Модели параллельных вычислений для оценки реального ускорения исследуемого алгоритма*

И.Н. Коньшин[1,2,3]

ВЦ РАН, ФИЦ «ИУ» РАН, Москва[1], ИВМ РАН, Москва[2], МГУ, Москва[3]

В работе представлены две модели выполнения параллельных программ на платформах с общей и распределенной памятью, с помощью которых можно оценить ускорение при работе на конкретной вычислительной системе. Первая модель для оценки ускорения при работе с OpenMP использует закон Амдала. Вторая модель использует свойства исследуемого алгоритма, такие как арифметическая сложность алгоритма и общее количество информации, пересылаемой на другие процессоры. Для оценки ускорения используются также и характеристики вычислительной системы, в которые входит быстродействие и скорость передачи данных. Для некоторых алгоритмов, например, для метода сопряженных градиентов, получены оценки ускорения, а также проведены численные эксперименты для сравнения реального полученного ускорения с предсказанным теоретически.

Ключевые слова: параллельные вычисления, вычислительная сложность, коммуникационная сложность, ускорение.

1. Введение

Уже несколько десятилетий параллельные вычисления являются основным средством решения наиболее трудоемких задач вычислительной математики, линейной алгебры и многих других областей применения современных суперкомпьютеров [1]. Правильно подобранная модель параллельных вычислений позволяет наиболее адекватно оценить эффективность вычислительных алгоритмов. Это дает возможность сравнить производительности анализируемых алгоритмов на конкретных архитектурах и заранее выбрать для кодирования наиболее удачные из них.

Существует большое количество моделей параллельных вычислений (см., например, [1-3]), однако одни из них слишком поверхностны для получения количественных оценок ускорения, другие, наоборот, требуют учета слишком детальной информации об алгоритме и выполняемом коде. Причем, большинство моделей слабо учитывают особенности вычислительных систем, на которых выполняются программы, реализующие исследуемые алгоритмы. Хотелось бы, используя только макроструктуру алгоритма, разобраться какие из особенностей алгоритма наиболее важны при получении конструктивных количественных оценок ускорения, которые можно получить на конкретной вычислительной системе. Хотелось бы также знать какие характеристики вычислительных систем могут быть использованы при получении этих оценок.

Для оценки свойств алгоритмов на различных компьютерных архитектурах, например, системах с общей или распределенной памятью, могут понадобиться различные, наиболее подходящие к архитектуре, модели вычислений. При анализе алгоритмов для каждой из моделей, особенно важным оказывается знание условий и границ применимости этих моделей.

В настоящей работе сделана попытка описать две модели параллельных вычислений, оценки, получаемые на их основе, а также условия их применения. Выписываются и анализируются конструктивные оценки сверху на параллельную эффективность для нескольких алгоритмов линейной алгебры, включая метод сопряженных градиентов, а также проводятся качественное сравнение полученных оценок с результатами численных экспериментов.

Данная работа может быть полезна вычислителям, реализующих свои алгоритмы на параллельных компьютерах, и желающих заранее оценить эффективность работы алгоритма на

* Работа частично поддержана грантом РНФ 14-11-00190.

конкретной вычислительной системе. Она также может помочь в структурировании и классификации параллельных алгоритмов описываемых в открытом проекте AlgoWiki [4].

2. Модель параллельных вычислений для компьютеров с общей памятью

Пусть параллельные вычисления ведутся на компьютере с общей памятью, а для распараллеливания вычислений используется среда программирования OpenMP. В наиболее простом случае ее можно трактовать как систему расстановки директив компилятора для распараллеливания циклов. На некотором промежуточном этапе распараллеливания программы или из-за особенностей алгоритма, не все циклы могут оказаться распараллеленными.

Предположим, что вычисления достаточно однородны по составу выполняемых операций и, в принципе, мы можем подсчитать общее количество таких операций и узнать, сколько операций выполняется параллельно, а сколько последовательно. Пусть доля не распараллеленных операций составляет $f \in [0; 1]$ (эту величину также иногда обозначают символом s от английского слова «serial» (последовательный), но мы не будем его использовать, чтобы не путать с общепринятым обозначением ускорения S от английского «speedup»).

Если время выполнения арифметических операций T является линейным относительно количества операций, то можно оценить максимальное ускорение, достигаемое для рассматриваемой реализации алгоритма.

2.1. Закон Амдала

Обозначим время выполнения программы на p процессорах через $T(p)$, тогда ускорение, полученное при расчете на p процессорах, будет выражаться классической формулой:

$$S(p) = T(1) / T(p). \quad (1)$$

Если доля последовательных операций алгоритма равна f, то, расписывая подробнее формулу (1), максимально достижимое ускорение можно оценить сверху следующим образом:

$$S(p) = T(1) / (f T(1) + (1 - f) T(1) / p)) = p / (1 + f (p - 1)). \quad (2)$$

Формула (2) носит название закона Амдала (Amdahl’s law) [5]. Она может применяться и в более широкой трактовке, как закон, оценивающий максимально достижимое ускорение, которое можно получить на произвольной параллельной архитектуре для анализируемого алгоритма или конкретного кода [6].

