типа используется набор параметров отвечающих за функции расчета экранировки и локальных электронных плотностей.

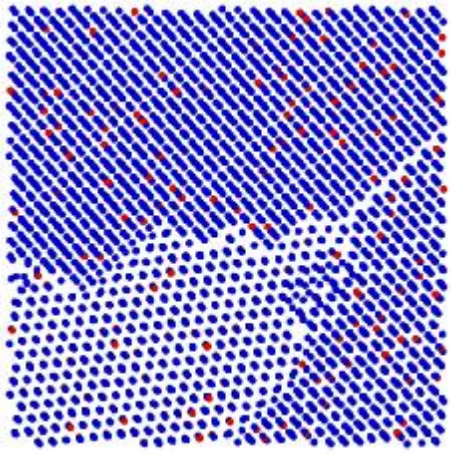


Рисунок 2 – Поликристаллический Pu-Ga (содержание Ga 5.2 %). Размер образца 20x20x20, 10 зерен. Синим цветом помечены атомы Pu, красным – Ga

Все ускорительные функции MEAM потенциала были доработаны для использования частиц двух типов. В программу подготовки начальных данных добавлено копирование всех наборов параметров потенциалов, в программы расчета взаимодействия частиц добавлена ускорительная функция выбора типа взаимодействия и соответственно набора параметров. При этом набор ускорительных аналитических функций претерпел лишь небольшие изменения, связанные с добавлением нескольких дополнительных параметров – множителей.

Все изменения в алгоритмах и программах расчета взаимодействия с потенциалом MEAM позволяют проводить моделирование свойств и процессов в конструкционном материале – сплав Pu-Ga [3].

Результаты экспериментов

Исследования, представленные в отчете проведены на различных вычислительных системах:

- персональная вычислительная система с АрУ NVIDIA GTX260 и NVIDIA GTX470;
- специализированная компактная вычислительная система ГВС-10 «Кубань»;
- мультипроцессорная гибридная вычислительная система.

На каждой из перечисленных вычислительных систем выполнен расчет по комплексу МД в различных постановках (потенциалы Морзе, EAM, MEAM).

При проведении экспериментальных исследований задействованы различные арифметические ускорители:

- NVIDIA GTX260;
- NVIDIA GTX295;

– NVIDIA Tesla C1060;
– NVIDIA GTX470.

Спецификации АрУ приведены в таблице 1.

Таблица 1 - Спецификация арифметических ускорителей

Характеристики	**NVIDIA GTX260**	**NVIDIA GTX295**	**NVIDIA Tesla C1060**	**NVIDIA GTX470**
Тип GPU	GT200	GT200	GT200	GF100
Частота ядра, ГГц	1,296	1,24	1,3	1,22
Потоковых процессоров, шт	192	480	240	448
Мультипроцессоров, шт	24	60	30	14
Объем глобальной памяти, Мбайт	896	1792	4096	1280
Интерфейс	PCI-Express x16	PCI-Express x16	PCI-Express x16	PCI-Express x16

Для обобщения результаты экспериментов на различных арифметических ускорителях приведены на рисунках 3-5. Также на рисунках 6-8 даны ускорения, полученные при максимальном задействовании используемых вычислительных систем.

На рисунках 3-5 представлены длительности выполнения вычислений полученные на различных ускорителях относительно одного ядра универсального процессора Intel Core i7 920 при использовании 1 MPI-процесса на задачах различных размеров для различных потенциалов.

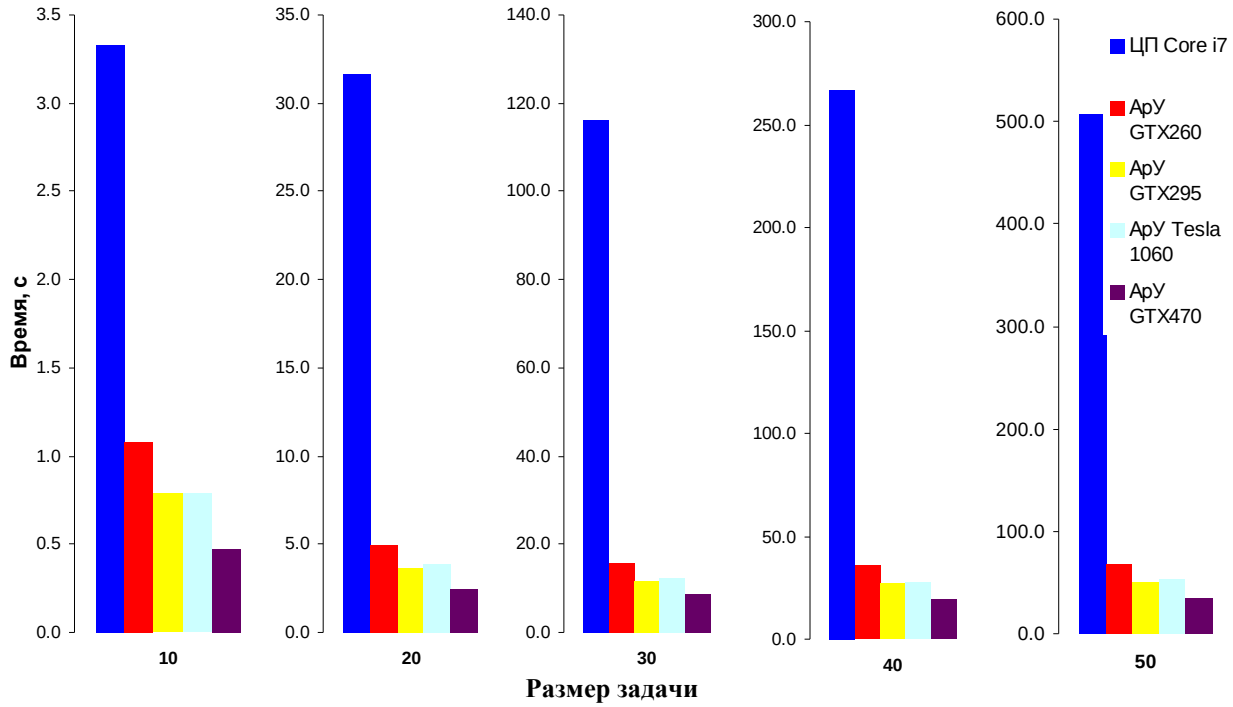


Рисунок 3 – Длительности вычислений по программе МД (потенциал Морзе, 1 MPI-процесс)

Ускорение, полученное на одном АрУ, составило от 3 до 14,5 раз по сравнению с одним ядром универсального процессора. Наибольшее ускорение получено на АрУ нового поколения NVIDIA GTX470.

