УДК 519.6

Е. В. Дементьева [1], Е. Д. Карепова [2], А. В. Малышев [2]

[1] Сибирский федеральный университет
пр. Свободный, 79, Красноярск, 660041, Россия
E-mail: lionesskate@gmail.com

[2] Институт вычислительного моделирования СО РАН
Академгородок, Красноярск, 660036, Россия
E-mail: e.d.karepova@icm.krasn.ru; amal@icm.krasn.ru

ЭФФЕКТИВНОСТЬ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ НА КЛАСТЕРНЫХ СИСТЕМАХ РАСПРОСТРАНЕНИЯ ПОВЕРХНОСТНЫХ ВОЛН *

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

Ключевые слова: высокопроизводительная кластерная система, параллельный алгоритм, MPI, стратегия распределения памяти.

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, а также обнаружено несколько интересных эффектов, возникающих при измерении и оценке затрат на обмен данными между вычислительными процессами.

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 поддержана грантом РФФИ № 08-01-00621-а, интеграционным проектом № 26 СО РАН и ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы, ГК № 02.740.11.0621.

Авторы благодарят коллектив МВЦ ТГУ и ИВЦ НГУ, профессора А. В. Старченко и Д. Л. Чубарова за предоставленную возможность проведения серии вычислительных экспериментов на кластере СКИФ Cyberia и кластере ИВЦ НГУ.

Параллельный алгоритм

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 разбиения.

1. Декомпозиция 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. Поскольку алгоритм вычисления невязки требует хранения в каждой граничной точке подобласти семь коэффициентов матрицы жесткости, три значения вектора решения текущей и предыдущей итераций и значение правой части, то избыточность хранения информации для p процессов составляет $56(p-1)N_{bnd}$*ЧSizeOfDouble* байт. Здесь через N_{bnd} обозначено количество точек на разрезе, *SizeOfDouble* – количество байт, занимаемых переменной, хранимой с двойной точностью.

2. Декомпозиция 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, однако предполагает дополнительные арифметические операции на каждой итерации Якоби. Поскольку количество точек на разрезе мало по сравнению с общим количеством расчетных точек в подобласти, время, затрачиваемое на дополнительное суммирование, незначительно, кроме того, оно не зависит от количества используемых процессов. В наших расчетах время, затрачиваемое на дополнительное суммирование, было на три порядка меньше времени, затрачиваемого на обмен значениями на разрезе шириной в одну точку. Таким образом, декомпозиция без теневых граней выглядит несколько более привлекательной для наших целей. 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 также 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 требуется выполнить следующие действия, порождаемые распределенностью данных: 1) глобальную операцию приведения для вычисления критерия останова итерационного процесса; 2) обмен значениями части вектора невязки в каждой точке разреза; 3) дополнительные вычисления полной невязки в каждой точке разреза.

Оценим затраты времени на эти операции.

1. Для вычисления критерия останова итерационного процесса на каждой итерации требуется вычислить глобальный максимум по всем процессам. Предположим, что реализация глобальных операций приведения в MPI выполняется по оптимальному алгоритму сдваивания, что дает следующую оценку времени выполнения одной операции приведения $T^1_{comm} \sim (t_{op} + t_{comm}) \log_2 p$.

2. При использовании библиотеки MPI возможно два принципиально разных способа организации двухточечных обменов – с использованием блокирующих или неблокирующих функций передачи данных. Наша реализация алгоритма допускает использование неблокирующих обменов, которые в этом случае, безусловно, выгоднее блокирующих. Время обменов в рамках выбранной нами схемы не зависит от количества процессов, участвующих в расчетах. Имеет место следующая оценка времени для обмена частями вектора невязки: $T^2_{comm} = k\nu m N_{bnd}\, t_{comm}$, где m – количество значений которые необходимо передать соседнему процессу для одной точки разреза.