Для анализа полученной формулы (2), заметим, что если в программном коде не распараллеленными остались лишь 1% операций, то при выполнении кода на 100 процессорах, вычисления не распараллеленной части операций на 1 процессоре займут примерно такое же время, как и вычисления распараллеленной части на 100 процессорах, значит, ускорение получится равным примерно 50 (или эффективность равной 50%).

Если количество используемых процессоров можно увеличивать не ограничено, то предельно достижимое ускорение будет равно $S = 1 / f$. Отметим также другой крайний случай, когда линейное ускорение $S(p) = p$ достигается при $f = 0$.

Оценим условия наилучшей применимости закона Амдала для вычислений на компьютере с общей памятью, т.е. когда реальное ускорение алгоритма будет максимально близким к значению (2), и перечислим основные из этих условий:

- арифметические операции достаточно однородны;
- все доступные нити участвуют в вычислениях параллельной части кода;
- сбалансированность вычислений для всех нитей;
- масштабируемость работы нитей, т.е. скорость работы нитей не зависит от количества нитей (или, другими словами, время выполнения параллельной части действительно уменьшается в p раз при использовании p нитей).

Проанализируем последнее условие более детально.

2.2. Реальная эффективность работы параллельной программы

Исследуем вопрос: если алгоритм распараллеливается идеально, то какого максимального ускорения можно достигнуть на конкретном параллельном вычислительном устройстве.

Рассмотрим реализацию операции DAXPY из BLAS1, которую вместе с директивой OpenMP запишем следующим образом:

```
#pragma omp parallel for
for (i=0; i<n; i++) y[i] += a * x[i];
```

Численные эксперименты проводились на кластере ИВМ РАН [7]. Характеристики вычислительных узлов из очереди «x6core», используемых для проведения расчетов:

- Compute Node Asus RS704D-E6;
- 12 ядер (два 6-ядерных процессора Intel Xeon X5650@2.67ГГц);
- Оперативная память: 24 Гб.;
- Дисковая память: 280 Гб.;
- Операционная система: SUSE Linux Enterprise Server 11 SP1 (x86_64).

Для сборки кода использовался компилятор Intel языка C версии 4.0.1, с поддержкой MPI версии 5.0.3.

Вышеприведенный фрагмент кода помимо окружения OpenMP для сравнения запускался также и в окружении MPI. Полученные значения ускорения (1) и эффективности $E = S / p$ приведены в таблице 1. Для распараллеливания с помощью OpenMP можно отметить две особенности:

- ожидаемое уменьшение эффективности E^*_{omp} для большого количества нитей $p = 11, 12$, вызванное, вероятно, недостаточной пропускной способностью канала памяти;
- неожиданное сверхлинейное ускорение вычислений для $p = 3,\ldots,8$ нитей, нарушающее выполнение закона Амдала, что было вызвано, вероятнее всего, когерентностью доступа к памяти для этой операции, а также эффективностью работы компилятора.

Для реализации в окружении MPI эффективность E^*_{mpi} вычислений идеально параллельной программы, как и следовало ожидать, оказалась близка к 1.

Таблица 1. Эффективность выполнения операции DAXPY в окружении OpenMP и MPI

p	E^*_{omp}	S^*_{omp}	E^*_{mpi}	S^*_{mpi}
1	1.000	1.00	1.000	1.00
2	0.939	1.87	0.948	1.89
3	1.778	5.33	0.994	2.98
4	1.929	7.71	0.987	3.94
5	1.496	7.48	0.994	4.97
6	1.481	8.89	0.986	5.92
7	1.095	7.66	0.997	6.98
8	1.011	8.09	0.977	7.82
9	0.863	7.77	0.988	8.89
10	0.842	8.42	0.933	9.33
11	0.638	7.02	0.960	10.56
12	0.385	4.63	0.985	11.82

Таким образом, если бы мы хотели подправить формулу закона Амдала (2) в соответствии с особенностями численного эксперимента на конкретной вычислительной системе, мы должны были бы домножить правую часть формулы (2) на величину E^*_{omp}:

$$S(p) = p\, E^*_{\text{omp}} / (1 + f\,(p - 1)). \qquad (3)$$

Величина E^*_{omp} при этом считалась бы заданной в табличном виде, в соответствии с таблицей 1. При этом возможная «сверхлинейность» ускорения оказалась бы заложенной в новую формулу (3), что предотвратило бы нарушение закона Амдала.

3. Модель параллельных вычислений для компьютеров с распределенной памятью

При работе алгоритма на системе с распределенной памятью часто оказывается что все процессы выполняются независимо и последовательно выполняемые операции полностью отсутствуют. Это означает что $f = 0$ и $S = p$, т.е. применение закона Амдала (2) для оценки ускорения в этом случае становится бессмысленным. Требуется найти основу для построения количественных оценок ускорения на другой основе.