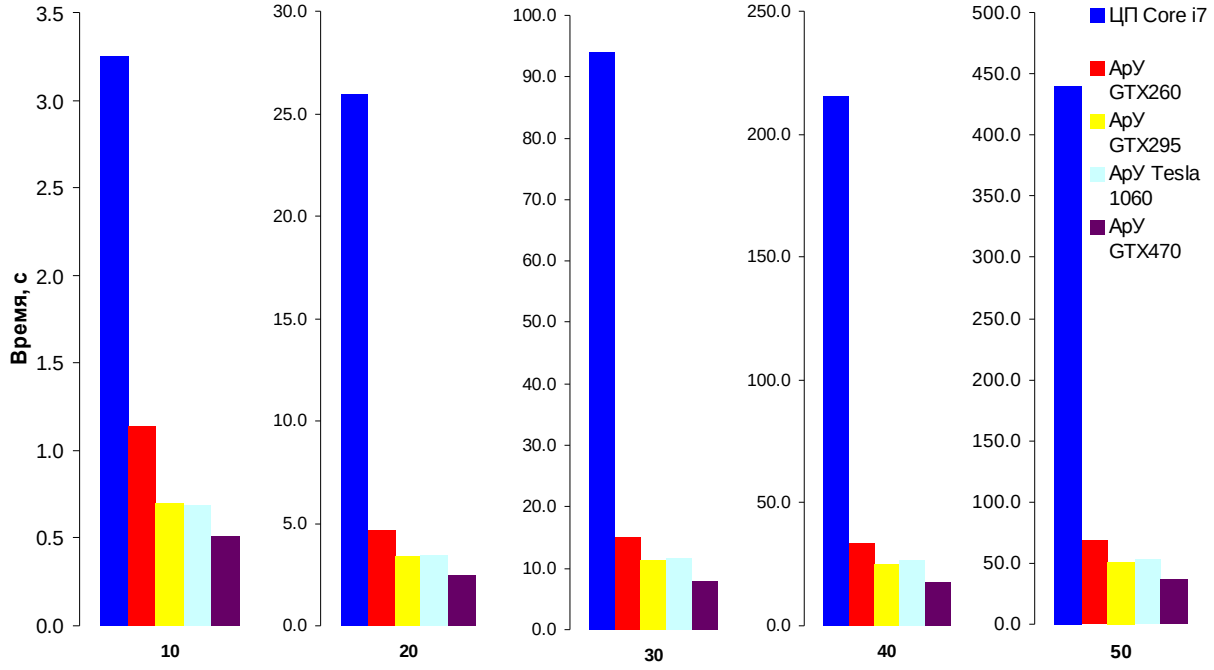


Рисунок 4 – Длительности вычислений по программе МД (потенциал EAM, 1 MPI-процесс)

Ускорение, полученное на одном АрУ, составило от 2,8 до 12,1 раз по сравнению с одним ядром универсального процессора. Наибольшее ускорение получено на АрУ нового поколения NVIDIA GTX470 (от 1,45 до 1,9 раз больше чем другие АрУ).

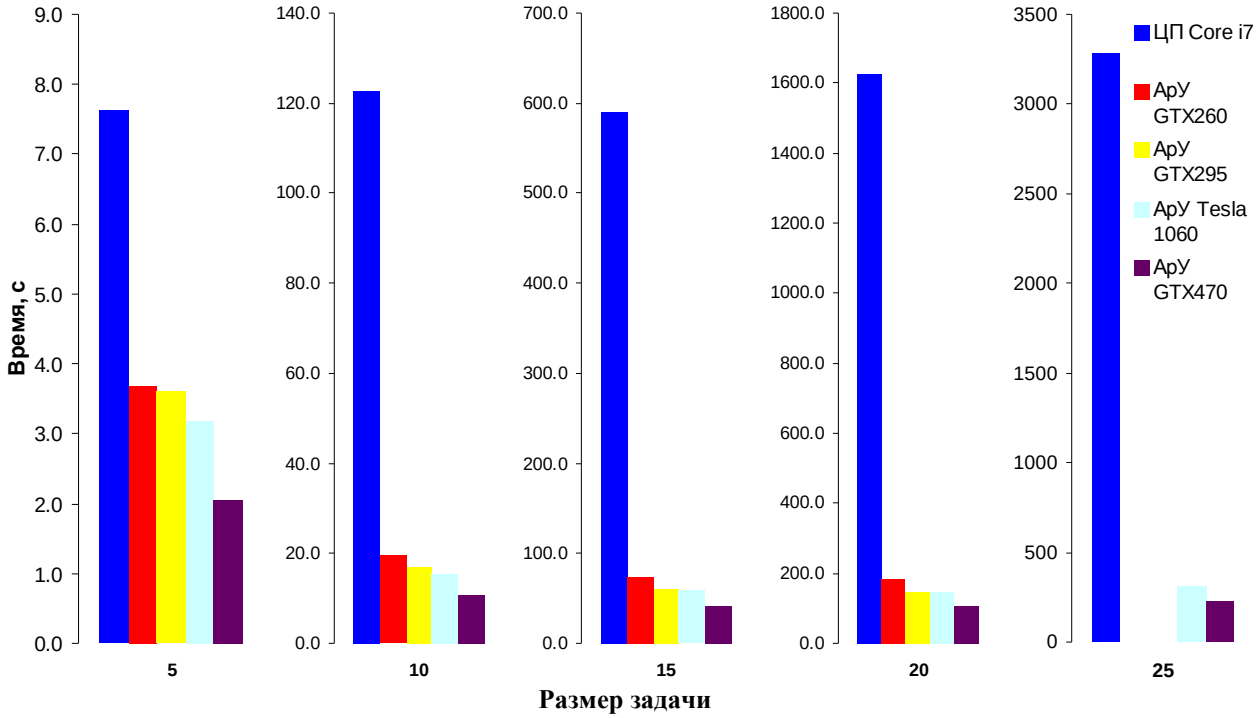


Рисунок 5 – Длительности вычислений по программе МД (потенциал MEAM, 1 MPI-процесс)

Ускорение, полученное на одном АрУ, составило от 2 до 15 раз по сравнению с одним

ядром универсального процессора. Наибольшее ускорение получено на АрУ нового поколения NVIDIA GTX470 (от 1,35 до 1,7 раз больше чем другие АрУ).

