УДК 004.272

ПРИЙМАК А.В.

ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ НА CUDA ПРИ МОДЕЛИРОВАНИИ НЕУСТОЙЧИВОСТИ ЛЕНГМЮРОВСКИХ ВОЛН В ПЛАЗМЕ

Рассмотрена оптимизация вычислений при использовании технологии параллельного программирования на графических процессорах CUDA для моделирования параметрической неустойчивости ленгмюровских волн в плазме. Определены затраты времени на различные этапы численного моделирования. Проанализирована эффективность применения различных способов оптимизации вычислений. Показано, что для данной задачи наибольший прирост скорости вычислений можно получить при оптимизации использования тригонометрических функций.

Optimization of parallel computing on graphic processors with CUDA was considered for simulation of parametric instability of Langmuir waves in plasma. The time required for different stages of the numerical simulation have been defined. The success of different optimization methods was analyzed. It is shown that the greatest increase in computational speed can be obtained by optimizing the use of trigonometric functions.

1. Введение

В настоящее время растет популярность использования графических процессоров (graphics processing unit, GPU) для ускорения численных экспериментов, т.к. GPU имеют производительность параллельной архитектуры [1]. Такой подход использования GPU для вычислений общего назначения, которые обычно проводит центральный процессор (central processing unit, CPU), известен как general-purpose computing on graphics processing units (GPGPU) [2].

Изучение гибридных моделей параметрической неустойчивости ленгмюровских волн в плазме требует проведения большого количества численных экспериментов и, соответственно, значительных затрат времени. Затраты времени зависят от количества ионов в модели и от других параметров, и составляют обычно несколько часов на один численный эксперимент. Интерес к параметрической неустойчивости ленгмюровских волн обусловлен возможностями нагрева плазмы в установках термоядерного синтеза. Аппарат описания параметрической неустойчивости был создан в работах В.П. Силина [3,4] и В.Е. Захарова [5]. В настоящей работе модели параметрической неустойчивости ленгмюровских волн в плазме относятся к малоизученным гибридным, когда электроны описаны гидродинамически, а ионы описаны кинетически, крупными частицами.

Для проведения численных экспериментов выбрана compute unified device architecture (CUDA) - архитектура параллельных вычислений, разработанная компанией NVIDIA, и одна из основных GPGPU технологий [6].

При использовании технологии CUDA важную роль играет оптимизация, благодаря которой скорость вычислений можно увеличить в несколько раз [6-8].

2. Постановка задачи

При разработке алгоритма и программного обеспечения для проведения численных экспериментов необходимо учесть основные рекомендации по оптимизации вычислений, детально описанные в [6-8]. Необходимо оценить возможность переноса вычислений на GPU и возможность их распараллеливания, минимизировать обмен данными между памятью GPU и оперативной памятью (ОЗУ), выбрать правильные параметры распараллеливания и использовать для вычислений быстрые виды памяти GPU.

Необходимо оценить затраты времени на различные этапы моделирования и при оптимизации уделить внимание наиболее продолжительным этапам. При этом проведение оптимизации не должно сильно влиять на точность вычислений (не более чем на 1-2%) и не должно усложнять код программы, т.к. изучение гибридных моделей параметрической неустойчивости ленгмюровских волн требует периодического изменения некоторых параметров и уравнений.

Гибридные модели Силина и Захарова, результаты моделирования которых приведены в [9], состоят из уравнений амплитуды e_n, фазы ψ_n, реальной и мнимой частей плотности ионов M_{nr} и M_{ni}, напряженности электрического по-

ля $\bar{E}_{nr}$ и $\bar{E}_{ni}$ (число уравнений каждого типа равно числу мод, т.е. 400) и уравнений движения ионов с координатами и скоростями (ξ_s, v_S), число которых равно 20000. Также имеются два уравнения для описания волны накачки, амплитуда и фаза которой a_0, ϕ. Математические модели представлены в [10]. Динамика процессов параметрической неустойчивости в гибридных моделях Силина и Захарова подобна вследствие подобия систем уравнений [10].