Основной особенностью работы параллельного алгоритма на системе с распределенной памятью будет наличие обменов данными и связанными с ними дополнительной потерей эффективности вычислений. Рассмотрим вопросы, связанные с межпроцессорной передачей данных более подробно.

3.1. Скорость передачи сообщений

Чтобы оценить время передачи сообщений воспользуемся достаточно часто используемой формулой:

$$T_c = \tau_0 + \tau_c L_c,$$

где τ_0 – время инициализации сообщения, τ_c – скорость передачи сообщений (т.е. время передачи сообщения единичной длины), а T_c – время передачи сообщения длины L_c. Величина τ_0 определяется длительностью подготовки сообщения к передаче, поиском маршрута в сети, передачей служебной информации, например, заголовка сообщения, и т.п. Величина τ_c определяется полосой пропускания коммуникационных каналов в сети. В принципе, время инициализации сообщения τ_0 (латентность передачи сообщений) может быть достаточно большим, например, $\tau_0 = 100\tau_c$, т.е. время передачи 100 чисел может занимать всего лишь вдвое больше времени, чем передача одного числа. Однако, если длина сообщения достаточно велика, например, больше 1000, то временем инициализации сообщений можно пренебречь.

Наиболее эффективной реализацией алгоритма, будет реализация с передачей сообщения большой длины. Такие алгоритмы называют алгоритмами с «крупноблочным» («large-grained») параллелизмом. Если считать, что мы рассматриваем именно такие алгоритмы, то время передачи сообщения можно выразить упрощенной формулой:

$$T_c = \tau_c L_c,$$

Другими словами, мы пренебрегаем латентностью коммуникационной сети, и считаем, что скорость передачи сообщений определяется ее пропускной способностью. Отметим, что при этом скорость передачи сообщений становится линейной. Дополнительно, это означает, что общее время передачи нескольких сообщений будет определяться только суммарной длиной всех сообщений. В дальнейшем, этот факт позволит существенно упростить оценку эффективности рассматриваемых параллельных алгоритмов.

3.2. Оценка параллельной эффективности алгоритма

Введем те же обозначения, что и в подразделе 2.1. Пусть p – количество используемых процессоров, а $T(p)$ – время решения задачи на p процессорах. Тогда ускорение расчета, которое можно получить, используя данный алгоритм, будет выражаться формулой $S = T(1) / T(p)$. При этом эффективность работы алгоритма можно будет вычислить по формуле $E = S / p$.

Чтобы оценить время работы алгоритма, нам потребуется знание, как характеристик анализируемого алгоритма, так и параметров используемого параллельного компьютера.

Пусть L_a – общее количество арифметических операций алгоритма, а τ_a – время выполнения одной арифметической операции. Аналогично, пусть L_c – общая длина всех сообщений, а τ_a – время передачи сообщения единичной длины. Тогда, время выполнения арифметических операций можно выразить формулой $T_a = \tau_a L_a$, а время передачи всех сообщений $T_c = \tau_c L_c$.

В принципе, теперь все готово для оценки ускорения, но введем еще две вспомогательные величины, первая из которых будет отражать общую характеристику «параллельности» компьютера:

$$\tau = \tau_c / \tau_a,$$

означающую, сколько можно успеть выполнить арифметических операций за время передачи одного числа на другой процессор (при теоретически бесконечно быстрых обменах, или, формально, при использовании асинхронных обменов, эта величина равна 0, для компьютеров с достаточно быстрыми обменами может оказаться $\tau = 10$, а для медленных обменов часто бывает $\tau = 100$). Вторая величина будет выражать общую характеристику «параллельности» алгоритма:

$$L = L_c / L_a,$$

означающую, сколько арифметических операций делается на передачу одного числа.

Теперь уже все готово для оценки ускорения:

$$S = S(p) = T(1) / T(p) = T_a / (T_a / p + T_c / p) = p\, T_a / (T_a + T_c) = p / (1 + T_c / T_a) =$$
$$= p / (1 + (\tau_c L_c) / (\tau_a L_a)) = p / (1 + \tau L),$$

и, аналогично, оценки эффективности:

$$E = S / p = 1 / (1 + \tau L).$$

В итоге, получена предельно простая формула оценки эффективности, зависимая только от двух параметров τ и L, характеризующих параллельные свойства компьютера и алгоритма, соответственно. Только на первый взгляд удивительно, что в ней нет явной зависимости от количества процессоров p, на самом же деле она неявно присутствует в характеристике L через зависимость от общей длины всех сообщений L_c при заданном количестве процессоров p.