На рисунках 6-8 представлены ускорения, полученные при выполнении вычислений на ГВС-10 «Кубань» и мультипроцессорной системе на задачах различных размеров для различных потенциалов.

Для ГВС-10 «Кубань» максимально задействовано 8 MPI-процессов, при этом вычисления выполнялись на 4 ядрах универсального процессора и 8 арифметических ускорителях.

Для мультипроцессорной системы максимально задействовано 64 MPI-процесса, при этом вычисления выполнялись на 64 ядрах универсальных процессоров и 64 арифметических ускорителях.

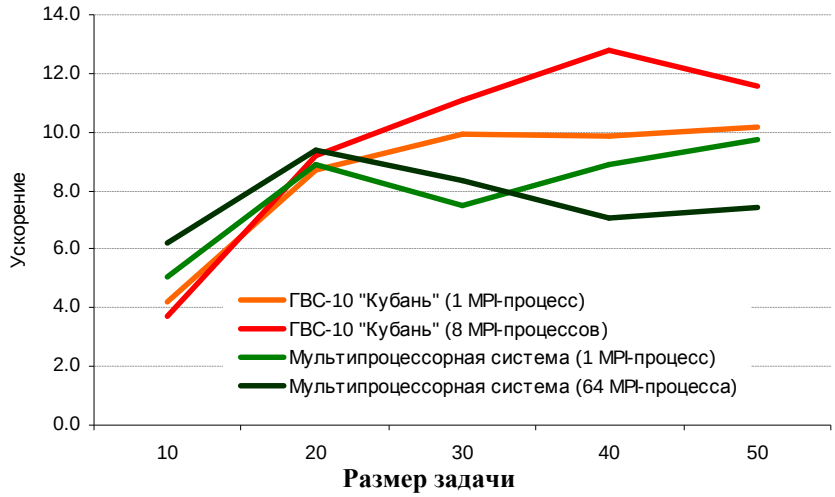


Рисунок 6 – Ускорение вычислений по программе МД (потенциал Морзе)

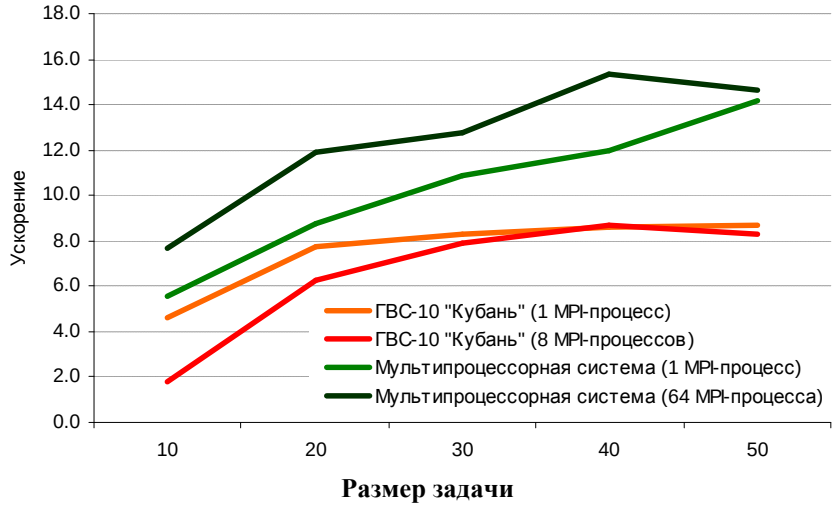


Рисунок 7 – Ускорение вычислений по программе МД (потенциал EAM)

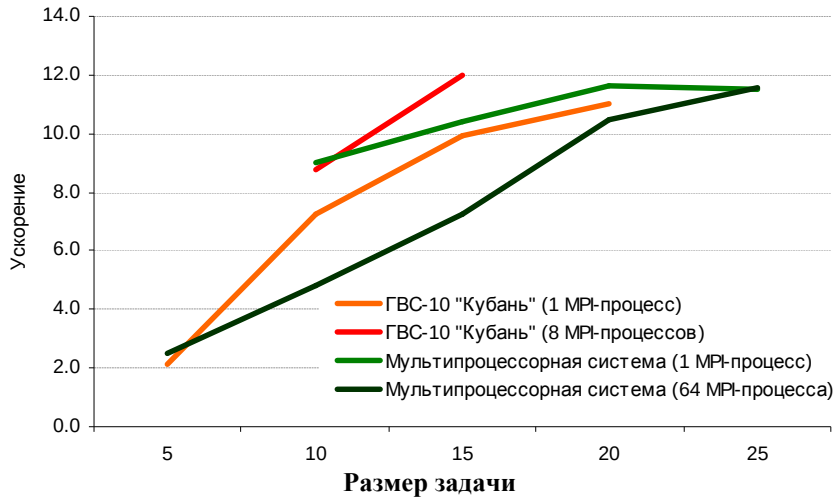


Рисунок 8 – Ускорение вычислений по программе МД (потенциал MEAM)

Ускорение, полученное на обеих гибридных вычислительных системах, на одном и на максимальном количестве MPI-процессов отличается не более чем на 20% на различных размерах задач для потенциалов Морзе, EAM и MEAM.

Перспективы

С применением графических ускорителей проведено моделирования всестороннего сжатия сплава Pu-Ga (рисунок 9). Результаты получены в кратчайшие сроки, время моделирования составило около 5 суток, без применения ускорителей длительность выполнения возрастает на порядок и составит примерно 50 суток.