Численный эксперимент представляет собой решение задачи Коши методом Эйлера. Начальные условия и параметры численных расчетов взяты следующие. Количество крупных частиц, моделирующих ионы $0 < s \le S = 20000$, число мод спектра $-N < n < N$, $N = S/100$, $a_0(0) = 0.06$, $d\xi_s / d\tau|_{\tau=0} = v_s|_{\tau=0} = 0$. Крупные частицы, моделирующие ионы, равномерно распределены на интервале $-1/2 < \xi < 1/2$. Начальная амплитуда ВЧ моды задается формулой $e_n|_{\tau=0} = e_{n0} = (2+g)\cdot 10^{-3}$ в модели Силина и формулой $e_n|_{\tau=0} = e_{n0} = (0.5+g)\cdot 10^{-4}$ в модели Захарова, где $g \in [0;1]$ - случайное число, $\psi_n|_{\tau=0}$ случайным образом распределялись в интервале $0 \div 2\pi$. Параметры легких ионов в модели Силина: $\frac{m_e}{M}\cdot\frac{\omega^2{}_p}{\delta^2} = 0.43$, $\frac{\delta}{\omega_0} = 0.034$, $\frac{\omega_0}{\delta} = 29.4$. Параметры легких ионов в модели Захарова: $\frac{m_e}{M}\cdot\frac{\omega_p^2}{\delta^2} = 20$, $\frac{\delta}{\omega_0} = 3.5\cdot 10^{-3}$, $\frac{\omega_0}{\delta} = 282.6$. Выбранные начальные условия отвечают равновесному состоянию электронов и ионов плазмы.

3. Реализация уравнений моделей с помощью технологии JCUDA

Программа, реализующая математическую модель задачи, создана с использованием технологии java compute unified device architecture (JCUDA). JCUDA обеспечивает взаимодействие с CUDA из Java-программы [11]. Вычисления производятся параллельно на GPU GeForce GT630 (GV-N630D3-2GL). Управление выполнением кода на GPU задается в Java-программе как в [12,13]. Код для выполнения на GPU пишется в виде CUDA модулей (.cu) без применения специализированных библиотек на основе CUDA API. При выполнении программы расчет всех уравнений моделей происходит на GPU. Алгоритм вычислений на GPU детально описан в [13].

Все вычисления выполняются в двойной точности (double), т.к. проведенные вычисления в одинарной точности (float) меняют динамику процесса.

На CUDA явно написан код, рассчитывающий значения функций Бесселя, т.к. встроенные в CUDA функции Бесселя дают недостаточную точность [1], близкую к одинарной. Другие используемые математические функции имеют двойную точность вычислений с малой погрешностью.

Наряду с параллельными вычислениями на GPU проведены непараллельные вычисления на CPU, которые подтвердили высокую скорость и точность вычислений на GPU [13]. Установлено, что скорость вычислений на GPU в 37 раз (модель Силина) и в 35 раз (модель Захарова) выше, чем на одном ядре CPU AMD Athlon 64 X2 с частотой ядра 2.2ГГц. Результаты вычислений на GPU и CPU отличаются на 1.5% в модели Силина и на 0% в модели Захарова. Сравнение результатов проводилось по распределению ионов $S(\xi)$ в момент времени, когда энергия ионов, пропорциональная величине $\sum_s (d\xi_s/dt)^2$ уже не возрастает. Отличия между случаями зависят от момента времени, с течением времени отличия возрастают.

4. Применение основных способов оптимизации вычислений на CUDA

4.1. Минимизация обмена данными между CPU и GPU

Для лучшей производительности приложений важно свести к минимуму передачу данных между памятью GPU и ОЗУ, даже если это означает запуск участков кода на GPU, которые не демонстрируют ускорение по сравнению с запуском на CPU [8]. Так, вычисление параметров $\partial a_0, \partial\phi$ происходит на GPU непараллельно на одном ядре, т.к. они описываются только двумя уравнениями. Вычисление $\partial a_0, \partial\phi$ на CPU происходило бы быстрее, однако требовало бы при вычислении для каждого момента времени передачи большинства массивов в ОЗУ, что привело бы к затратам времени на обмен данными. Как будет показано дальше, расчет $\partial a_0, \partial\phi$ занимает незначительное время,

поэтому не является узким местом в производительности.

Исходные данные для расчета загружаются в память GPU единожды перед началом моделирования. Далее обмен данными между CPU и GPU отсутствует кроме тех моментов времени, результаты которых отображаются графически. Дополнительным преимуществом такого подхода является то, что часть данных существует только в памяти GPU и не дублируется в ОЗУ для экономии памяти. Дублирование данных можно будет избежать, используя унифицированную виртуальную память, поддержка которой будет добавлена в CUDA Toolkit 6.0.