3. Пусть g – количество дополнительных вычислений, связанных с декомпозицией области, которые необходимы для одной точки разреза. Декомпозиция с теневыми гранями не требует прямых дополнительных вычислений, т. е. $g = 0$. При декомпозиции без теневых граней требуется дополнительное суммирование частей невязок для трех компонент вектора (u, v, ξ), насчитанных в соседнем процессоре, поэтому $g = 3$. Поскольку суммирование процессы выполняют асинхронно и независимо, то время, затрачиваемое на дополнительные вычисления в N_{bnd} точках разреза, можно оценить следующим образом:

$$T_{over} \sim k\nu g\, N_{bnd}\, t_{op}.$$

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

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

Из оценки (2) следует, что для достаточно мелких сеток потенциальное ускорение близко к линейному на достаточно большом диапазоне 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 $\zeta = t_{comm}/t_{op}$ характеризует коммуникационную среду. Этот параметр описывает вычислительную сеть очень условно, но он показывает, что для приемлемого ускорения следует выбирать архитектуру кластера с небольшими значениями ζ. Отметим также, 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.

Вычислительный эксперимент

Для численного исследования ускорения параллельного алгоритма рассматривалась модельная задача в «квадрате» на сфере: $\Omega = [0, \pi/10] \times [\pi/2, \pi/2 + \pi/10]$ с «твердыми» границами, для которой известно точное решение [2]. В расчетной области построена равномерная квадратная сетка 801×801 точек с соответствующей согласованной триангуляцией. В вычислительных экспериментах было сделано 1000 шагов по времени.

Серии расчетов проводились на трех высокопроизводительных комплексах.

Во-первых, вычисления выполнялись на 99-процессорном кластере ИВМ СО РАН. Кластер МВС-1000/ИВМ (собственная сборка ИВМ СО РАН, [5]) содержит 27 вычислительных узлов AMD Athlon64/3500+/1Гb (однопроцессорные, одноядерные); 12 вычислительных узлов AMD Athlon64 X2 Dual Core/4800+/2Гб (однопроцессорные, двуядерные); 12 вычислительных узлов 2XDual-Core AMD Opteron Processor 2216/4Гб (двухпроцессорные, двухъядерные). 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. Отличительной чертой кластера является его гетерогенная архитектура, чем мы пытались объяснить немонотонное ускорение, показанное нашей параллельной программой [3–4]. Для того чтобы исключить возможное влияние неоднородности, было проведено дополнительное исследование ускорения на подмножестве однородных (двухпроцессорных двухъядерных) узлов кластера ИВМ СО РАН.

Вторым кластером, на котором проведены серии расчетов, является кластер ТГУ SKIF Cyberia, который содержит 283 двухпроцессорных двухъядерных узла (1132 ядра) IntelXeon5150 2,66 Ггц, с суммарным объемом дисковой памяти 1136 Гб и объемом дискового пространства 22,56 Тб. Все узлы объединены высокопроизводительной сетью передачи данных InfiniBand.

Наконец, эксперименты были частично повторены на высокопроизводительном вычислительном комплексе НГУ, на базе кластера из 64 восьмиядерных вычислительных узлов, основанных на блэйд-серверах Hewlett-Packard BL460c. Все вычислительные узлы объединены высокопроизводительной сетью InfiniBand DDR 4x. В кластере используется параллельная файловая система хранения SFS емкостью 24 Тб, основанная на технологии Lustre. Кластер SKIF Cyberia и HP-кластер ИВЦ НГУ отличаются количеством ядер на узел.

Ясно, что потенциальное ускорение параллельной программы зависит от размерности сетки. Исследования зависимости ускорения проводились вплоть до 32 процессов, где на рассматриваемой сетке алгоритм хорошо масштабируем. Каждый вычислительный процесс всегда запускался на отдельном вычислительном ядре. Временные характеристики замерялись средствами MPI каждым процессом в отдельности, за показатель принималось максимальное значение. Все временные характеристики усреднялись по результатам нескольких десятков расчетов, исключая экстремальные значения. Дисперсия измерений, как правило, была незначительна, за исключением некоторых расчетов на кластере ИВМ СО РАН.