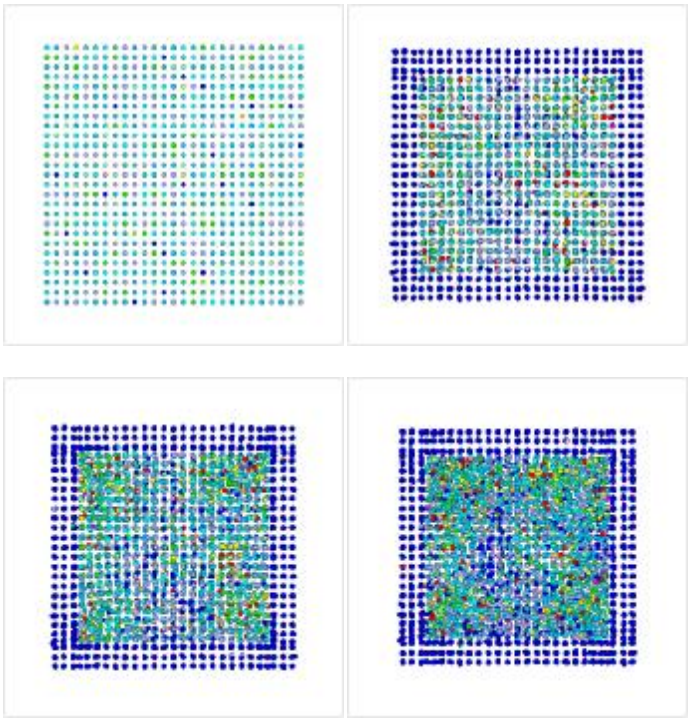


Рисунок 9 – Моделирование всестороннего сжатия монокристаллического образца Pu-Ga (содержание Ga 5.2 %).

Заключение

Реализованный в данной работе подход организации вычисления значений межчастичного взаимодействия позволяет производить производственные расчеты по всем основным типам потенциалов. Разработаны программы молекулярно-динамического решателя в соответствии с использованием табличных значений потенциалов и их производных для выполнения на АрУ. Добавлены ускорительные функции интерполяции парных и многочастичных потенциалов. Программы доработаны с учетом использования в расчетах нескольких материалов.

Все ускорительные функции MEAM потенциала были доработаны для использования частиц двух и более типов. При этом набор аналитических функций для ускорителя претерпел лишь небольшие изменения, связанные с добавлением нескольких дополнительных параметров – множителей.

Все эти изменения позволили провести расчеты получения холодных кривых для сплава Pu-Ga (актуально для моделирования упругопластических свойств конструкционных материалов).

В целом возможность применения в расчете нескольких материалов расширяет класс решаемых задач на ускорителях.

В работе получены длительности выполнения на различных гибридных вычислительных системах (однопроцессорной персональной системе, специализированной компактной вычислительной системе ГВС-10 «Кубань», гибридных мультипроцессорной системе). На каждой из приведенных систем получены значения ускорений, относительно универсальных составляющих.

Длительность вычислений уменьшена на однопроцессорной персональной системе по сравнению с одним ядром универсального процессора:

- потенциал Морзе - от 4 до 10 (АрУ NVIDIA GTX260) и от 9 до 19 раз (АрУ NVIDIA GTX470);
- потенциал EAM - от 4 до 9 (АрУ NVIDIA GTX260) и от 9 до 17 раз (АрУ NVIDIA GTX470);
- потенциал MEAM - от 3 до 14 (АрУ NVIDIA GTX260) и от 5 до 23 раз (АрУ NVIDIA GTX470).

Длительность вычислений уменьшена на ГВС-10 «Кубань» по сравнению с одним ядром универсального процессора:

- потенциал Морзе - от 4 до 10 раз;
- потенциал EAM - от 4 до 9 раз;
- потенциал MEAM - от 2 до 11 раз.

Длительность вычислений уменьшена на мультипроцессорной системе по сравнению с одним ядром универсального процессора:

- потенциал Морзе - от 5,1 до 9,7 раз;
- потенциал EAM - от 5,6 до 14,2 раз;
- потенциал MEAM - от 9 до 11,6 раз.

Выполнены расчеты в многопроцессорном режиме, показано масштабирование ускорения на всех исследованных вычислительных системах.

Выполнено сравнение различных арифметических ускорителей, включая АрУ нового поколения NVIDIA GTX470. Полученные результаты позволяют рассчитывать на возможность эффективного использования арифметических ускорителей нового поколения для расчета задач молекулярной динамики.

Список использованных источников

1. Воронин Б.Л., Ерофеев А.М., Копкин С.В., Крючков И.А., Рыбкин А.С., Степаненко С.А., Южаков В.В. Применение графических арифметических ускорителей для расчета задач молекулярной динамики по программному комплексу МД. Доклад X Международный семинар Супервычисления и Математическое Моделирование 29.09 – 3.09, Саров, 2008.
2. Крючков И.А., Копкин С.В. Адаптация алгоритма расчета взаимодействия для многочастичного потенциала MEAM на гибридных вычислительных системах. Доклад XI Международный семинар Супервычисления и Математическое Моделирование 5.10 – 9.10, Саров, 2009.
3. С.В. Копкин, И.А. Крючков. Алгоритм модернизированного многочастичного потенциала для молекулярно-динамического моделирования на графическом арифметическом ускорителе// Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2010. Вып. 3.
4. Коваленко Н.О., Воронин Б.Л., Копкин С.В., Грушин С.А., Кечин А.Г., Анисимов А.Н., Иоилев А.Г., и др. Молекулярно-динамическое моделирование взаимодействия расплавленных наночастиц с поверхностями металлов. Доклад Международная научная конференция по проблемам физики высоких плотностей энергии «XII научные Харитоновские чтения» 19-23 апреля, Саров, 2010

Программный комплекс на базе гибридных вычислительных систем для расчета критических параметров методом Монте-Карло

А.С. Рыбкин, А.Н. Залялов, А.Г. Малькин, С.П. Огнев, В.И. Рослов
Российский федеральный ядерный центр — Всероссийский научно-исследовательский институт экспериментальной физики. Институт теоретической и математической физики (РФЯЦ-ВНИИЭФ ИТМФ)

Введение

Одной из основных сфер внедрения программных комплексов на базе гибридных вычислительных систем с графическими арифметическими ускорителями является атомная энергетика. В частности, важнейшей задачей при разработке и эксплуатации сложных технических объектов атомной энергетики является расчёт критических параметров систем.