Перечислим предположения, которые были сделаны при выведении верхней оценки ускорения и эффективности работы параллельного алгоритма при работе на компьютере с распределенной памятью:

- в отличие от формулы закона Амдала, здесь считается, что распараллелены все вычисления и полностью последовательная часть алгоритма отсутствует;
- задержки в вычислениях происходят только из-за обменов, причем под эту модель больше подходят алгоритмы с синхронными обменами;
- процессоры загружены равномерно (или, другими словами, вычисления сбалансированы);
- вычислительные узлы однородны, это выражается в одинаковости параметра τ на всех вычислительных узлах (хотя, как известно, MPI может работать и в неоднородной вычислительной среде);
- скорость τ_a арифметических вычислений не зависит от количества процессоров p (для систем с распределенной памятью это выполняется гораздо чаще, чем при распараллеливании в общей памяти с использованием OpenMP, см. таблицу 1);
- скорость τ_c передачи сообщений, в пересчете на единичную длину сообщения также не зависит от количества процессоров p (этот менее очевидный факт означает масштабируемость коммуникационной сети используемого компьютера).

3.3. Оценка параллельной эффективности алгоритмов линейной алгебры

Рассмотрим теперь на некоторых примерах алгоритмов линейной алгебры применение полученных оценок ускорения и эффективности.

Пример 1 (идеально распараллеливаемые операции).

(а) Сложение двух векторов:

$$Z_i = X_i + Y_i, \qquad i = 1,\ldots,n.$$

(б) Нормализация вектора (умножение на константу):

$$X_i = \alpha\, X_i, \qquad i = 1,\ldots,n.$$

(в) Операция AXPY (является комбинацией двух первых операций, активно используется в численных методах, реализована в BLAS1):

$$Y_i = \alpha\, X_i + Y_i, \qquad i = 1,\ldots,n.$$

(г) Операция умножения на блочно-диагональную матрицу, с блоками, соответствующими каждому из процессоров, причем структура разреженности внутри блоков роли не играет, главное, чтобы общее количество ненулевых элементов внутри блоков для балансировки вычислений было примерно одинаковым.

(д) Операция решения системы с блочно-треугольной матрицей при выполнении прямого или обратного хода исключения, с блоками, соответствующими каждому из процессоров. Как и в предыдущем случае, структура разреженности внутри блоков роли не играет, если общее количество ненулевых элементов внутри блоков примерно одинаково.

Очевидно, что для выполнения этих операций не требуется производить обменов вычислениями ($L_a = 0$, а значит и $L = 0$), поэтому ускорение будет линейным: $S = p$ при любом значении параметра τ, а эффективность полной: $E = 1$. Все вычисления здесь независимы, причем, для случаев (а)-(в) максимально можно задействовать $p = n$ процессоров. Стоит также отметить, что здесь существенно предполагается равномерное распределение загрузки по процессорам, т.е. количества компонент вектора для случаев (а)-(в) и количества ненулевых элементов для случаев (г) и (д).

Пример 2 (скалярное произведение):

$$c = \Sigma X_i Y_i, \quad i = 1,...,n.$$

На каждом из процессоров сначала нужно локально подсчитать «свою» часть частичной суммы, а затем необходимо провести общее суммирование и разослать результат остальным процессорам. С помощью библиотеки MPI это можно сделать, например, вызвав функцию MPI_AllReduce(). Способ реализации этой функции в стандарте MPI не фиксирован и оставляется на усмотрение конкретной реализации MPI. Однако, при оценке ускорения мы можем воспользоваться самой простой идеей, переслав все частичные суммы на головной процесс, выполнив там суммирование, а затем разослав результат на все остальные процессоры.

Общее количество арифметических операций (считая за операцию сложение с умножением или отдельную операцию сложения на головном процессоре) будет равно $L_a = n + (p\text{-}1)$, а общая длина обменов составит $L_c = 2(p\text{-}1)$.

Так как при вычислении L нас интересует только отношение этих величин, то удобнее выписывать их в пересчете на каждый из процессоров, т.е. $L_a = (n+(p\text{-}1)) / p$ и $L_c = 2(p\text{-}1) / p$. В дальнейшем, не оговариваясь, будем подразумевать именно такие оценки.

В итоге, оценка ускорения будет выглядеть следующим образом:

$$L = L_c / L_a = 2(p\text{-}1) / (n + (p\text{-}1)),$$

$$S = p / (1 + 2(p\text{-}1)\,\tau/(n + (p\text{-}1))).$$

Например, для $n = 1000000$ и $\tau = 10$ подсчитаем несколько значений оценок:

$$S(p=1) = 1, \quad S(p=100) \approx 99.8, \quad S(p=1000) \approx 98,$$

а при $\tau=100$ и той же размерности вектора мы получим:

$$S(p=1) = 1, \quad S(p=100) \approx 98, \quad S(p=1000) \approx 80.$$

Операция вычисления скалярного произведения является очень важной и часто используемой в линейной алгебре операцией. Замечено, что при росте числа процессоров до величины порядка $p = 1000$ часто происходит резкое замедление скорости ее работы. Приведенные оценки косвенно подтверждают этот факт.

