ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧИ О ДВИЖЕНИИ ЖИДКОСТИ В КАВЕРНЕ НА КОМПЬЮТЕРАХ С ПАРАЛЛЕЛЬНОЙ АРХИТЕКТУРОЙ

Д.В. Деги

Целью данной работы является создание эффективных параллельных алгоритмов для решения уравнений Навье-Стокса в различных моделях параллельного программирования (OpenMP, MPI, CUDA).

Рассматривается стационарное, изотермическое движение вязкой, несжимаемой жидкости в квадратной каверне с движущейся верхней стенкой со скоростью U. Значение плотности и коэффицента вязкости постоянны.

Данный процесс описывается уравнениями движения Навье-Стокса и уравнения неразрывности:

$$\frac{\partial v_x v_x}{\partial x}+\frac{\partial v_y v_x}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 v_x}{\partial x^2}+\frac{\partial^2 v_x}{\partial y^2}\right);$$

$$\frac{\partial v_x v_y}{\partial x}+\frac{\partial v_y v_y}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v_y}{\partial x^2}+\frac{\partial^2 v_y}{\partial y^2}\right);$$

$$\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}=0.$$

Здесь Vx=Vx(x,y) и Vy=Vy(x,y) - соответственно горизонтальная и вертикальная компоненты вектора скорости движения жидкости, p=p(x,y) - давление.

Граничные условия следующие:

x=0: Vx=0, Vy=0;
x=Lx: Vx=0, Vy=0;
y=0: Vy=0, Vx=0;
x=Ly: Vy=0, Vx=U.

Где Lx и Ly – соответственно длина и ширина каверны.

Решение поставленной задачи ищется численно с помощью метода конечных объемов [1]. Для нахождения решения используется равномерная шахматная сетка (компоненты скорости определяются на гранях конечных объемов, а давление – в центре). Конвективные члены аппроксимируются с помощью схемы против потока, диффузионные – центрально-разностной. Погрешность аппроксимации разностных схем уравнений движения имеет первый порядок по каждой координате, а уравнения неразрывности – второй. Полученные разностные схемы представляются в виде:

$$aP_{i+1\,j}(v_x)_{i+1\,j}=\sum_{nb}a_{nb}(v_x)_{nb}+d_{i+1\,j}(p_{ij}-p_{i+1\,j}),\ i=\overline{0,Nx-1},\ j=\overline{0,Ny};$$

$$bP_{i\,j+1}(v_y)_{i\,j+1}=\sum_{nb}b_{nb}(v_y)_{nb}+l_{i\,j+1}(p_{ij}-p_{i\,j+1}),\ i=\overline{0,Nx},\ j=\overline{0,Ny-1};$$

$$[(v_x)_{i+1\,j}-(v_x)_{ij}]hy+[(v_y)_{i\,j+1}-(v_y)_{ij}]hx=0,\ i=\overline{0,Nx},\ j=\overline{0,Ny}.$$

Здесь nb означает суммирование по соседним узлам сеточного шаблона. Из-за неллинейности уравнений Навье-Стокса невозможно получить прямое решение системы и рассматриваемый процесс приобретает итерационный характер. Так коэффициенты anb и bnb – переменные, зависящие от компонент скорости полученных на предыдущей итерации.

Для совместного решения из трех разностных уравнений использован алгоритм SIMPLE С. Патанкара

Точность вычислений составляет 0.001. Расчет неизвестных проводился с одинарной точностью. В качестве критерия завершения итерационного процесса выбрано выполнение условий: норма поправки давления меньше либо равно заданой точности, а также выполнения дискретного аналога уравнения неразрывности.

Основные вычислительные трудности данного алгоритма - это решение уравнений для поправки скоростей (1), а также для поправки давления (2) на каждой итерации. Уравнения (1) решались методом нижней релаксации с коэффициентом релаксации равным – 0.2, а у уравнение (2) – верхней релаксацией с коэффициентом – 1.85. Сходимость данных методов следует из диагонального преобладания матриц систем уравнений. Для организации вычислений на многопроцессорных ЭВМ использовался метод ‹‹красно-черного›› упорядочивания.