В практике ВНИИЭФ для расчета критических параметров методом Монте-Карло используется программный комплекс, являющийся развитием программы С-95[1], написанный на языке программирования Фортран-90 и предназначенный для универсальных вычислительных систем. Задействование графических арифметических ускорителей потребовало создание новой программы, способной выполняться на гибридных вычислительных системах. Разработка программы, адаптированной к особенностям архитектуры гибридной вычислительной системы, потребовала коренного изменения структуры комплекса. В данном докладе приводятся результаты разработки программы, в результате чего основная часть кода для расчета критических параметров комплекса переведена на графические арифметические ускорители. Представлены характерные для данного класса задач тесты и первые численные исследования разрабатываемой программы на гибридных вычислительных системах.

Численные исследования проводились на высокопроизводительной вычислительной системе ГВС-10 «Кубань» с универсальным процессором архитектуры x86 и арифметическим ускорителем (АрУ) основанном на графических процессорах фирмы NVIDIA GeForce GTX295.

Постановка задачи

Метод Монте-Карло является одним из основных методов для определения эффективного коэффициента размножения нейтронов активных зон (АЗ) ядерных реакторов, обоснования радиационной и ядерной безопасности транспортных упаковочных комплектов (ТУК) для перевозки и долговременного хранения отработанного ядерного топлива АЭС. Для данного класса задач характерным является наличие большого количества (порядка нескольких сотен) тепловыделяющих систем (ТВС), которые заполняют внутренность АЗ или ТУК. Каждая из ТВС, в свою очередь, содержит несколько сотен тепловыделяющих элементов (ТВЭЛ), конструкционных и управляющих стержней. Кроме того, ТВС отличаются друг от друга химическими составами ТВЭЛ. Такое детальное описание задач создает определенные трудности, как при подготовке начальных данных, так и при проведении расчетов.

В разработанной программе основным объектом в задании геометрических данных выбран геометрический блок. Блок представляет собой совокупность областей и поверхностей определенного типа.

В задаче может быть описано произвольное количество блоков. Они рассматриваются как независимые «строительные» элементы, которые следует так разместить друг относительно друга, чтобы составить требуемую геометрию системы. Геометрия, составленная из вложенных друг в друга блоков, напоминает «матрешку». Отличие состоит лишь в том, что в блок можно вкладывать не один, а несколько других (не обязательно

разных) блоков. Важно лишь, чтобы размещенные блоки не пересекались между собой и не выходили за пределы блока-вместилища.

Расчет эффективного коэффициента размножения нейтронов $k_{эфф}$ выполняется методом Монте-Карло по поколениям нейтронов. Очередное поколение нейтронов формируется моделированием траектории нейтронов предыдущего поколения до первой точки деления. Во время моделирования производится снятие результатов с траекторий частиц. Моделирование прекращается по достижении заданной точности $k_{эфф}$.

Источник определяет фазовые параметры начального распределения нейтронов. В начале счета задач, в течение нескольких поколений нейтронов, выполняется моделирование траекторий без снятия с них результатов (процесс установления собственной функции).

Моделирование траекторий осуществляется в соответствии с выбранной системой констант взаимодействия частиц с веществами. Для розыгрыша свободного пробега и выбора вещества, на котором произошло столкновение, используется схема максимальных кусочно-постоянных сечений. Учет теплового движения ядер среды происходит либо в приближении свободного максвелловского газа (FRGAS), либо с учетом химических связей вещества (модель $S(\alpha,\beta)$).

Для расчета $k_{эфф}$ используется одна оценка по пробегу и три оценки по столкновениям: по собственно столкновениям, по делениям и по поглощениям. Оптимальная оценка вычисляется на основе этих четырех оценок.

Модификация программы для использования на АрУ

Специфика программирования на АрУ потребовала существенного изменения алгоритма работы программы и системы хранения данных.

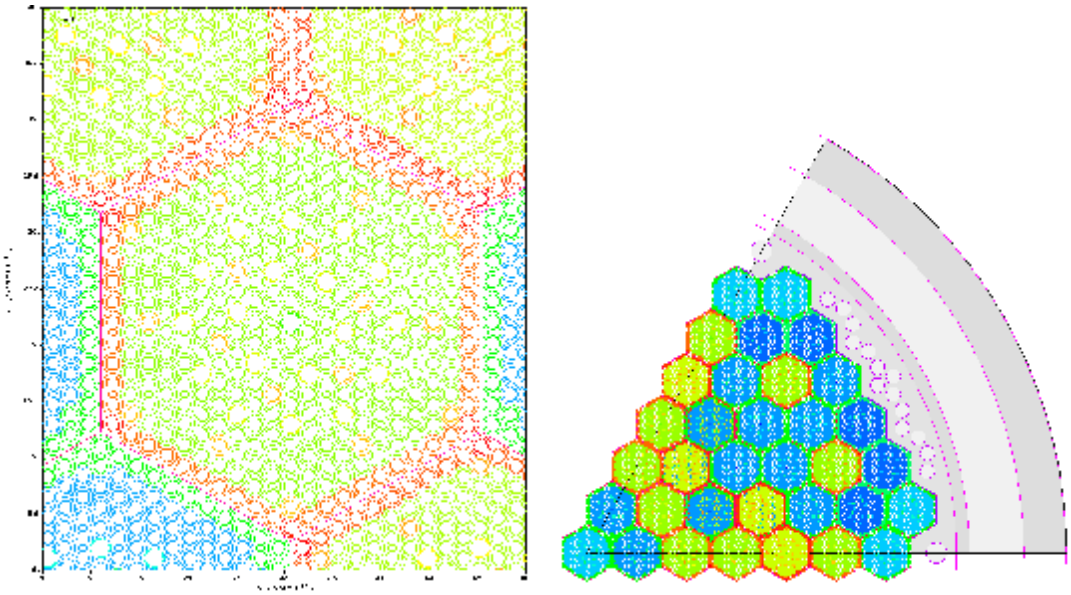
Счетная часть программы была переведена на язык программирования С. При этом все структуры данных были трансформированы в одномерные массивы, передача данных в процедуры и функции осуществлялась явно в виде параметров подпрограмм без использования модулей.

