УДК 519.622.2

Реализация проекционного явного метода решения жесткой системы ОДУ на графическом процессоре общего назначения

С.Н. Чадов
ФГБОУВПО«Ивановский государственный энергетический университет имени В.И. Ленина»,
г. Иваново, Российская Федерация
E-mail: sergei.chadov@gmail.com

Авторское резюме

Состояние вопроса: Для решения жестких систем ОДУ, как правило, применяются неявные методы, которые имеют низкую вычислительную эффективность. В последнее время растет интерес к решению подобных систем явными методами, в частности методом Гира, разработана группа так называемых явных проекционных методов. С другой стороны, множество различных вычислительных задач решается на графических процессорах общего назначения, что позволяет добиться существенного повышения производительности. В связи с этим актуальным является исследование реализации явного проекционного метода с использованием графического процессора общего назначения**.**
Материалы и методы: Рассмотрена математическая модель энергетической системы, состоящей из турбины и регулятора. Численные эксперименты по интегрированию этой модели проведены с использованием проекционного метода Эйлера на центральном процессоре ПК (как в однопоточном, так и в многопоточном варианте) и графическом процессоре общего назначения.
Результаты: Описана реализация модели системы и проекционного метода Эйлера на графическом процессоре, проведены численные эксперименты по сравнению производительности данной реализации и реализации на центральном процессоре. Описаны необходимые модификации математической модели и их программные оптимизации. Показано, что использование графического процессора позволяет в некоторых случаях повысить производительность вычислений в 10 и более раз, по сравнению с четырехъядерным центральным процессором.
Выводы: Полученные результаты свидетельствуют о том, что графические процессоры имеют существенное преимущество в производительности для моделирования рассмотренной системы, что позволяет рекомендовать их для решения этой и подобных задач.

Ключевые слова: графические процессоры общего назначения, проекционный метод Эйлера, жесткие системы ОДУ, CUDA.

A GPU-based implementation of a projective stiff ODE solution

S.N. Chadov
Ivanovo State Power Engineering University, Ivanovo, Russian Federation
E-mail: sergei.chadov@gmail.com

Abstract

Background: Stiff ODE systems are commonly solved by implicit methods of low numerical efficiency. Lately, explicit methods have been increasingly employed for this task, one example being explicit projection methods developed by Gear et al. A number of the so-called explicit projection methods have been developed. On the other hand, a lot of computation is done by general purpose GPUs with a significant performance increase. Therefore, it is urgent to study the explicit projection method realization by using a general purpose GPU.
Materials and methods: The paper presents a power system model consisting of a turbine and a control system. The model integration numerical experiments are carried out by the Euler projection method on the CPU (both single- and multithreaded types) and on a general purpose GPU.
Results: The paper describes the implementation of the model system and Euler projection method on a GPU, the numerical experiments comparing the model realization performance on the CPU and GPU. It also reports on the necessary adjustments of the model and software optimizations. The resulting GPU performance is shown to be more than 10 times higher than that of the quad-core CPU.
Conclusions: The results show that using GPUs has substantial performance benefits and can be recommended for this and related tasks.

Key words: General Purpose GPU, Euler projection method, stiff ODE, CUDA.

Введение. В работе [1] рассматриваются уравнения, описывающие систему, состоящую из турбогенератора, центробежного регулятора и некоторой нагрузки. Показано, что для интегрирования такой системы явные жесткие методы, в частности проекционный метод Эйлера, имеют значительное преимущество в производительности перед широко распространенными явными и неявными методами. Однако, если мы будем рассматривать аналогичную систему, в которой количество уравнений имеет порядок нескольких

тысяч, просто выбора оптимального алгоритма недостаточно для обеспечения высокой производительности. Одним их способов решения данной проблемы является разработка параллельной версии алгоритма интегрирования. В частности, интерес представляет реализация программы решения на графическом процессоре общего назначения, так как это устройство обладает значительно большей теоретической пиковой производительностью, чем современные центральные процессоры. Однако практические результаты, в зависимости от задачи и особенностей реализации, могут быть различными, сам факт реализации на графическом процессоре еще не гарантирует ускорения решения. Рассмотрим реализацию интегрирования системы из [1] на GPU NVIDIA с использованием технологии CUDA.