4.2. Максимальное распараллеливание вычислений

Вычисления для всех типов уравнений, кроме расчета параметров $\partial a_0, \partial\phi$, происходят параллельно. Это означает, что для вычисления одного уравнения создается один поток (thread). Выполнение потока представляет собой вызов функции, на вход которой подаются необходимые для вычисления данные, а на выходе возвращается результат вычисления. Функция имеет общий вид для всех уравнений одного типа.

4.3. Определение оптимальных размеров сетки потоков

Модель программирования в CUDA предполагает группирование потоков. Потоки объединяются в блоки потоков (thread block). Программа (ядро, kernel) исполняется над сеткой (grid) блоков потоков.

Для уравнений амплитуды ∂e_n, фазы $\partial\psi_n$, плотности ионов M_{nr} и M_{ni}, напряженности электрического поля $\bar{E}_{nr}$ и $\bar{E}_{ni}$ и функций Бесселя J_0, J_1, J_n расчеты можно выполнять с одинаковыми параметрами распараллеливания. Для расчета этих уравнений создается сетка, состоящая из одного блока потоков: размерность сетки $\dim3(1,1,1)$ и размерность блока $\dim3(N,1,1)$. Такой алгоритм поддерживает число мод до 1024, т.к. CUDA имеет ограничение до 1024 потоков в одном блоке. При проведении вычислений отдельно для отрицательных и отдельно для положительных мод, т.е. создании двух сеток размером $N/2$, скорость расчета уравнений падает на 38%.

Для расчета параметров $\partial a_0, \partial\phi$ задаются следующие параметры распараллеливания: размерность сетки $\dim3(1,1,1)$ и размерность блока $\dim3(1,1,1)$, что означает создание всего одного потока, т.е. непараллельное вычисление.

Для расчета уравнений движения ионов dv_s и $d\xi_s$, а также поля ВЧ давления на ионы $\bar{E}\xi_s$ выбрана размерность сетки $\dim3(S/500,1,1)$ и размерность блока $\dim3(500,1,1)$. При числе потоков в блоке меньше 500 скорость вычисления уравнений падает на несколько процентов.

4.4. Предварительный расчет функций Бесселя в гибридной модели Силина

Как было сказано выше, расчет функций Бесселя происходит явно для поддержания высокой точности вычислений. При этом для каждого момента времени для каждой моды необходимо вычислить по одной функции Бесселя J_0, J_1, J_n. Поэтому перед началом расчета уравнений для каждого момента времени происходит предварительный расчет функций Бесселя, и далее полученные значения используются в уравнениях на протяжении счета для данного момента времени.

4.5. Оптимизация использования памяти

Все загружаемые в память GPU данные хранятся в глобальной памяти, размер которой в используемом GPU составляет 2Гбайт. Однако глобальная память обладает низкой скоростью доступа [7]. GPU включает и другие типы памяти, которые имеют небольшой объем и высокую скорость доступа, в первую очередь разделяемую (shared) память и константную (constant) памятью. Для использования разделяемой памяти необходимо проводить дополнительное копирование данных, но даже при этом скорость вычислений должна повыситься. Однако попытки использовать разделяемую и константную память дали эффект около 1%. В разделяемую память были положены несколько массивов, например массивы со значениями функций Бесселя. В константную память были положены постоянные значения, например число π и другие константы.

4.6. Уменьшение количества вычислений

В некоторых местах программного кода многократно выполняются одинаковые вычис-

ления, поэтому применен способ для сокращения числа вычислений. Например, при расчете dv_S присутствует повторяющееся в цикле вычисление $2\cdot\pi\cdot\xi_S$, поэтому была создана переменная $p = 2\cdot\pi\cdot\xi_S$, которая затем использовалась в цикле. Подобным образом были сделаны еще несколько оптимизаций, однако их суммарный эффект не превышает 1% и при этом усложняет программный код.

5. Определение затрат времени на различные этапы алгоритма

После оптимизации проводится определение затрат времени на выполнение разных этапов алгоритма (табл. 1). Измерение времени происходит в Java-программе с помощью метода System.currentTimeMillis().