На рис. 1 представлена зависимость ускорения вычислений от количества используемых процессов, полученных на кластерах ИВМ СО РАН и SKIF Cyberia для случая декомпозиции расчетной области без перекрытий. Для сравнения представлен график потенциального ускорения согласно оценке (2). На эффективность параллельной программы способ декомпозиции расчетной области значимо не влияет. На большом количестве ядер проявляется явное преимущество реализации двухточечных обменов с помощью неблокирующих функций.

Расчеты, проведенные на кластере СКИФ Cyberia, показывают классическую картину ускорения, подтверждающую линейный характер его роста с увеличением количества процессоров с эффективностью около единицы (эффективность расчета для 32 узлов $\approx 0{,}85$). Эксперименты на кластере НГУ подтверждают эти результаты. В экспериментах на кластере ИВМ СО РАН общий тренд ускорения совпадает с теоретическими оценками, однако его рост имеет неустойчивый характер, что побудило авторов к дальнейшим исследованиям.

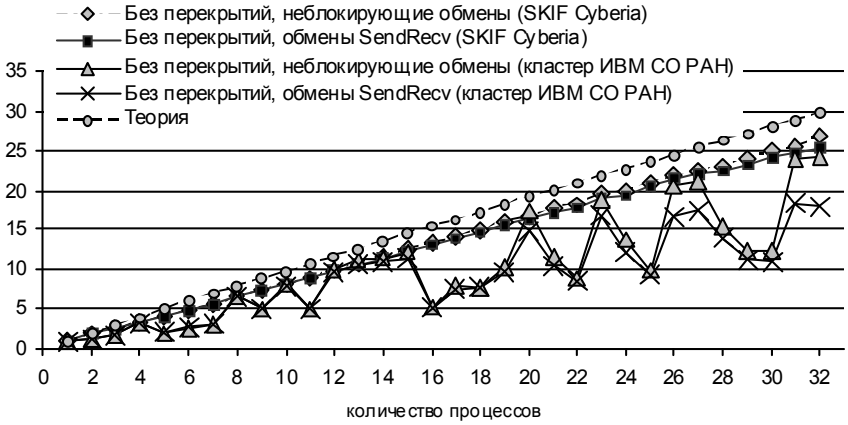

Рис. 1. Зависимость ускорения вычислений от количества доступных процессоров для различных кластерных систем и способов реализаций двухточечных обменов

Поскольку время выполнения программы складывается из времени вычислений, времени коммуникаций и дополнительного времени для операций, связанных с распределенностью данных, то были проведены серии расчетов, в которых эти составляющие были измерены отдельно. Во всех обсуждаемых далее экспериментах тестировался вариант программы, реализующий декомпозицию без теневых граней и неблокирующую схему двухточечных обменов.

Сравнение двух реализаций MPI и стратегий управления памятью

На рис. 2 приведены результаты численных экспериментов по замеру времени, затрачиваемого на содержательные вычисления. Для того чтобы была возможность сравнить результаты расчетов с теоретическими оценками, исследована безразмерная величина $\beta_{calc} = T_P^{calc} / T_1$. – отношение времени выполнения вычислений для P процессов к времени выполнения всего алгоритма для одного процесса. Из рис. 2 видна хорошая согласованность времени, реально затраченного на вычисления с его теоретической оценкой. Немонотонное поведение β_{calc} для кластера ИВМ СО РАН продиктовало наше дальнейшее исследование, проведенное на подмножестве однородных узлов.