Пример 3 (умножение на плотную матрицу, строчный вариант).

Рассмотрим умножение плотной матрицы на вектор:

$$Y_i = A_{ij} X_j, \quad i,j = 1,\ldots,n,$$

считая, что на каждом процессоре хранится часть строк матрицы, а также соответствующие части вектора X и вектора результата умножения Y:

```
[:]   [== == ==]   [:]
---   ----------   ---
[:] = [== == ==] * [:]
---   ----------   ---
[:]   [== == ==]   [:]
```

Для выполнения умножения сначала нужно собрать на каждом из процессоров копию вектора X полной размерности, а затем уже выполнить умножение на «свою» часть матрицы A.

Пусть n – размерность матрицы и строки матрицы распределены по процессорам равномерно, тогда:

$L_a = n^2/p$, $L_c = (n/p)\,(p\text{-}1)$, $L = L_c / L_a = (p\text{-}1) / n$,

$S = p / (1+(p\text{-}1)\,\tau / n)$.

Если $n = 1000$ и $\tau = 10$, то $S(p\text{=}1) = 1$, $S(p\text{=}10) \approx 9$, $S(p\text{=}100) \approx 50$.

Пример 4 (умножение на транспонированную плотную матрицу).

Рассмотрим умножение транспонированной плотной матрицы на вектор:

$Y_i = A^T_{ij} X_j$, $i, j = 1,\ldots,n$,

считая, что на каждом процессоре хранится часть столбцов транспонированной матрицы (или часть строк исходной), а также соответствующие части вектора X и вектора результата умножения Y:

```
[:]    [:: :: ::]    [:]
---    ----------    ---
[:] =  [:: :: ::] *  [:]
---    ----------    ---
[:]    [:: :: ::]    [:]
```

Для выполнения умножения сначала нужно подсчитать произведение локальных столбцов матрицы A на «свою» часть вектора X и получить частичную сумму Z (полной длины n), затем части вектора Z разослать на соответствующие процессоры, а уже потом на каждом из процессоров сложить присланные частичные суммы и получить «свою» часть вектора результата Y.

Пусть n – размерность матрицы, тогда:

$L_a = n^2 / p$, $L_c = (n/p)\,(p\text{-}1)$, $L = L_c / L_a = (p\text{-}1) / n$,

$S = p / (1+(p\text{-}1)\,\tau / n)$.

Удивительным здесь является то, что, несмотря на совершенно другую организацию работы алгоритма умножения на матрицу, получена те же самые оценки, что и в предыдущем примере из-за того, что общая длина обменов осталась без изменения.

Пример 5 (умножение на ленточную матрицу).

Рассмотрим умножение хранящейся по строкам ленточной матрицы на вектор, считая, как и раньше, что на каждом процессоре хранится часть строк матрицы, а также соответствующие части вектора X и вектора результата умножения Y:

```
[:]    [===     ]    [:]
---    ---------    ---
[:] =  [  ===   ] *  [:]
---    ---------    ---
[:]    [     ===]    [:]
```

Пусть n – размерность матрицы, а r – полуширина ленты, тогда для выполнения умножения на «свою» часть строк матрицы A каждый процессор должен дополнительно получить от двух соседних процессоров по r компонент вектора X:

$L_a = (2r+1)\,n / p$, $L_c = 2r\,(p\text{-}1) / p$, $L = (2r/(2r+1))\,(p\text{-}1) / n \approx (p\text{-}1) / n$,

$S \approx p / (1 + (p\text{-}1)\,\tau / n)$.

В этой оценке удивительным оказалось то, что она почти в точности повторяет оценку для умножения на плотную матрицу и почти не зависит от полуширины ленты r. Это означает, что хотя, по сравнению со случаем плотной матрицы, у нас уменьшилось количество операций умножения, но пропорционально уменьшилась и длина обменов.

Пример 6 (умножение на диагонально-разреженную матрицу).

Рассмотрим умножение на разреженную матрицу, ненулевые элементы которой находятся на диагоналях, соответствующих некоторому шаблону дискретизации. Пусть, как и раньше, на каждом процессоре хранится часть строк матрицы, а также соответствующие части вектора X и вектора результата умножения Y:

```
[:]    [\\ \      ]    [:]
---    -----------    ---
[:] = [ \ \\\ \ ] * [:]
---    -----------    ---
[:]    [      \ \\]    [:]
```

Пусть n – размерность матрицы, r – полуширина ленты, а d – общее количество диагоналей (или количество узлов в шаблоне дискретизации), тогда для выполнения умножения на свою часть строк матрицы A каждый процессор, как и в предыдущем примере, должен дополнительно получить от двух соседних процессоров по r компонент вектора X:

$$L_a = d\,n\,/\,p, \qquad L_c = 2r\,(p\text{-}1)\,/\,p, \qquad L = 2r\,(p\text{-}1)\,/\,(d\,n),$$

$$S = p\,/\,(1 + 2r\,(p\text{-}1)\,\tau\,/\,(d\,n)).$$

Стоит отметить, что для двумерной задачи размера $n = m \times m$ при использовании 5-точечного шаблона дискретизации задачи параметры разреженной матрицы будут равны $r = m$ и $d = 5$. Если при $\tau = 10$ мы возьмем $m = 1000$ и $n = 1000000$, то получим $S(p{=}1) = 1$, $S(p{=}10) \approx 9.7$, $S(p{=}100) \approx 71.9$.