Табл. 1 – Затраты времени на выполнение этапов алгоритма после оптимизации

	Время выполнения, минут	
	Модель Силина	Модель Захарова
Этап 1. Расчет дополнительных параметров для построения графиков и их рисование (CPU)	2.7	2.5
Этап 2. Расчет J_0, J_1, J_n, ∂e_n, $\partial\psi_n$, M_{nr}, M_{ni}, $\bar{\mathrm{E}}_{nr}$, $\bar{\mathrm{E}}_{ni}$ (GPU)	12.3	1.3
Этап 3. Расчет dv_S, $d\xi_S$ (GPU)	5	0.6
Этап 4. Расчет накачки $\partial a_0, \partial\phi$ непараллельно (GPU)	0.7	0.1
Этап 5. Обновление значений параметров с учетом посчитанных на данном шаге (GPU)	0.4	0.1
Этап 6. Получение результатов из GPU (GPU→CPU)	0.01	0.01
Этап 7. Расчет $\bar{E}\xi_S$ (GPU)	0.1	0.1
Общее время выполнения на GPU	18.5 (87.3% общего времени)	2.2 (47% общего времени)
Общее время выполнения на CPU	2.7 (12.7% общего времени)	2.5 (53% общего времени)
Общее время выполнения моделирования	21.2	4.7

Определение затрат времени показывает успешность минимизации обмена данными между GPU и CPU, а также малые затраты времени на расчет накачки, который проводится непараллельно.

Недостатком является высокая доля (53%) времени выполнения на CPU в модели Захарова при том, что расчет всех уравнений был перенесен на GPU. Однако на CPU происходит подготовка и рисование графиков и интерфейса (200 раз в течении численного эксперимента). Меньшая доля времени выполнения на CPU в модели Силина объясняется большим числом вычислений на GPU, а именно тем, что перед получением результатов из GPU происходит расчет для 40 моментов времени, в то время как у Захарова для 5 моментов времени. Скорость вычислений для одного момента времени в моделях Силина и Захарова практически одинакова.

Значительная доля времени моделирования приходится на 2-й и 3-й этапы, т.е. непосредственно на вычисление уравнений. При этом определено, что 85% этого времени занимает вычисление функций синусов и косинусов.

6. Оптимизация использования тригонометрических функций

Т.к. определено, что затраты на вычисление синусов и косинусов составляют 85% от общего времени вычисления уравнений, были опробованы различные способы оптимизации использования тригонометрических функций.

6.1. Использование альтернативных тригонометрических функций

Существуют функции $\sin pi(\mathrm{x}) = \sin(\pi\cdot\mathrm{x})$, $\cos pi(\mathrm{x}) = \cos(\pi\cdot\mathrm{x})$, однако в моделях Силина и Захарова только девятая часть функций имеет такой вид, поэтому эффект от их использования составляет лишь до 2%.

Также существует функция $\sin\cos(\mathrm{x})$, одновременно рассчитывающая синус и косинус. Эффект от ее использования составляет около

50%. Однако существенным недостатком функции является необходимость для каждого вычисления выделять память под хранение значений синуса и косинуса, что усложняет ее использование.

6.2. Создание таблицы значений тригонометрических функций

Сделана попытка делать предварительный расчет синусов и косинусов в каждый момент времени как это делается с функциями Бесселя, а затем использовать посчитанные значения в уравнениях. Четверть функций имеют подобную себе функцию с одинаковым аргументом, однако применение этого способа оптимизации дает эффект до 5% и усложняет код программы.

6.3. Понижение точности вычислений синуса и косинуса

В CUDA каждая математическая функция имеет несколько реализаций, отличающихся по точности результата вычисления – double-функции (например $\sin(x)$, $\cos(x)$), float-функции (например $\text{sinf}(x)$, $\text{cosf}(x)$), функции пониженной точности (например $\_\_\text{sinf}(x)$, $\_\_\text{cosf}(x)$). Таблицы функций различной точности приведены в [1].

Т.к. основные затраты времени идут на вычисление тригонометрических double-функций, сделана попытка использовать аналоги этих функций меньшей точности. При этом остальные вычисления продолжают проводиться в двойной точности.

В отличие от CPU, GPU не оптимизированы для вычислений двойной точности. В связи с этим, скорость вычислений на GPU в двойной точности меньше, чем в одинарной.