В отличие от стандартного метода, где все частицы одного поколения моделируются последовательно, в новой программе предназначенной для работы на гибридных вычислительных системах с графическими арифметическими ускорителями различные части траектории частиц (расчет расстояния, свободный пробег, розыгрыш столкновения и т.д.) рассчитываются параллельно для всех частиц поколения.

Описание тестовых задач

Для тестирования программы выбраны две тестовые задачи. Их подготовка основана на результатах расчета активной зоны реактора ВВЭР-1000 с 1/3 загрузкой МОХ-топлива, выполненной специалистами Курчатовского института [2].

В качестве первого теста была выбрана бесконечная система, состоящая из ТВС с одним и тем же химическим составом ТВЭЛ. Геометрия задачи представлена на рисунке 1а, в качестве второго теста было выбрано одно из состояний АЗ реактора ВВЭР-1000 из работы [2]. Картограмма заполнения АЗ представлена на рисунке 1б. В силу симметрии АЗ расчеты проводились 1/6 части АЗ с применением тактики отражения на границах. На рисунке разные химические составы ТВЭЛ в ТВС выделены разным цветом.



а) б)

Рисунок 1–(а) Поперечное сечение ТВС и (б) 1/6 картограммы заполнения АЗ.

Гибридная вычислительная система ГВС-10 «Кубань»

Система содержит вычислительный модуль с универсальным процессором Intel Core i7-920, работающий на тактовой частоте 2,66 ГГц и арифметический ускоритель.

АрУ состоит из графических процессоров (ГП) NVIDIA GT200 и оперативной памяти типа GDDR3. Объем оперативной памяти доступный каждому ГП составляет 896 МБ.

Структура гибридной вычислительной системы представлена на рисунке 2.

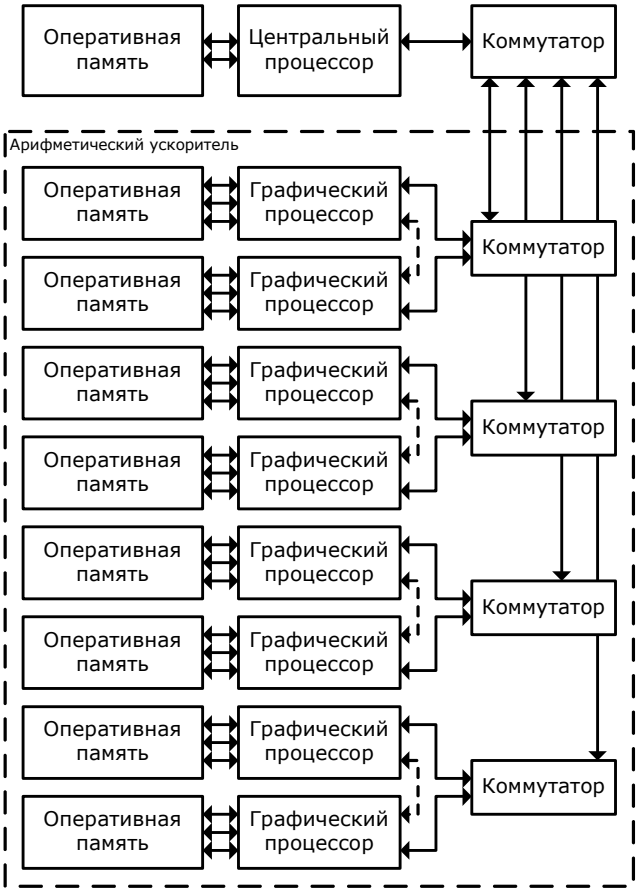


Рисунок 2 – Структурная схема ГВС-10 «Кубань»

ГП NVIDIA GT200 содержит 30 векторных ядер и 7 64-битных контроллеров памяти. Векторные ядра ГП работают на частоте 1,242 ГГц. Теоретическая пиковая производительность вычислительной системы составляет 638 ГФ/с. Агрегированная пропускная способность памяти АрУ составляет 896 ГБ/с. Интерфейс подключения АрУ – PCI Express 2.0 x16 (четыре шины).

В состав системного программного обеспечения входят операционная система Microsoft Windows Server 6 R2 для платформы x86_64, библиотека межпроцессорных обменов MPI MPICH2-1.1.1, сконфигурированная для работы с системой программирования Intel Fortran 11.0.72. Для графического арифметического ускорителя использовалась NVIDIA CUDA 3.0 SDK.

Результаты тестирования

В данной главе приводятся результаты тестирования программы в четырех конфигурациях:

- вычислительный процесс с использованием одного ядра универсального процессора;
- вычислительный процесс с использованием четырех ядер универсального процессора;
- вычислительный процесс с использованием одного ядра универсального процессора и одного ГП АрУ;
- вычислительный процесс с использованием четырех ядер универсального процессора и восьми ГП АрУ.

Численные исследования выполнялись на различном числе траекторий в пакете (ансамбле). Максимальное количество траекторий в пакете определялось доступным объемом оперативной памяти ГП. Для иллюстрации все расчеты выполнялись для двух пакетов установления собственной функции и двух пакетов счета.

Далее на рисунках 3-4 приняты следующие обозначения коэффициентов ускорения:

- «k0». Коэффициент ускорения, определяемый соотношением длительности вычисления на одном ядре универсального процессора Intel Core i7-920 к длительности вычисления одного ядра универсального процессора и на одного графическом процессоре АрУ.
- «k1». Коэффициент ускорения, определяемый соотношением длительности вычисления на четырех ядрах универсального процессора Intel Core i7-920 к длительности вычисления на четырех ядрах универсального процессора и восьми графических процессорах АрУ.

Расчет тепловыделяющей сборки

В таблицах 1-2 представлены длительности счета для различных частей разработанной программы: 1 – розыгрыш параметров частицы после столкновений; 2 – процедура розыгрыша столкновений частиц; 3a – блок гибели; 3b - расчет расстояния до границы области вдоль направления полета частицы; 3c – розыгрыш свободного пробега частиц; sort – сортировка частиц для оптимальной загрузки АрУ.

В таблице 1 представлено время счета ТВС при задействовании одного ядра универсального процессора.