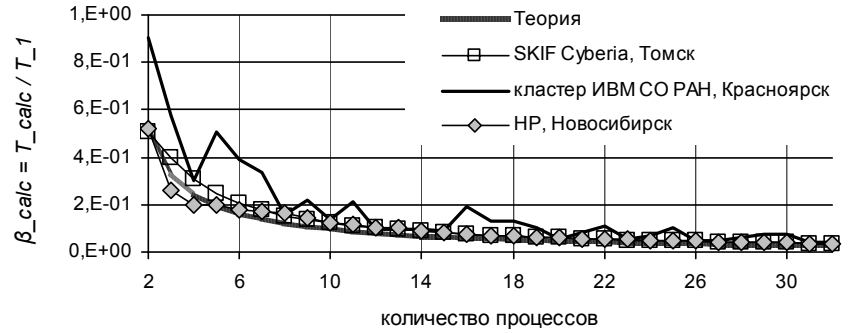

Рис. 2. Исследование для различных кластерных систем времени, затраченного на вычисления в зависимости от количества доступных процессоров

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 (и забегая вперед), 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 за счет 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. 3, *а* представлены графики зависимости времени выполнения параллельной программы от количества процессов для различных стратегий работы с памятью.

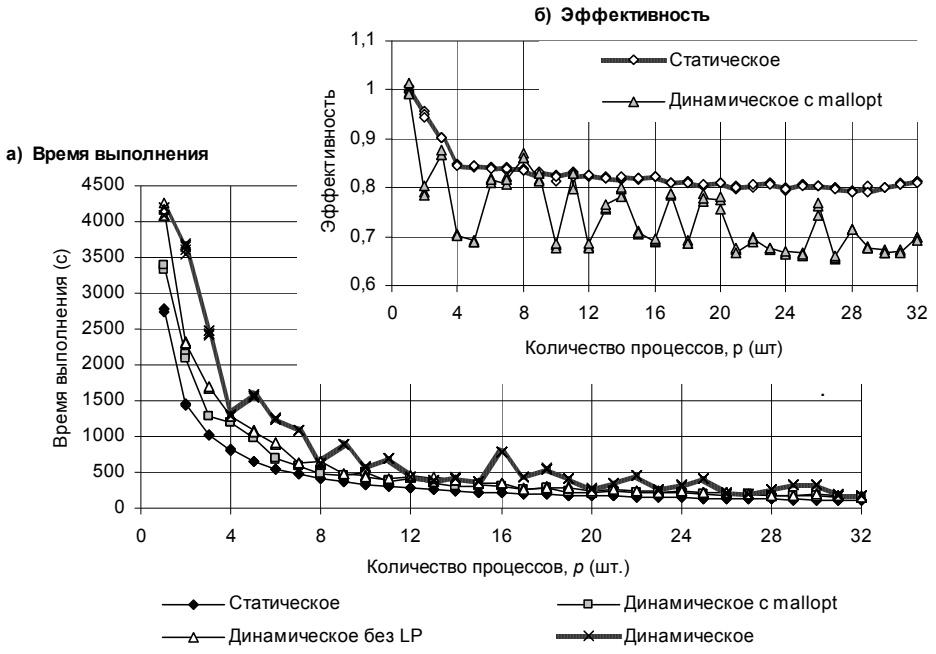

Рис. 3. Графики зависимости времени выполнения (*а*) и эффективности (*б*) от количества процессов для различных стратегий работы с памятью

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. 3, *б*. Интересен заметный перелом эффективности в точке $p = 4$, когда задача начинает считаться на более чем одном вычислительном хосте, и в обмены вовлекается сеть передачи данных кластера. Фактически это означает уменьшение отношения, определяющего сравнительную эффективность кластера.

Второй результат по малости времени вычислений и соответствию теории (рис. 3, *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 объем 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. Авторы не настаивают на этом объяснении, но предложить иное пока не могут.

Отключение mpi_leave_pinned в пакете OpenMPI дает временную кривую «динамическое без LP», изображенную на рис. 3, *а*. Графики эффективности для всех случаев динамического распределения памяти выглядят приблизительно одинаково (с точностью до локализации зубцов) и являются аналогами второго графика на рис. 3, *б*.

На рис. 4 представлен фрагмент тех же самых графиков, позволяющий увидеть отличия между кривыми более детально.

