Численное моделирование распространения длинных волн в больших акваториях с помощью SMP-узловых кластеров*

Е.В. Дементьева
Институт математики Сибирского федерального университета
e-mail: <lionesskate@gmail.com>

Е.Д. Карепова, А.В. Малышев
Учреждение Российской академии наук
Институт вычислительного моделирования Сибирского отделения РАН
e-mail: e.d.karepova@icm.krasn.ru
e-mail: amal@icm.krasn.ru

В работе обсуждаются некоторые аспекты эффективного использования кластерных систем на примере реализации метода конечных элементов для начально-краевой задачи для уравнений мелкой воды.

1. Введение

В процессе разработки программного обеспечения (ПО) для решения начально-краевой задачи для уравнений мелкой воды возникли затруднения, обусловленные недостатком информации об эффективности того или иного способа решения частных подзадач. В связи с этим было проведено исследование, результаты которого относятся не столько к самой задаче, сколько к инструментам, с помощью которых она решается. В частности, была сопоставлена эффективность двух широко распространенных реализаций стандарта MPI, исследовано поведение нашего ПО при использовании различных способов выделения памяти.

Численное моделирование поверхностных волн в больших акваториях проводилось с учетом сферичности Земли и ускорения Кориолиса на основе уравнений мелкой воды [1]. В [2] для этой задачи построен метод конечных элементов, который приводит к системе линейных алгебраических уравнений. Полученная система решается итерационным методом Якоби, который обладает хорошим параллелизмом, а диагональное преобладание для его сходимости обеспечивается выбором шага по времени [3, 4].

Отметим некоторые особенности реализации алгоритма решения, диктуемые методом конечных элементов. Глобальная матрица жесткости зависит от времени и должна пересчитываться на каждом временном шаге. Однако для реализации метода Якоби на конечных элементах не требуется явного хранения глобальной матрицы жесткости. В программе насчитываются только элементы локальных матриц жесткости (причем только их диагональные элементы зависят от времени и перевычисляются на каждом временном шаге). Сборка невязки производится по треугольникам с использованием элементов локальных матриц жесткости.

*Работа выполнена при финансовой поддержке РФФИ (код проекта 11-01-00224-а)

2. Параллельная реализация. Оценка потенциального ускорения параллельного алгоритма

Используя явный параллелизм по данным, исходную расчетную область можно разбить на несколько частично перекрывающихся подобластей. Расчеты в каждой подобласти выполняются независимо друг от друга в рамках итерации Якоби. После каждой итерации Якоби необходимо проводить согласование данных в перекрытиях. Имеет место, по крайней мере, два варианта декомпозиции расчетной области: декомпозиция с теневыми гранями и без теневых граней.

При декомпозиции с теневыми гранями исходная область включает взаимно перекрывающиеся подобласти, ширина перекрытия определяется шаблоном дискретного аналога. При этом невязка в граничных точках подобласти i-го процесса насчитывается в подобластях соседних процессов. С учетом семиточечного шаблона и согласованности триангуляции в нашем случае достаточно перекрытия подобластей в два слоя расчетных точек. В случае декомпозиции без теневых граней исходная область разрезается на подобласти, пересекающиеся только по границам разреза. Для каждой граничной точки подобласти невязка насчитывается частично только по тем треугольникам, которые лежат в подобласти. При обмене данными после каждой итерации Якоби требуется дополнительное суммирование для значений невязки в граничных для подобласти точках.

Второй способ декомпозиции более экономичен по памяти, прост в программировании, очевидно ее достоинство для неструктурированных сеток, когда границы подобластей не являются последовательным множеством точек.

Реализация параллельной программы осуществлялась на языке программирования Си с применением функций библиотеки передачи сообщений MPI.

В рамках выбранной схемы распределения данных все процессы осуществляют одни и те же вычисления, но только над разными подобластями. Структура обменов также является однородной, за исключением первого и последнего процессов. После каждой итерации метода Якоби процесс выполняет обмен данными со всеми своими соседями, число которых определяется декомпозицией и не зависит от количества процессов, принимающих участие в расчете.

Потенциальное ускорение алгоритма оценивается как отношение времени T_1 вычисления на одном процессоре к времени T_p вычислений на p процессорах: $S_p = T_1/T_p$. Выполним теоретические оценки потенциального ускорения, по возможности учитывая время, затрачиваемое при выполнении алгоритма на обмены, а так же накладные расходы на вычисления в перекрывающихся подобластях.

Обозначим время выполнения одной арифметической операции t_{op}, а время выполнения пересылки одного значения — t_{comm}. Ясно, что последняя величина является скорее виртуальной характеристикой скорости пересылок, однако вполне подходит для наших теоретических оценок.