Так, применение float-функций позволяет ускорить вычисление уравнений на 60%. После окончания моделирования была проверена точность вычислений и отклонения результатов составили 1.2% от случая с использованием double-функций синуса и косинуса.

Применение функций синуса и косинуса пониженной точности позволяет ускорить вычисление уравнений на 70%. Отклонения результатов составили 1.7% от случая с использованием double-функции синуса и косинуса.

Кроме того, возможно комбинирование разных способов оптимизации тригонометрических функций. Однако этого не было выполнено в настоящей работе, т.к. основной эффект достигается только благодаря понижению точности вычислений синуса и косинуса.

7. Выводы

Важность оптимизации вычислений на CUDA подтверждена в настоящей работе. В то же время основные способы оптимизации показали разные результаты. Это говорит о том, что эффективность различных способов оптимизации зависит от специфики задачи.

В работе описаны результаты применения основных способов оптимизации CUDA-вычислений. Наибольший прирост скорости вычислений дает использование функций синуса и косинуса пониженной точности.

Дальнейшая оптимизация параллельных вычислений на CUDA при моделировании параметрической неустойчивости ленгмюровских волн в плазме возможна при использовании новых технологий, связанных с CUDA, динамического параллелизма и унифицированной виртуальной памяти.

Автор выражает благодарность Куклину В.М. за внимание к работе.

Список литературы

1. CUDA C Programming Guide. - NVIDIA Corporation [Электронный ресурс]. - Режим доступа: http://docs.nvidia.com/cuda/cuda-c-programming-guide. - Электрон. версия, 2013. – HTML формат.
2. CUDA. - Wikipedia [Электронный ресурс]. - Режим доступа: http://en.wikipedia.org/wiki/CUDA. - Электрон. версия, 2013. – HTML формат.
3. Silin V.P. Parametric resonance in plasma./ JETP. -1965. – Vol. 48. N6. – P. 1679-1691.
4. Silin V.P. Parametric Influences of high-energy Radiation on Plasma. – Moscow: Nauka, 1973.
5. Zakharov V.E. Weak-turbulence spectrum in a plasma without a magnetic field //Sov. Phys. JETP. –1967. – Vol. 24(2), P.455-459. The Instability of Waves in Nonlinear Dispersive Media // Sov. Phys. JETP. -1967. – Vol. 24. – P.740. Collapse of Langmuir Waves // Sov. Phys. JETP. -1972. –Vol. 35(5). –P.908-914.
6. Д. Сандерс, Э. Кэндрот. Технология CUDA в примерах. Введение в програмирование графических процессов: Пер. с англ. Слинкина А.А., научный редактор Боресков А.В. М.: ДМК Пресс, 2011. - 232 с.

7. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. - 232с.
8. CUDA C Best Practices Guide. - NVIDIA Corporation [Электронный ресурс]. - Режим доступа: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide. - Электрон. версия, 2013. – HTML формат.
9. Belkin E.V., Kirichok A.V., Kuklin V.M., Pryjmak A.V., Zagorodny A.G. Dynamics of ions during development of parametric instability of Langmuir waves // Вопросы атомной науки и техники. Серия «Плазменная электроника и новые методы ускорения». 2013, №8, с.260-266.
10. Kuklin V.M. Similarity of 1D Parametric Instability description of Langmuir waves. / The Journal of Kharkiv National University, physical series: Nuclei, Particles, Fields. – 2013. - №1041. - Iss. 2 (58). - P.20-32.
11. Marco Hutter. JCUDA. - jcuda.org, 2008 [Электронный ресурс]. - Режим доступа: http://jcuda.org/. - Электрон. версия, 2008. - HTML формат.
12. Мишин А.В., Приймак А.В. Моделирование неустойчивости движущегося в плазме сгустка заряженных частиц. // Вестник Харьковского национального университета, – 2012. – № 1037. Сер. "Математическое моделирование. Информационные технологии. Автоматизированные системы управления", вып. 20. – С. 133-145.
13. Приймак А.В. Использование технологии JCUDA для моделирования динамики ионов при развитии параметрической неустойчивости ленгмюровских волн. – Информатика и компьютерные технологии / Сборник трудов IX международной научно-технической конференции студентов, аспирантов и молодых ученых. – 5-6 ноября 2013, Донецк, ДонНТУ. – 2013. – С. 200-204.