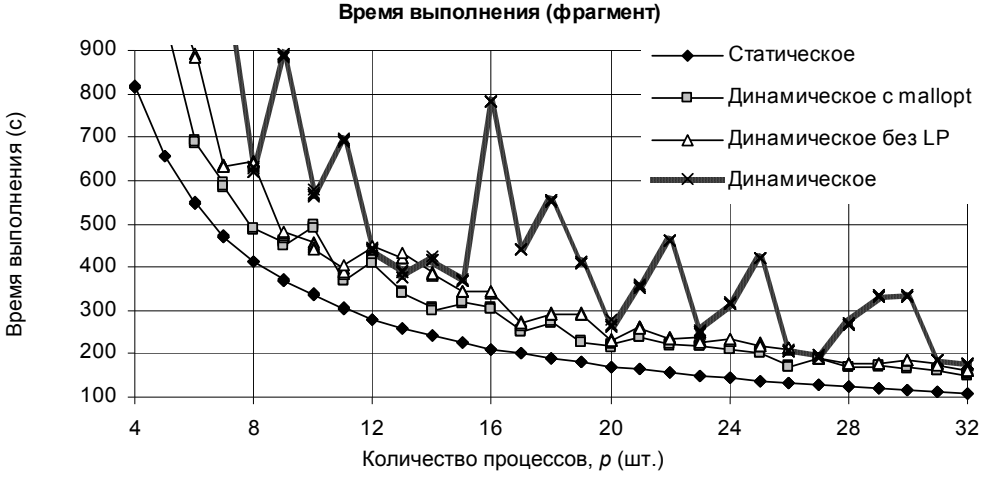

Рис. 4. Графики зависимости времени выполнения параллельной программы от количества процессов для различных стратегий работы с памятью

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.

О замерах времен обмена данными в SMP-узловых кластерах

Многие задачи, в том числе и тестируемая, используют блочное распараллеливание по области данных. Это приводит к необходимости регулярных обменов промежуточными данными вдоль линий разреза области, между узлами, обрабатывающими смежные области. Обработка данных каждым узлом в нашем случае использует алгоритм пульсации, т. е. до завершения обмена между соседями на каждой итерации продолжать вычисления на конкретном узле невозможно. Иными словами, цепочка «вычисления – обмен – вычисления – обмен…» имеет принципиально последовательный характер внутри каждого узла. В этих условиях операция обмена является локальным барьером, останавливающим счет до прихода данных от всех соседей. Это справедливо при выборе любой стратегии обменов, как неблокирующей, так и блокирующей: последняя может помочь «распараллелить» обмен с несколькими соседями, но совместить обмены и счет все равно не удастся.

Если разрезы области данных выполнены по линейной схеме, общее время передачи данных теоретически не должно зависеть от количества процессов.

При попытке замерить время, которое тратится на обмен данными в такой ситуации, обычно поступают примерно так, как показано в листинге 1 (неблокирующий вариант).

```
…
t0 = MPI_Wtime();
// m.1 – вызов функций приема данных
// m.2 – вызов функций передачи данных
// m.3 – проверка готовности данных на приеме
// m.4 – проверка на освобождение передаваемых данных
t1 = MPI_Wtime();
t = t1 – t0;
```

Точки *m*.3 и *m*.4 могут вызываться в цикле или фактически являться блокирующими вызовами – в данном случае это малосущественно. В дальнейшем значения времен *t* суммируются и используются для оценок времени или скорости обмена данными.

Результаты таких замеров для кластера SKIF Cyberia и HP-кластера НГУ демонстрирует рис. 5. Для того чтобы была возможность сравнить результаты расчетов с теоретическими оценками, на рисунке приведен график безразмерной величины $\beta_{comm} = T_P^{comm} / T_1$ – отношения времени выполнения обменов для P процессов к времени выполнения всего алгоритма для одного процесса.