Таблица 2 – Длительность вычислений на универсальном процессоре, в секундах

Тип	Количество траекторий в пакете, тыс.шт.						
	1.6	3.2	6.4	12.8	25.6	51.2	102.4
1	0.4	0.8	1.9	4.3	9.4	21.6	46.0
2	2.0	4.3	9.7	21.2	46.8	109.2	235.6
3a	1.8	3.8	9.1	22.0	52.2	131.4	282.1
3b	45.0	90.8	183.2	370.5	748.0	1537.8	3113.1
3c	9.2	19.3	40.8	86.0	186.0	419.5	889.1
Sort	0.3	0.6	1.2	2.5	6.2	18.7	55.9
ИТОГО	58.7	119.6	245.8	506.4	1048.6	2238.1	4621.9

Аналогично, в таблице 2 представлены значения длительности вычислений при задействовании АрУ.

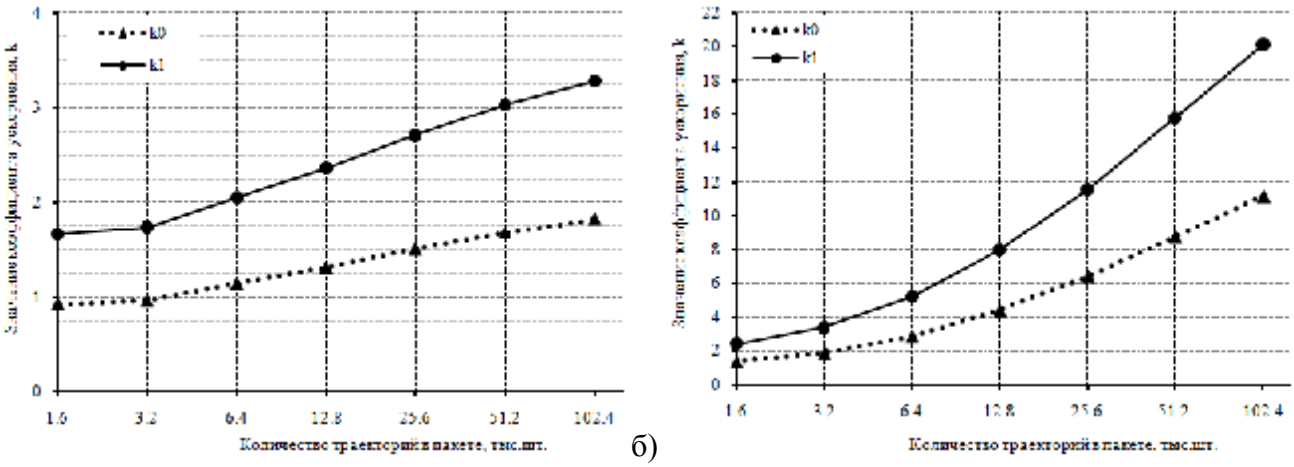
Таблица 3 – Длительность вычислений на АрУ, в секундах

Устройство	Тип	Количество траекторий в пакете, тыс.шт.						
		1.6	3.2	6.4	12.8	25.6	51.2	102.4
Графический арифметический ускоритель	1	0.6	1.2	1.7	2.3	3.5	6.0	10.4
	2	1.2	3.3	5.7	9.0	14.0	24.5	42.1
	3a	0.5	1.2	1.6	2.9	3.6	5.8	7.9
	3b	10.6	19.2	27.5	38.8	58.8	99.2	170.7
	3c	5.5	13.7	22.9	33.1	50.2	85.8	145.4
	sort	20.3	60.6	131.1	272.5	534.9	1076.5	2125.4
Универсальный процессор	1	0.2	0.2	0.2	0.3	0.3	0.3	0.3
	2	1.3	1.5	1.7	2.0	2.3	2.5	2.6
	3a	0.9	1.0	1.1	1.4	1.7	2.0	2.1
	3b	17.0	16.1	15.5	16.8	20.0	19.0	18.3
	3c	5.2	5.6	6.0	6.9	7.9	8.0	8.2
	sort	0.2	0.2	0.2	0.3	0.3	0.3	0.3
	ИТОГО	63.4	123.8	215.2	386.3	697.5	1330.0	2533.7

Из таблицы видно, что большую часть времени занимает сортировка частиц. Данная процедура в настоящее время реализована в последовательном режиме. Ведутся работы по ее распараллеливанию, что приведет к снижению времени счета. Также достаточно большое время занимает поиск расстояний вдоль полета частицы. Авторы изучают возможность оптимимизации алгоритмов для ускорения этого блока.

Необходимо отметить, что при масштабировании количества ГП в АрУ эффективность использования остается на прежнем уровне, и не снижается с увеличением их числа.

На рисунке 3 проиллюстрированы значения коэффициентов ускорения при расчете ТВС при различных значениях количества траекторий в пакете.



а) б)

Рисунок 10 – Значения коэффициентов ускорения: а) всей задачи, б) счетной части (без сортировки)

При количестве траекторий равного 102400 шт. ускорение всей задачи составляет 1,82 раза при задействовании одного графического процессора АрУ. С использованием восьми

ГП АрУ длительность вычислений уменьшается в 3,28 раза по сравнению с четырьмя ядрами универсального процессора Intel Core i7-920.

Расчет активной зоны

В таблице 3 представлено время счета отдельных частей расчета активной зоны (АЗ) на универсальном процессоре.