Для трехмерной задачи размера $n = m \times m \times m$ при использовании 7-точечного шаблона дискретизации задачи стоит выбрать значения $r = m^2$ и $d = 7$. Если при $\tau = 10$ мы возьмем $m = 100$ и $n = 1000000$, то получим $S(p{=}1) = 1$, $S(p{=}10) \approx 8.0$, $S(p{=}100) \approx 26.2$. Последнее значение соответствует случаю, когда трехмерная область высоты $m = 100$ разбита на 100 слоев толщиной в одну ячейку, понятно, что в этом случае обменов будет чрезвычайно много и эффективность выполнения программы будет минимальна.

Дополнительно можно заметить, что эффективная полуширина ленты матрицы зависит также от разбиения области дискретизации по процессорам, например, в трехмерном случае выгоднее нарезать область не «слоями», а трехмерными подобластями. Это может существенно уменьшить общую длину обменов и повысить эффективность операции умножения на разреженную матрицу.

Однако, по сравнению с умножение на ленточную матрицу, достаточно низкая эффективность рассматриваемой операции при той же размерности обусловлена гораздо меньшим количеством арифметических операций при той же ширине ленты, а значит и при таких же коммуникационных затратах.

3.4. Метод сопряженных градиентов

В качестве заключительного примера рассмотрим оценку ускорения для метода сопряженных градиентов [8]. Рассмотрим применение переобуславливателя достаточно простой, но чрезвычайно часто применяемой структуры: блочного метода Якоби без перекрытия с использованием неполного разложения Холецкого IC0 без расширения структуры разреженности исходной матрицы. Основные операции, составляющие алгоритм были уже рассмотрены в предыдущем разделе 3.3:

- три операции «AXPY» (Пример 1с);
- два вычисления скалярного произведения «DOT» (Пример 2);
- умножение разреженно-диагональной матрицы на вектор «MVM» (Пример 6),
- решение линейной системы с блочно-диагональной матрицей переобуславливателя «SOL» (Пример 1д).

Выпишем теперь оценку ускорения при выполнении одной итерации этого алгоритма.

Пример 7 (метод сопряженных градиентов).

Вычислительные и коммуникационные затраты одной итерации метода сопряженных градиентов с переобуславливанием составляют:

$$L_a = 3L_a^{AXPY} + 2L_a^{DOT} + L_a^{MVM} + L_a^{SOL} = 3(n\,/\,p) + 2(n\,/\,p) + (d\,n\,/\,p) + (d\,n\,/\,p) = (2d + 5)\,n\,/\,p,$$

$$L_c = 3L_c^{AXPY} + 2L_c^{DOT} + L_c^{MVM} + L_c^{SOL} = 3 \cdot 0 + 2(2(p\text{-}1)/p) + (2r(p\text{-}1)/p) + 0 = (2r + 4)\,(p\text{-}1)/p.$$

Тогда характеристику «параллельности» алгоритма можно записать как

$$L = L_c\,/\,L_a = (2r + 4)\,(p\text{-}1)\,/\,((2d + 5)\,n),$$

при этом оценка ускорения будет выражаться формулой:

$$S = p / (1 + \tau L) = p / (1 + (2r + 4)\,\tau\,(p\text{-}1) / ((2d + 5)\,n)). \quad (4)$$

3.5. Численный эксперимент и сравнение с оценкой ускорения

Для проведения численных экспериментов использовался кластер ИВМ РАН [7], с вычислительными узлами из очереди «x6core», уже описанной в разделе 2.2.

Сначала были вычислены используемые в разделе 3.2 характеристики «параллельности» компьютера. Для замера скорости вычислений использовалась операция DOT над векторами двойной точности длины 10^6, а скорость обменов замерялась на двух одновременных асинхронных обменах также длины 10^6 для чисел двойной точности, обмены выполнялись без перекрытия вычислений. Были получены следующие значения:

$\tau_a = 3.14 \cdot 10^{-10}$, $\tau_c = 3.06 \cdot 10^{-8}$.

Это означает, что в качестве основной характеристики данного параллельного компьютера можно положить:

$\tau = \tau_c / \tau_a = 100$.

Заметим в скобках, что для других компьютеров эти характеристики могут быть более оптимистичны, например, при замере их на суперкомпьютере Ломоносов были получены значения: $\tau_a = 1.72 \cdot 10^{-10}$ и $\tau_c = 7.13 \cdot 10^{-9}$, что означает, что для этого компьютера можно положить $\tau = 40$. На современных компьютерах с еще более быстрыми обменами эта характеристика может достигать $\tau = 10$.