Пусть N_{nd} — общее количество точек сетки расчетной области, s — количество операций, выполняемых в одной расчетной точке на каждой итерации Якоби, k - количество шагов по времени, ν - среднее количество итераций Якоби на каждом временном шаге. Тогда время выполнения алгоритма на одном процессоре можно оценить следующим образом: $T_1 \sim k\nu s N_{nd} t_{op}$.

Предположим, что при декомпозиции области нам удается равномерно распределить весь объем вычислений по процессорам. В этом случае для времени выполнения

алгоритма на p процессорах можно записать следующее:

$$T_p \sim T_1/p + T_{over} + T_{comm}. \quad (1)$$

Здесь T_{over} - время, затрачиваемое на дополнительные вычисления, связанные с декомпозицией области, T_{comm} — время, требуемое для обменов.

Как следует из принятой нами схемы распределения данных, на каждой итерации метода Якоби требуются выполнить следующие действия, порождаемые распределенностью данных: 1) глобальную операцию приведения для вычисления критерия останова итерационного процесса $T^1_{comm} \sim (t_{op} + t_{comm}) \log_2 p$; 2) обмен значениями части вектора невязки в каждой точке разреза $T^2_{comm} = \mu k \nu m N_{bnd} t_{comm}$, где m — количество значений которые необходимо передать соседнему процессу для одной точки разреза, μ — количество соседних процессов, с которыми происходит обмен; 3) дополнительное суммирование частей невязок для трех компонент вектора (u, v, ξ), насчитанных в соседнем процессоре $T_{over} \sim k \nu g N_{bnd} t_{op}$, поэтому, $g = 3$

В результате оценка (1) потенциального ускорения нашего параллельного алгоритма для случая использования неблокирующих двухточечных обменов имеет вид:

$$S_p = \frac{1}{\frac{1}{p} + \frac{g}{s}R + \frac{\log_2 p}{sN_{nd}}(1+\kappa) + \frac{\mu m}{s}R\kappa}. \quad (2)$$

Из оценки (2) следует, что для мелких сеток потенциальное ускорение близко к линейному для достаточно большого диапазона количества процессов. Величина ускорения определяется двумя параметрами. Первый параметр $R = N_{bnd}/N_{nd}$ характеризует декомпозицию расчетной области. Приемлемое ускорение обеспечивается малостью R, поэтому при построении декомпозиций сложных вычислительных областей наряду с требованием равенства вычислительной нагрузки на процесс необходимо обеспечивать минимальную протяженность границы подобласти для каждого процесса. Второй параметр $\kappa = t_{comm}/t_{op}$ характеризует коммуникационную среду. Этот параметр описывает вычислительную сеть очень условно, но он показывает, что для приемлемого ускорения следует выбирать архитектуру кластера с небольшими значениями κ. Еще раз отметим, что величина T_{over} на несколько порядков меньше, чем другие слагаемые знаменателя в (1) и не зависит от количества процессов. С учетом экономии памяти и легкости реализации на неструктурированных сетках, это дает преимущество декомпозиции без перекрытий.

3. Сравнение двух реализаций MPI и стратегий управления памятью

Вычисления выполнялись на кластере МВС-1000/ИВМ (собственная сборка ИВМ СО РАН). Кластер неоднородной архитектуры, содержит 48 вычислительных узлов, 99 ядер, сеть передачи данных — GigabitEthernet. Управляющий узел, сервер доступа и файловый сервер — Athlon64/3500+/1Gb с общей дисковой памятью 400 Гб под управлением ОС Gentoo Linux. Управляющая сеть — FastEthernet, сеть передачи данных — GigabitEthernet. Исследование ускорения было поведено на подмножестве однородных (двухпроцессорных двухъядерных) узлов кластера.

В исследовании была сопоставлена производительность двух популярных реализаций MPI: общеизвестного MPICH2 v.1.2.1p1 и OpenMPI v.1.4.1, являющегося "наследником" пакета LAM.

Рассматриваемая задача тестировалась в двух модификациях: со статическим и динамическим (calloc/free) выделением памяти под основные массивы и буферы. Сразу отметим, что вариант со статическим выделением не показал существенного преимущества какого-то одного пакета: расхождения во времени счета и во времени обменов данными во всех опробованных конфигурациях оказались достаточно малы, чтобы не принимать их во внимание.

Напротив, вариант с динамическим выделением памяти показал наличие зависимости от использованного пакета и его настроек. Строго говоря, исследования показали, что обнаруженные различия в поведении задачи зависят даже не от особенностей реализации тех или иных функций MPI в обоих пакетах, а связаны с отличиями в части работы с динамической памятью.