В качестве проверки правильности работы были проведены сравнения результатов расчетов с результатами расчетов [5]. Результаты расчетов представлены на сетке размера 256*256 при числе Рейнольдса Re=1000. На графике 1 показан профиль продольной скорости в среднем поперечном сечении x=Lx/2 для квадратной каверны, получаемый в результате работы различных программ и построен график поперечной скорости в среднем продольном сечении y=Ly/2.

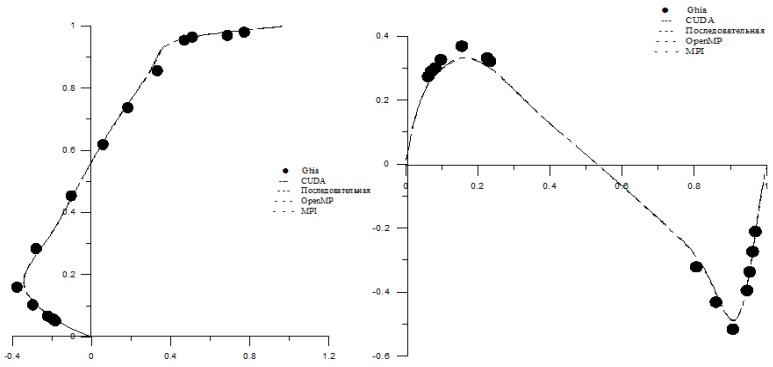

График 1 - Графики продольной и поперечной скоростей при x=Lx/2 и y=Ly/2 соответственно

Рассматриваемая задача решалась на следующих многопроцессорных ЭВМ: систем с общей памятью (с помощью системы OpenMP), систем с распределенной памятью (с помощью библиотеки MPI), графических процессоров (с помощью технологии CUDA). В качестве системы с распределенной памятью использован кластер ТГУ СКИФ Cyberia. Один вычислительный узел кластера рассматривался как система с общей памятью. Вычислительный узел имеет следующие характеристики: 4 Гб оперативной памяти, два двухъядерных процессора Intel Xeon 5150 Woodcrest с тактовой частотой 2,66 ГГц. В качестве графических процессоров использовалась видеокарта GTX 260. Данная видеокарта состоит из 192 потоковых ядер с тактовой частотой 1,24 ГГц. Объем памяти составляет – 2 Гб.

Перед написанием параллельной версии программы нужно убедиться, что в данной программе учтено влияние факторов, увеличивающих ее быстродействие. К числу таких факторов относится ,например, влияние оперативной памяти (ОЗУ), учет количества арифметических операций, минимизация операций, требующих больше тактов работы процессора (например деления). При утете этих условий получается следующее время работы программы (написанной на языке программирования С) в зависимости от размера используемой сетки:

Таблица 1 - Время работы программы при использовании сеток различного размера

Сетка	Итераций	Время (сек.)
128×128	1004	21
256×256	3176	294
512×512	12160	5037
1024×1024	53713	107655

Из таблицы 1 видно, что при увеличении каждого размера сетки в 2 раза, число итераций увеличивается более чем в 2 раза, возрастает объем вычислений и время работы программы увеличивается более чем в 10 раз и время счета переходит из секунд в из минут в часы, а из часов в сутки. Тем самым становятся актуальным вопросы уменьшения времени работы за счет использования ЭВМ многоядерной архитектуры.