Модель электромеханической системы. Для исследования возьмем модель системы, рассмотренной в [1] и [2].

Система состоит из турбогенератора, центробежного регулятора и некоторой нагрузки. Вращение турбины происходит под действием крутящего момента Q, момента сопротивления, зависящего от квадрата угловой скорости вращения турбины, и момента, создаваемого нагрузкой. Для угловой скорости вращения уравнение будет иметь вид

$$\frac{d\omega}{dt} = \gamma Q - \eta\omega|\omega| - \xi\omega I,$$

где ω – угловая скорость вращения турбины; I – сила тока, проходящего через нагрузку; η, ξ, γ – некоторые коэффициенты.

Состояние регулятора описывается следующим уравнением:

$$\frac{d^2\rho}{dt^2} = \omega^2\rho - \alpha\frac{d\rho}{dt} - k^2(\rho - \rho_0),$$

где ρ – уровень регулирования (т. е. отклонение регулятора от оси вращения); k^2 – отношение жесткости пружины к центробежной массе; α – коэффициент, учитывающий действие вязких сил; ρ_0 – значение ρ в состоянии покоя.

Примем, что Q зависит от ρ следующим образом:

$$Q = Q_{\max}\frac{e^{-\theta}}{1+e^{-\theta}},$$

где

$$\theta = \frac{\rho - \rho_{\min}}{\rho_{\max} - \rho_{\min}}\upsilon_1 - \upsilon_2.$$

Коэффициенты υ_1, υ_2 выберем таким образом, чтобы обеспечить достаточно плавное изменение Q в зависимости от ρ на всей области определения, например, $\upsilon_1 = 4$, $\upsilon_2 = 2$.

Сила тока вычисляется из уравнения

$$(r + R)I = \psi\omega = \varepsilon,$$

где r – внутреннее сопротивление генератора; R – сопротивления нагрузки; ε – ЭДС генератора.

Для нескольких генераторов, подсоединенных к общей нагрузке, уравнения для ω и ρ не изменятся, за исключением того, что в системе теперь будет по паре таких уравнений для каждого генератора. Уравнение связи, используемое для вычисления силы тока, найдем из закона Ома:

$$R\sum I_i = \varepsilon$$

и упрощенного предположения, что ЭДС генератора прямо пропорциональна скорости вращения ротора:

$$\psi_i\omega_i = \varepsilon_i.$$

Выразим I_i и проведем рад алгебраических преобразований. В результате получим следующее выражение:

$$I_i = \frac{\frac{\psi_i}{r_i}\omega_i\left(\sum\frac{R}{r_j} - 1\right) - \frac{R}{r_i}\sum\frac{\psi_j}{r_j}\omega_j}{1 - \sum\frac{R}{r_j}}.$$

Значения коэффициентов (табл. 1) также возьмем из [1].

Таблица 1. **Параметры системы**

Параметр	Значение (m – номер генератора)
γ	9,31 + 0,1m
η	0,00323
ξ	0,0001
α	80
k^2	520
$\rho_0, \rho_{\max}, \rho_{\min},$	5,10,50
$Q_{\max}$	1
ψ	20
R	100
r	10

Метод интегрирования. Для интегрирования системы будем использовать проекционный метод Эйлера [1, 3], реализованный на графическом процессоре. Графические процессоры ввиду особенностей архитектуры (множество относительно медленных потоковых процессоров) получают преимущество перед центральным процессором лишь в том случае, если задачу можно разбить на большое количество параллельных подзадач. Таким образом, задачи невысокой размерности (до сотен уравнений) решать на GPU, как правило, неэффективно. Для задач высокой размерности можно выделить две сферы применения графических процессоров: распараллеливание вычисления правой части системы, а также, для неявных методов, распараллеливание решения получаемой системы алгебраических уравнений. Другие источники параллелизма при решении систем ОДУ (например, параллельное исполнение

шагов метода Рунге-Кутты), как правило, не играют большой роли при использовании GPU, так как не способны обеспечить распараллеливание на сотни и тысячи потоков.

Рассмотрим реализацию решения системы для *N* порядка нескольких тысяч генераторов. Для простоты примем, что количество генераторов является степенью двойки (ниже будет показано, чем вызвано такое требование). На решении реальной задачи такое упрощение не скажется, так как мы всегда можем дополнить систему виртуальными генераторами, не влияющими на решение.