Пакет OpenMPI использует для управления динамической памятью менеджер памяти *ptmalloc*, применяя его как для борьбы с фрагментацией, так и для увеличения производительности приложения засчет ускорения работы процедур *malloc/free*. Одна из настроек, управляющая стратегией выделения / освобождения памяти, называется *mpi_leave_pinned*, и по умолчанию эта настройка включена. В пакете MPICH2 подобной настройки нет, однако есть возможность управлять стратегией, которой руководствуется системная библиотека *glibc* при обработке запроса о выделении памяти путем вызова функции *mallopt()* с соответствующими аргументами.

На рис. 1а представлены графики зависимости времени выполнения параллельной программы от количества процессов для различных стратегий работы с памятью.

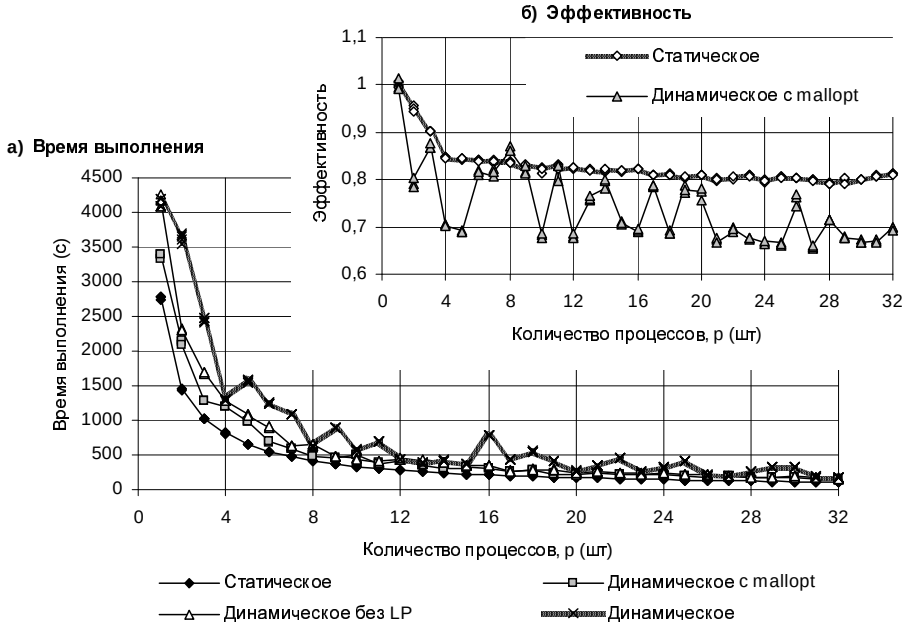

Рис. 1. Графики зависимости времени выполнения (а) и эффективности (б) от количества процессов для различных стратегий работы с памятью

Из рисунка видно, что лучшие результаты по времени счета и по соответствию теоретическим оценкам (выражающимся в гладкости кривой) показывает версия программы, работающая со статической памятью. Соответствующая ей кривая эффективности представлена на рис. 1 б.

Второй результат по малости времени вычислений и соответствию теории (рис. 1 а,б) демонстрирует стратегия "динамическое выделение + *mallopt*" (пакет MPICH2), в котором память распределяется динамически, однако системной библиотеке передана команда не использовать для выделения памяти механизм *mmap* (отображение страниц памяти). В условиях, когда приложение использует ресурсы ОС Linux в монопольном режиме — однократно выделяет большой объём памяти при старте и освобождает его только в конце работы, — фрагментация памяти не является актуальной проблемой, и, вероятно, оказывается, что скорость работы с памятью, выделенной из кучи (heap), выше.

Отключение *mpi_leave_pinned* в пакете OpenMPI дает временную кривую "динамическое без LP", изображенную на рис. 1 а. Графики эффективности для всех случаев динамического распределения памяти выглядят приблизительно одинаково (с точностью до локализации зубцов) и являются аналогами второго графика на рис. 1 б.

Таким образом, исследование показало, что динамическое выделение памяти без уточнения стратегии демонстрирует наихудшие результаты как по скорости работы, так и по гладкости кривой временных затрат. Отметим что, полученные результаты с высокой степенью точности совпадают для обоих исследованных пакетов.

В исследовании был также произведен замер времени обменов. График зависимости времени, затраченного на передачу данных, от количества использованных процессов представлен на рис. 2.