Рассмотрим ускорение получаемое с помощью системы OpenMP. Система OpenMP проста в использовании. Для написания параллельной версии программы достаточно в последовательную программу вставить соответствующие директивы для распараллеливания. Если ресурс параллелизма мал, то есть мало участков, которые могут выполняться независимо друг от друга, то можно использовать конструкции, задающие конечный параллелизм, такие как независимые секции (#pragma omp section) или низкоуровневое распараллеливание по номеру нити [2]. Но как показывает практика, основной ресурс параллелизма расположен в циклах, итерации которого можно просто распределить по процессам с помощью директивы #pragma omp for. Рассматриваемая последовательная программа с точки зрения распараллеливания в системе OpenMP состоит в основном из следующих друг за другом циклов, тем самым упрощается сам процесс распараллеливания. Те циклы, которые в данной программе на каждой итерации занимают более 20% процессорного времени выполнения всей итерации, распараллеливаются с помощью конструкции #pragma omp parallel for. То есть для

данного цикла создается параллельная область и сразу начинается его распараллеливание, после чего параллельная область заканчивается. Для оставшихся циклов (коррекция скорости, давления, проверка критерия завершения итерационного процесса) сначала создается параллельная область, и в ней уже идет распараллеливание циклов. По объему используемого ручного кода OpenMP – программа практически совпадает с последовательной (больше на 12 строк). В таблице 2 показано ускорение программы, получающееся при использовании системы OpenMP, на сетках различной длины.

Таблица 2 - Ускорение OpenMP - программы

Количество вычислительных ядер	Ускорение при использовании сеток различной длины			
	128×128	256×256	512×512	1024×1024
2	1.6	1.7	1.8	1.5
4	2.9	3.2	3.4	2.2

Из таблицы 2 видно, что наиболее лучшее ускорение получается на сетке 512?512. Среднее ускорение (относительно размеров сетки) получаемое при использовании 2 ядер составляет 1.7, 4 – 2.9.

Для решения задач на системах с распределенной памятью широко применяется стандарт MPI. Библиотека MPI представляет собой достаточно большой набор функций, с помощью которых выполняется различные действия такие как обмен значений между процессами, создание декартовой топологии, объявления производных типов данных [3]. Так для рассматриваемой задачи применялись:

1. функции совмещающие пересылки и прием элементов (MPI_Sendrecv);
2. для удобства передачи и приема целого набора чисел объявлялись специальные типы (MPI_Type_Vector), с помощью которых передача или прием осуществлялся как с одним значением;
3. распараллеливание рассматривалось по принципу геометрической декомпозиции (использовалась двумерная декомпозиция данных), для этого вся совокупность загружаемых процессоров разбивалась с помощью функции создания декартовой топологии (MPI_Cart_create).

Все массивы для хранения используемых значений объявляются внутри области создания параллельной программы. Таким образом каждый процессор не будет выделять на доступной ему ОЗУ место для хранения всех вычисляемых величин. Выделение памяти имеет место только для тех величин, которые будут использоваться в вычислениях на данном процессоре. Это позволяет запускать задачи с большим объемом данных (необходимо, чтобы объем используемых значений процессором не превосходил объема доступной ему ОЗУ, в данном случае – 4 Гб). В таблице 3 показано ускорение программы, получающееся при использовании библиотеки MPI, на сетках различной длины.

Таблица 3 - Ускорение MPI – программы

Кол-во вычислительных ядер	Ускорение при использовании сеток различной длины			
	128×128	256×256	512×512	1024×1024
2	1.8	1.9	1.9	1.7
4	3.3	4	2.6	2.1
8	4.8	7.1	8.2	4.3
16	6.6	11.8	16.6	11.2
32	7.5	18.4	29	32.2
64	9.1	24.5	46.6	74

Согласно таблице 3 следует, что при увеличении количества вычислительных ядер ускорение увеличивается, и также получается эффективное расспараллеливание. Там где ускорение больше числа используемых ядер (например на сетке 1024?1024 при использовании 64 ядер) наблюдается эффект сверхлинейного ускорения. Это объясняется наличием у каждого процессора своей кэш-памяти. Ускорение, получаемое при использовании большого количества вычислительных ядер (32 и 64), с возрастанием размеров сетки возрастает. Так на сетке 1024?1024 с использованием 32 ядер ускорение составляет – 32.2, 64 – 74.