Наиболее очевидная стратегия распараллеливания любой системы – выделить по одному потоку на каждое уравнение. Однако от такого решения было принято отказаться по ряду причин:

1. Разные уравнения требуют разного количества вычислительных операций, соответственно, загрузка вычислительных устройств будет неравномерной.

2. Вычисления, выполняемые каждым потоком, имеют небольшой объем, в результате чего на производительность большее влияние начинают оказывать накладные расходы на вызов функций, а не вычисления.

В связи с этим будем использовать по одному потоку на каждый генератор, т. е. каждый поток будет отвечать за решение трех уравнений. Таким образом мы добиваемся более равномерной загрузки вычислительных устройств.

Что касается конфигурации блоков, выберем одномерную последовательность с 128 потоками в блоке. Число 128 выбрано исходя из следующих соображений:

1. Является степенью двойки, что упрощает реализацию редукции.

2. Содержит 4 warp, что обеспечивает загрузку исполнительных модулей GPU. Напомним, что warp является минимальной единицей, которой оперирует планировщик потоков графического процессора. Таким образом, наличие 4 warp на блок и, соответственно, на исполнительное устройство позволяет маскировать время выполнения длительных асинхронных операций, например, обращений к памяти.

3. Для рассматриваемого количества уравнений обеспечивает достаточное количество блоков для загрузки всех исполнительных устройств современных GPU.

Если запрограммировать интегрирование системы непосредственно по приведенным формулам, производительность полученной программы будет значительно меньше ее производительности на CPU. Чтобы получить существенное ускорение, нужно немного преобразовать модель для лучшего распараллеливания.

Рассмотрим формулу вычисления силы тока:

$$I_i = \frac{\frac{\psi_i}{r_i}\omega_i\left(\sum\frac{R}{r_j}-1\right)-\frac{R}{r_i}\sum\frac{\psi_j}{r_j}\omega_j}{1-\sum\frac{R}{r_j}}.$$

Во-первых, можно заметить, что коэффициенты ψ, *R, r* не меняются на протяжении решения, поэтому можно предварительно рассчитать значения $\sum\frac{R}{r_j}$ и подобные. На CPU такая оптимизация также возможна, однако не даст значительного прироста эффективности, тогда как на GPU, где время обращения к глобальной памяти на несколько порядков превышает время исполнения арифметических операций, эффект от такой оптимизации будет значителен. Немаловажно также то, что вычисление $\sum\frac{R}{r_j}$ требует обращения к памяти, обрабатываемой другими потоками, т. е. фактически операции редукции (хотя и упрощенной, так как значение не меняется). В выражении $\frac{R}{r_i}\sum\frac{\psi_j}{r_j}\omega_j$ ω_j меняется на каждом шаге, так что для вычислений потребуется полноценная редукция, что сложно организовать в середине функции на графическом процессоре. Поэтому выделим специальную область в глобальной памяти, где будет храниться результат редукции, и будем обновлять его при каждом вычислении правой части. Задачу можно немного упростить, потребовав, чтобы для всех генераторов выполнялось условие $\psi_i = \psi$, в таком случае уравнение для вычисления силы тока можно переписать следующим образом:

$$I_i = \frac{(S_{Rr}-1)\Psi_i\omega_i-\frac{R}{r_i}\Psi_i S_\omega}{1-S_{Rr}},$$

где S_{Rr} – предварительно вычисленное значение $\sum\frac{R}{r_j}$; Ψ_i – предварительно вычисленное значение $\sum\frac{\psi_j}{r_j}$; S_ω – результат редукции $\sum\omega_j$.

Реализация операции редукции. Касаемо реализации операции редукции, наиболее распространенным подходом является двухэтапная древовидная редукция, подробно описанная в работе [4]. Однако далее покажем, что для нашей системы такой подход является субоптимальным. Причина этого состоит в том, что для явного метода решения системы ОДУ и нашего способа разделения по потокам вычисления правой части накладные расходы, связанные с запуском ядра, составляют существенную долю общего

времени выполнения программы. В этом случае естественной оптимизацией является минимизация количества используемых ядер. Поэтому вместо двухэтапной редукции будем использовать одноэтапную, построенную на атомарных операциях. Нужно отметить, что необходимые для реализации атомарные операции доступны в устройствах с compute capability 1.1 и выше, однако на устройствах с compute capability менее 2.0 производительность их невелика, так что для этих устройств, возможно, стоит рассмотреть использование двухэтапной редукции.