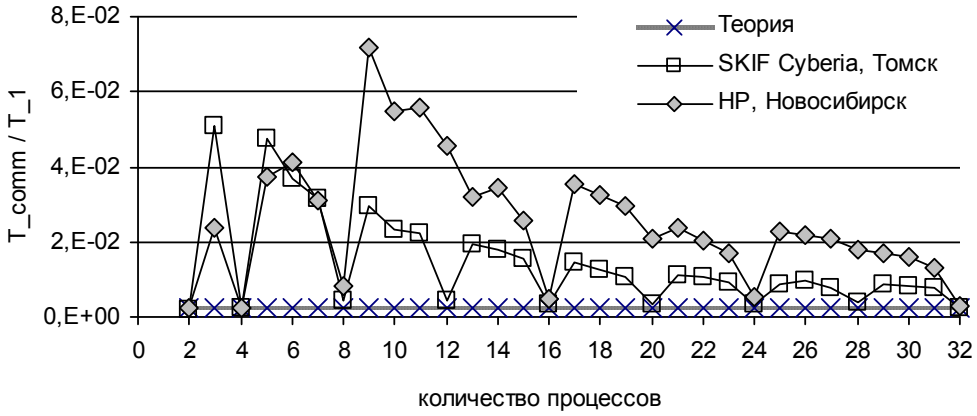

Рис. 5. Исследование для различных кластерных систем времени, затраченного на обмены в зависимости от количества доступных процессоров

Анализ полученных зависимостей показывает следующее:

1) время обменов минимально и не зависит от количества процессов, участвующих в обменах, если загружены все ядра на узле по одному процессу на ядро (для SKIF Cyberia количество процессов кратно четырем, для HP-кластера НГУ – восьми);

2) при существовании в расчетах узлов, неполностью загруженных, время обменов тем больше, чем больше количество простаивающих ядер;

3) время, затраченное на обмены, уменьшается с ростом количества задействованных процессов (т. е. уменьшением времени вычислений).

Такое странное поведение времени коммуникаций объясняется тем, что результаты измерений по схеме из листинга 1 иногда оказываются весьма далеки от реальности именно в силу «барьерности» функции приемопередачи: если сосед оказывается по какой-то причине не готов к передаче данных, то в измеренное значение *t* попадает и время ожидания его готовности.

Если измерения производятся на кластере, состоящем из одинаковых однопроцессорных узлов, и данные распределены равномерно, то эти погрешности невелики и не имеют систематического характера. Другую картину можно наблюдать, когда задача запущена на SMP-узловом кластере, где каждый хост несет некоторое количество (*m*) вычислительных ядер. В этом случае зависимость Σt от количества вычислительных узлов имеет характерный вид «пилы» с периодом, равным количеству ядер на хостах, причем максимумы наблюдаются в точках $p = m*i + 1$, $i = 1, 2,...$ (см. рис. 5). Это связано с тем, что процесс, выполняющийся на хосте в одиночку, всегда заканчивает счет немного быстрее, чем процессы, занявшие несколько узлов одного и того же хоста. Максимального значения этот разрыв достигает как раз тогда, когда, например, на одном хосте выполняются *m* процессов, а на другом – только один. Минимум же, напротив, наблюдается, когда все хосты нагружены одинаково, т. е. количество процессов *p* кратно *m*.

Теперь понятно, как следует изменить процедуру измерения времени. Для того чтобы исключить из замеров время простоя, необходимо с помощью MPI_Barrier() синхронизировать обменивающиеся хосты между собой, и только после этого засекать время и приступать к

обмену. Более тонким способом может быть раздельное измерение времени, проведенного в каждом неблокирующем вызове, синхронизация перед каждым блокирующим вызовом, или ручная организация одновременности передачи с учетом топологии разбиения данных. График зависимости времени, затраченного на передачу данных, от количества использованных процессов представлен на рис. 6.