Среднее ускорение (относительно размеров сетки) получаемое при использовании 2 ядер составляет 1.8, 4 – 3, 8 – 6.1, 16 – 11.6, 32 – 21.8, 64 – 38.6. Среднее ускорение (относительно размеров сетки) получаемое при использовании системы OpenMP практически совпадает с укорением получаемым при работе MPI–программы. Так при использовании 2 ядер ускорение OpenMP и MPI-программ равняются 1.7 и 1.8 соответственно, а при 4 – 2.9 и 3.0. Но по объему ручного кода MPI – программа больше и превосходит последовательную в 1.6 раза.

Модель программирования CUDA предполагает группирование потоков [4]. Так как рассматриваемый физический процесс двумерный, то использовалась двумерная сетка блоков, состоящая из двумерного блока потоков. При организации вычислений на видеокарте является важным работа с глобальной памятью, также как и для современных процессоров являютя важным «кэш-механизмы». Оптимизация работы с глобальной памятью включает в себя следующее: использование общей памяти (размером в 16 Кб, расположенные на каждом мультипроцессоре), объединение запросов на запись и считывание из глобальной памяти. При построении параллельного алгоритма применялись следующие моменты по оптимизации работы с памятью:

1. Для уменьшения повторных запросов из мультипроцессоров в глобальную память, данные сначала копировались в их общую память, а уже затем использовались (с учетом конфликтов банков).
2. С целью минимизации обменов данных между ОЗУ и глобальной памятью GPU, все вычисления происходили в видеокарте. После завершения каждой итерации из GPU в ОЗУ копировалось всего два значения для проверки критерия завершения итерационного процесса центральным процессором.
3. Использование блоков, размер которых кратен числу 16, для объединения запросов в глобальную память.

В таблице 4 показано ускорение CUDA – программы.

Таблица 4 - Ускорение CUDA программы

Сетка	Кол-во итераций последовательной программы	Кол-во итераций CUDA программы	Ускорение
128×128	1004	1005	12.4
256×256	3176	3176	22.6
512×512	12160	10324	37
1024×1024	53713	44998	48

Как видно, из таблицы 4 с увеличением размера сетки ускорение возрастает. На сетки 1024?1024 ускорение уже достигает 48. Из-за того, что использовались различные компиляторы для последовательной и CUDA - программы, количество их итераций различное. Итерационный процесс в CUDA - программе сошелся быстрее (на каждой сетке). Среднее ускорение (относительно размеров сетки) равно - 30. Представленные в таблице 4 результаты говорят о применимости GPU для решения задач гидродинамики. По объему ручного кода CUDA - программа оказалась самой длинной, больше последовательной в 2.6 раза. Организация вычислений на видеокартах хоть и требует дополнительных усилий, зато время вычислений значительно сокращается по сравнению с центральным процессорам. Учитывая относительно недорогую стоимость видеокарт, перспективным будет построение кластеров на основе видеокарт (как можно более производительных).

ЛИТЕРАТУРА:

1. Патанкар С - Численные методы решения задач теплообмена и динамики жидкости/ С. Патанкар. – М.: Энергоатомиздат, 1984, 124с.
2. Левин М.П. - Параллельное программирование с использованием системы OpenMP./ М.П. Левин. – М.: БИНОМ, 2008, 118с.
3. Гергель В.П. – Теория и практика параллельных вычислений/ В.П. Гергель. – Бином. Лаборатория знаний, 2007. – 424с.
4. Беликов Д.А. - Высокопроизводительные вычисления на кластерах: Учебн. пособие/ Д.А Беликов, И.В. Говязов и др.; Под ред. А. В. Старченко. - Томск: Изд-во Том. ун-та, 2008. - 198с.
5. Ghia - High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method / U., K.N. Ghia and C.T. Shin. - J. Comp. Phys., 1982. – Vol.48, P. 387--411.