Основная идея одноэтапной редукции заключается в том, что каждый блок после выполнения первого этапа редукции (редукции внутри каждого блока) атомарно увеличивает значение глобального счетчика. Тот блок, который увеличивает значение счетчика до значения, равного количеству блоков, очевидно, является последним и может выполнить второй этап редукции. Таким образом, алгоритм редукции следующий:

1. Каждый блок выполняет редукцию независимо.
2. Барьер памяти. Нужен для того, чтобы всем потокам были доступны все изменения, сделанные другими потоками.
3. Нулевой поток каждого блока атомарно увеличивает значение глобальной переменной retirementCount.
4. Нулевой поток каждого блока определяет, является ли его блок последним, посредством сравнения retirementCount с числом блоков.
5. Барьер исполнения внутри каждого блока.
6. Последний блок выполняет второй этап редукции.

Результаты численного эксперимента. Для того чтобы продемонстрировать преимущество интегрирования системы на GPU, приведем результаты численного эксперимента. Рассмотрим интегрирование систем с 2048, 4096 и 8192 генераторами, что соответствует 6144, 12288 и 24576 уравнениям. Сравнивать будем три реализации: однопоточную реализацию на CPU (CPU-ST), многопоточную реализацию на CPU (CPU-MT) и реализацию на GPU. В качестве тестового стенда использовался персональный компьютер с процессором Intel Core i5 750@2.66 GHz, имеющий 4 вычислительных ядра, а также видеокартой NVIDIA GeForce GTX 580, имеющей 512 потоковых процессоров (табл. 2).

Таблица 2. **Результаты численного эксперимента**

N	*T*1, CPU-ST	*T*2, CPU-MT	*T*2/*T*1	*T*3, GPU	*T*3/*T*2
2048	19.14	8.67	2.21	0.54	16.06
4096	38.96	13.90	2.8	0.61	22.79
8192	77.85	25.46	3.1	0.75	33.95

Заключение

Таким образом, мы показали, что для решения системы с большим количеством уравнений проекционным методом Эйлера графические процессоры общего назначения являются более эффективными, чем центральный процессор. Получено ускорение более чем в 10 раз, по сравнению с центральным процессором, что позволяет рекомендовать использование графических процессоров общего назначения для решения подобных задач.

Список литературы

1. **Чадов С.Н.** Интегрирование жесткой модели электромеханической системы явными методами // Вестник ИГЭУ. – 2012. – № 2. – С. 73–75.
2. **Чадов С.Н.** Численное исследование модели энергетической системы // Вестник ИГЭУ. – 2009. – № 4. – С. 49–52.
3. **Gear C.W., Kevrekidis I.G.** Projective Methods for Differential Equations // SIAM J. Sci. Comp. – 2003. – 24(4). – P. 1091–1106.
4. **Harris M.** Optimizing parallel reduction in CUDA. NVIDIA Dev. Tech. (2007) [Электронный ресурс]. – Режим доступа: http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

References

1. Chadov, S.N. Integrirovanie zhestkoy modeli elektromekhanicheskoy sistemy yavnymi metodami [Integrating a stiff electromechanical system by explicit methods]. *Vestnik IGEU*, 2012, issue 2, pp. 73–75.
2. Chadov, S.N. Chislennoe issledovanie modeli energeticheskoy sistemy [Numerical study of a model power system]. *Vestnik IGEU*, 2009, issue 4, pp. 49–52.
3. Gear, C.W., Kevrekidis, I.G. Projective Methods for Differential Equations, NECI Technical Report NECI-TR 2001-029, Apr. 2001, also SIAM J. Sci. Comp., 2003, 24(4), pp. 1091–1106.
4. Harris, M. Optimizing parallel reduction in CUDA. NVIDIA Dev. Tech. (2007). Available at: http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

Чадов Сергей Николаевич,
ФГБОУВПО «Ивановский государственный энергетический университет имени В.И. Ленина»,
аспирант кафедры высокопроизводительных вычислительных систем,
e-mail: sergei.chadov@gmail.com