Для проверки оценок ускорения была использована разработанная в Институте вычислительной математики РАН программная платформа INMOST [9], доступная в виде исходных кодов на сайте [10]. В качестве модельной задачи была взята разработанная автором статьи тестовая программа solver_test002, также доступная на этом же сайте. Матрица линейной системы была получена при дискретизации по трехмерному 7-и точечному шаблону для области размера $n = m \times m \times m$. Полученная линейная система решалась методом сопряженных градиентов с помощью пакета PETSc [11]. В качестве переобуславливателя использовался аддитивный метод Шварца без перекрытий с разложением IC0 в подобластях, который был задан с помощью параметров:

```
-ksp_type cg
-pc_type asm
-pc_asm_overlap 0
-sub_pc_type ilu
-sub_pc_factor_levels 0
```

Была рассмотрена серия задач различной размерности, размер области дискретизации по каждому из измерений составлял m = 64, 96, 128, 160. Количество неизвестных (равное n) при этом составляло от 262 тыс. до 4 млн., а количество процессоров выбиралось равным p = 1, 2, 4, 8, 16, 32, 64.

При вычислении оценки ускорения для одной итерации метода сопряженных градиентов по формуле (4) были использованы следующие значения параметров $r = m^2$, $n = m^3$, $d = 7$ и $\tau = 100$. Для четырех рассмотренных матриц были численно получены значения ускорения относительно запуска на одном процессоре, а также построены графики теоретических оценок этого ускорения по формуле (4). Графики приведены на рис. 1 и 2. Стоит отметить, что поведение графиков качественно совпадает.

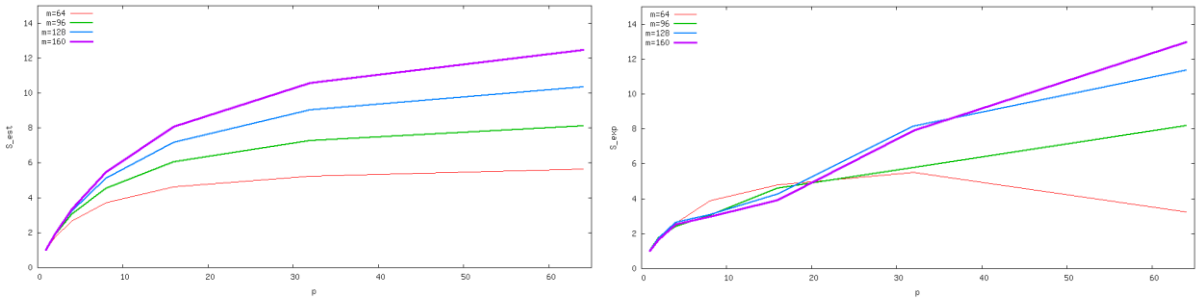

Рис. 1. Оценка ускорения по формуле (4) и реальное ускорение для задач m = 64, 96, 128, 160

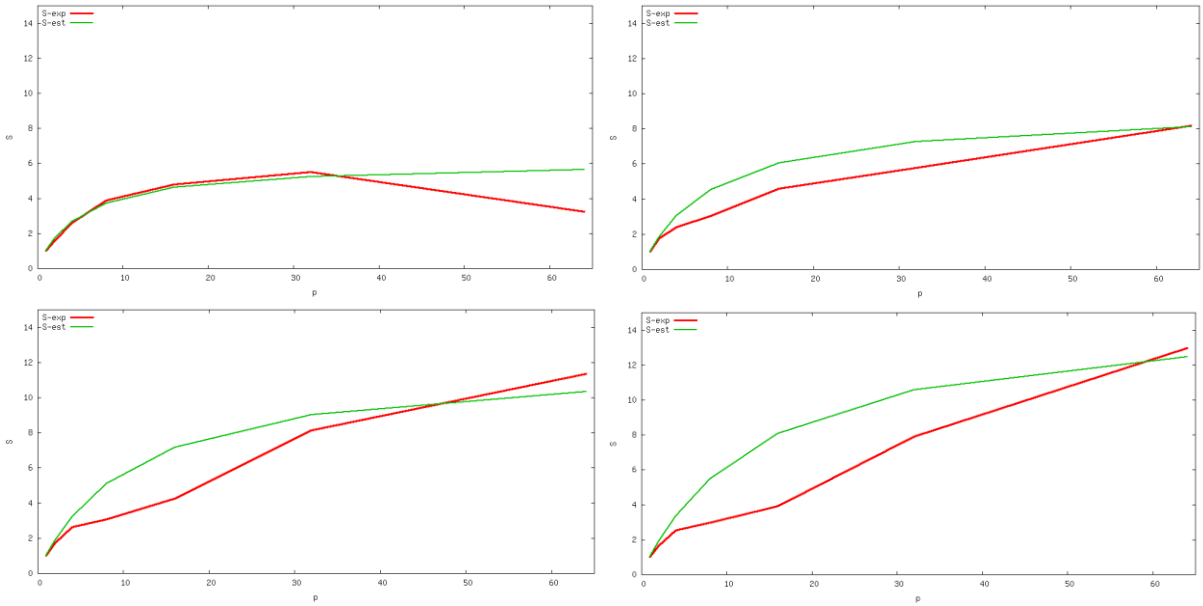

Рис. 2. Сравнение значений реального и оцененного ускорения для задач m = 64, 96, 128, 160