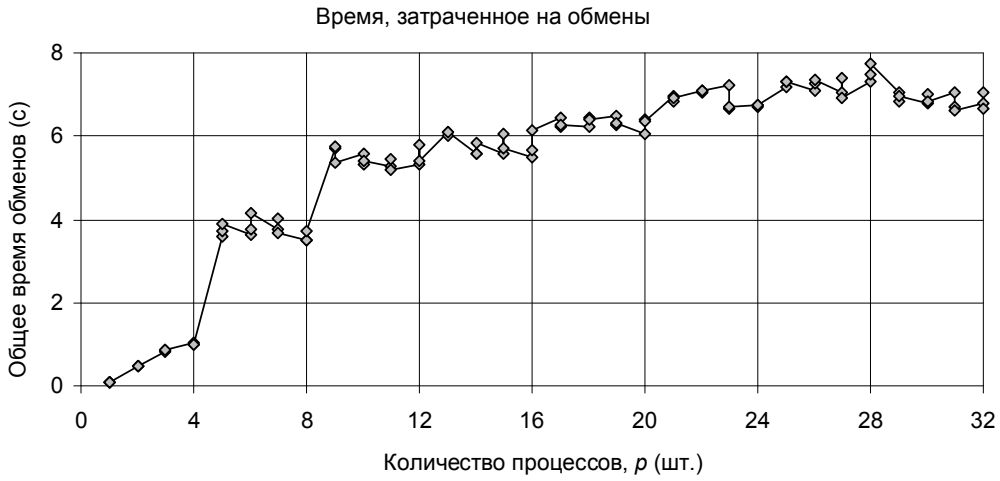

Рис. 6. Графики зависимости времени, затраченного на обмены от количества процессов

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 4ч7, начинает по той же сети обмениваться уже с двумя внешними соседями – третьим и восьмым. Затем количество внешних соседей перестает расти, 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

Заключение

В работе было исследовано несколько параллельных реализаций метода конечных элементов для начально-краевой задачи для уравнений мелкой воды. Наиболее эффективным следует признать алгоритм, основанный на декомпозиции без теневых граней с реализацией двухточечных обменов в неблокирующем режиме. Были также рассмотрены теоретические оценки ускорения параллельного алгоритма, которые хорошо согласуются с ускорением, показанным в расчетах. Численные эксперименты показали хорошую масштабируемость параллельной реализации.

Проведены серии численных экспериментов по сравнению производительности двух популярных реализаций MPI – общеизвестного MPICH2 v.1.2.1p1 и OpenMPI v.1.4.1. Расчеты показали чувствительность времени выполнения алгоритма к способу выделения памяти. Статическое распределение памяти дает преимущество в обеих реализациях MPI. Использование динамического распределения памяти в алгоритмах, аналогичных рассмотренному, требует дополнительного уточнения стратегии выделения памяти при компиляции. Лучший результат при использовании динамического выделения памяти показал пакет MPICH2 с уточнением стратегии выделения памяти mallopt(), отключающий механизм mmap.

Наконец, в работе исследована зависимость времени, затрачиваемого на обмены данными в SMP-узловых кластерах в алгоритмах пульсации.

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., Karepova E. D., Shaidurov V. V.* Simulation of surface waves in basins by the finite element method // Russian J. Numer. Anal. Math. Modelling. 2006. Vol. 21(4). P. 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 // Вестн. Южно-Урал. гос. ун-та. Серия Математическое моделирование и программирование. 2009. № 17 (150), вып. 3. С. 73–85.

5. *Исаев С. В., Малышев А. В., Шайдуров В. В.* Развитие Красноярского центра параллельных вычислений // Вычислительные технологии. 2006. Т. 11, спецвып. С. 28–33.

Материал поступил в редколлегию 07.10.2010

E. V. Dementyeva, E. D. Karepova, A. M. Malyshev

THE EFFICIENCY OF NUMERICAL MODELLING OF SEA SURFACE WAVES PROPAGATION USING HIGH PERFORMANCE CLUSTER SYSTEMS

Some aspects of effective application of high performance cluster systems for complex numerical problem are considered. Some parallel implementations of finite elements method for initial-boundary value problem for shallow water equations are investigated as example.

Keywords: high performance cluster system, parallel algorithm, MPI, strategy of memory allocation.