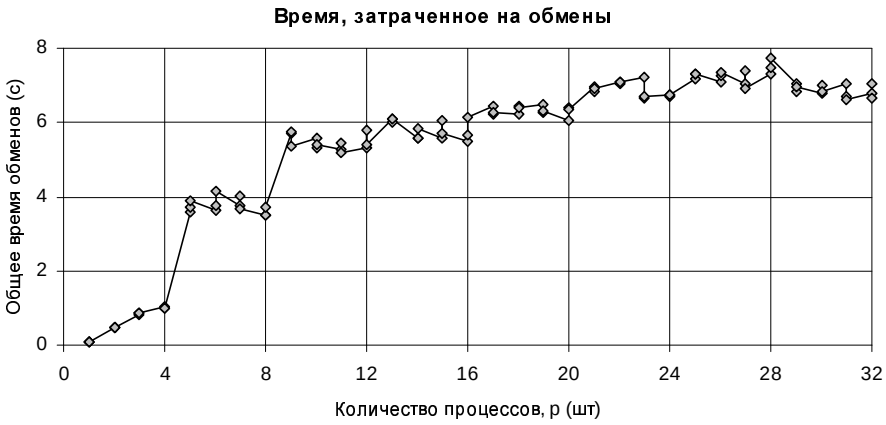

Рис. 2. Графики зависимости времени, затраченного на обмены от количества процессов

Время обменов мало различается во всех стратегиях, поэтому приводится только один график. Характерными его особенностями являются ожидаемый скачок в точке $p = 4$, который объясняется вводом в действие сети передачи данных для обмена между процессами номер 3 и 4, оказавшимися на разных хостах. Далее наблюдается менее очевидная, но также объяснимая ступенька в точке $p = 8$, когда хост, несущий процессы $4 \div 7$, начинает по той же сети обмениваться уже с двумя внешними соседями —

третьим и восьмым. Затем количество внешних соседей расти перестаёт, и мы видим слабый плавный рост времени обменов на графике. Последнее связано с необходимостью организации обменов суммарным значением невязки, а затраты на эту процедуру пусть и слабо, но зависят от количества узлов.

На рис. 3 представлена зависимость ускорения вычислений от количества используемых процессов для сетки 801×801, полученная на кластере ИВМ СО РАН после уточнения стратегии в случае динамического выделения памяти. Поскольку графики ускорений, полученных для декомпозиции с теневыми гранями и без теневых граней, практически совпадают, на рисунке изображен лишь последний из перечисленных. Для сравнения представлен график потенциального ускорения согласно оценке (2). Расчеты, проведенные на кластере после уточнения стратегии, показывают классическую картину ускорения, подтверждающую линейный характер его роста с увеличением количества процессов с эффективностью около единицы (эффективность расчета для 32 узлов ≈ 0.70).

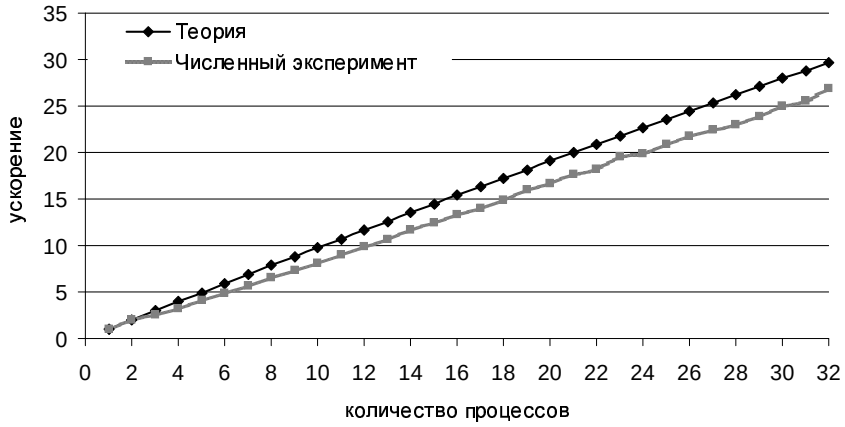

Рис. 3. Графики зависимости ускорения от количества процессов

Список литературы

[1] Agoshkov V.I. Inverse problems of the mathematical theory of tides: boundary-function problem // Russ. J. Numer. Anal. Math. Modelling. – 2005. – vol. 20, № 1. – P. 1–18.

[2] Kamenshchikov, L.P. Simulation of surface waves in basins by the finite element method / Kamenshchikov L.P., Karepova E.D., Shaidurov V.V. // Russian J. Numer. Anal. Math. Modelling – 2006. – Vol. 21(4). – Pp. 305–320.

[3] Карепова Е.Д., Шайдуров В.В. Параллельная реализация МКЭ для начально-краевой задачи мелкой воды // Вычислительные технологии. – 2009. – Т.14, № 6. – С. 45–57.

[4] Карепова Е.Д., Шайдуров В.В., Вдовенко М.С. Параллельные реализации метода конечных элементов для краевой задачи для уравнений мелкой воды // Вестник ЮУрГУ. Серия "Математическое моделирование и программирование". – 2009. – № 17 (150), Вып. 3. – С. 73–85.