Таблица 4 – Длительность вычислений на универсальном процессоре, в секундах

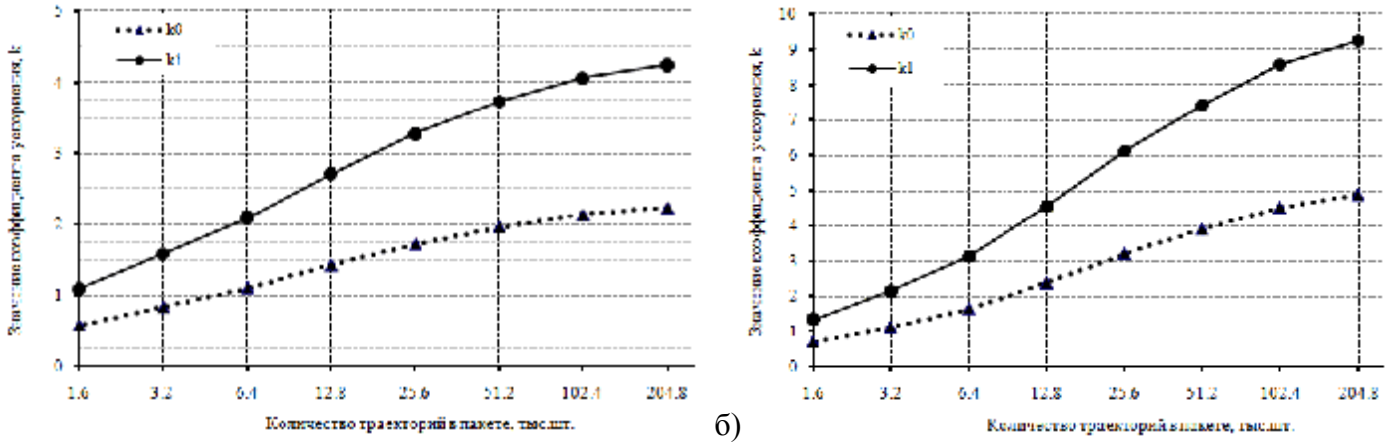
Тип	Количество траекторий в пакете, тыс.шт.							
	1.6	3.2	6.4	12.8	25.6	51.2	102.4	204.8
1	0.4	0.9	2.1	4.8	10.6	24.5	51.6	108.1
2	2.0	4.2	9.4	21.5	48.1	111.2	234.5	497.6
3a	1.8	3.7	8.9	22.1	54.3	135.7	280.7	624.0
3b	48.8	98.4	197.1	402.4	814.4	1664.0	3395.3	6824.9
3c	13.6	28.3	59.0	126.9	266.7	574.9	1178.2	2389.2
Sort	0.1	0.3	0.5	1.4	3.4	7.8	19.9	55.4
ИТОГО	66.7	135.8	276.9	579.0	1197.5	2518.2	5160.2	10499.2

В таблице 4 приведено время счета задачи с использованием АрУ.

Таблица 5 – Длительность вычислений на АрУ, в секундах

Тип	Количество траекторий в пакете, тыс.шт.							
	1.6	3.2	6.4	12.8	25.6	51.2	102.4	204.8
1	0.6	0.7	1.0	1.6	2.9	5.5	10.9	21.4
2	2.7	3.3	4.6	6.3	9.8	17.5	34.0	68.1
3a	0.3	0.4	0.5	0.6	0.9	1.5	3.0	6.5
3b	65.4	84.6	118.8	174.9	271.4	473.1	823.2	1538.6
3c	24.3	31.0	42.1	57.9	85.9	144.3	265.8	507.5
Sort	22.6	43.0	83.9	164.0	323.9	642.2	1278.6	2557.7
ИТОГО	115.9	163.0	250.9	405.3	694.8	1284.0	2415.4	4699.8

На рисунке 4 проиллюстрированы значения коэффициентов ускорения при расчете АЗ при различных значениях количества траекторий в пакете.



а) б)

Рисунок 11 – Значения коэффициентов ускорения: а) всей задачи, б) счетной части (без сортировки)

При количестве траекторий равного 204800 шт. ускорение всего расчета АЗ составляет 4,24 раза по сравнению с четырьмя ядрами универсального процессора при задействовании восьми графических процессоров ГВС-10 «Кубань».

Заключение

При количестве траекторий равного 102400 шт. ускорение расчета ТВС составляет 1,82 раза при задействовании одного графического процессора АрУ. С использованием восьми ГП АрУ длительность вычислений уменьшается в 3,28 раза по сравнению с четырьмя ядрами универсального процессора Intel Core i7-920.

При количестве траекторий равного 204800 шт. ускорение всей задачи расчета активной зоны составляет 4,24 раза по сравнению с четырьмя ядрами универсального процессора при задействовании восьми графических процессоров ГВС-10 «Кубань».

Список использованных источников

1. *Кочубей Ю.К., Житник А.К., Огнев С.П., Семенова Т.В. и др.* Программа С-95. Моделирование совместного переноса нейтронов и γ-квантов методом Монте-Карло // Вопросы атомной науки и техники. Серия Математическое моделирование физических процессов. 2000. № 2. С. 49-52.
2. E.Gomin, M.Kalugin, D.Oleynik. *VVER-1000 MOX Core Computational Benchmark*. OECD 2006, NEA №6088.

Тезисы стендовых докладов

Оценки ускорения вычислений гибридными системами

Степаненко С.А.

Российский федеральный ядерный центр — Всероссийский научно-исследовательский институт экспериментальной физики. Институт теоретической и математической физики (РФЯЦ-ВНИИЭФ ИТМФ)

ssa@vniief.ru

Тезисы

В современной практике для ускорения вычислений активно применяются гибридные вычислители. Они содержат параллельные компоненты двух видов – MIMD (multy instruction multy data) компоненту и SIMD (single instruction multy data) компоненту. Вычислительный процесс распределяется между этими компонентами и лишь затем между процессорами, образующими эти компоненты.

Результирующее ускорение зависит от ускорений, достигаемых на MIMD и SIMD компонентах и от размеров «долей» вычислительного процесса, приходящихся на эти компоненты.

В этой работе исследованы архитектуры гибридных вычислительных систем, содержащих универсальные процессоры (MIMD-компонент) и арифметические ускорители (SIMD-компонент). В частности:

- получены коэффициенты ускорения - аналитические соотношения, которые позволяют оценить ускорения вычислительного процесса гибридными вычислительными системами различных конфигураций по сравнению с универсальным процессором. Исходной информацией для таких оценок являются определенные первичные параметры вычислительного процесса, измеряемые (теоретически или экспериментально) применительно к простейшему вычислителю из одного процессорного ядра и одного ускорителя.
- сформулированы условия, при выполнении которых в составе гибридного вычислителя целесообразно наращивание количества процессоров, либо количества ускорителей;
- получены оценки предельных значений ускорений; они определяются первичными параметрами вычислительного процесса.

Полученные соотношения обобщают известные ранее оценки длительностей вычислений гибридными системами и являются инструментом создания эффективных гибридных систем.