5. Заключение

Представлены модели параллельных вычислений для систем с общей и распределенной памятью. Базируясь на макроструктурных свойствах рассматриваемого алгоритма, получены количественные оценки ускорения, получаемого при вычислениях на параллельных компьютерах. Оценки для компьютеров с общей памятью базируются на доле не распараллеленных вычислений, а оценки для компьютеров с распределенной памятью основаны на характеристиках параллельных свойств компьютера и рассматриваемого алгоритма.

Проведены численные эксперименты, показывающие качественное совпадение теоретических оценок ускорения с экспериментально полученными результатами.

Литература

1. Воеводин В.В., Воеводин Вл.В., Параллельные вычисления, СПб.: БХВ-Петербург, 2002, С. 608.
2. Богачев К.Ю., Основы параллельного программирования, М.: БИНОМ, Лаборатория знаний, 2003.
3. Гергель В.П., Стронгин Р.Г., Основы параллельных вычислений для многопроцессорных вычислительных систем, Нижний Новгород: Изд-во НГУ, 2003.
4. AlgoWiki: Открытая энциклопедия свойств алгоритмов. URL: http://algowiki-project.org (дата обращения: 15.06.2016)

5. Amdahl G.M., Validity of the single-processor approach to achieving large scale computing capabilities. In: AFIPS Conference Proceedings, vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485. URL: http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf (дата обращения: 15.06.2016)
6. Антонов А., Под законом Амдала, Компьютерра, 11.02.2002, No. 430.
7. Кластер ИВМ РАН. URL: http://cluster2.inm.ras.ru (дата обращения: 15.06.2016).
8. Saad Y., Iterative Methods for Sparse Linear Systems, PWS, 1996.
9. Василевский Ю.В., Коньшин И.Н., Копытов Г.В., Терехов К.М., INMOST – программная платформа и графическая среда для разработки параллельных численных моделей на сетках общего вида, Москва: Изд-во Московского университета, 2013, 144 с.
10. INMOST – a toolkit for distributed mathematical modeling. URL: http://www.inmost.org (дата обращения: 15.06.2016).
11. PETSc (Portable, Extensible Toolkit for Scientific Computation). URL: https://www.mcs.anl.gov/petsc (дата обращения: 15.06.2016).

Parallel computational models for estimation of actual speedup of analyzed algorithm

I.N. Konshin[1,2,3]

CC RAS, FRC «IC» RAS, Moscow[1], INM RAS, Moscow[2], MSU, Moscow[3]

The paper presents two models of parallel program runs on platforms with shared and distributed memory. With these models we can estimate the speedup when running on a particular computer system. To estimate the speedup of OpenMP program the first model exploits the Amdahl's law. The second model uses properties of the analyzed algorithm, such as algorithm arithmetic and communication complexities. The computer arithmetic performance and data transfer rate are used for speedup estimation. For some algorithms, such as the preconditioned conjugate gradient method, the speedup estimations were obtained, as well as numerical experiments were performed to compare the actual and theoretically predicted speedups.

Keywords: parallel computations, computational complexity, communication complexity, speedup.

References

1. Voevodin V.V., Voevodin Vl.V., Parallel computing, SPb.: BHV-Pererburg, 2002, P. 608.
2. Bogachev K.Yu., Parallel programming, Moscow: Binom, 2003.
3. Gergel V.P., Strongin R.G., Parallel computing for multiprocessor computers, Nizhnij Novgorod: NGU Publ., 2003 (in Russian).
4. AlgoWiki: Open encyclopedia of algorithm properties. URL: http://algowiki-project.org (accessed: 15.06.2016)
5. Amdahl G.M., Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485. URL: http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf (accessed: 15.06.2016)
6. Antonov A., Under the Amdahl's law, Computerra, 11.02.2002, No. 430.
7. INM RAS cluster URL: http://cluster2.inm.ras.ru (accessed: 15.06.2016) (in Russian).
8. Saad Y., Iterative Methods for Sparse Linear Systems, PWS, 1996.
9. Vassilevski Yu., Konshin I., Kopytov G., Terekhov K. INMOST – a software platform and graphical environment for development of parallel numerical models on general meshes. Moscow State Univ. Publ., Moscow, 2013, 144 p. (in Russian).
10. INMOST – a toolkit for distributed mathematical modeling. URL: http://www.inmost.org (accessed: 15.06.2016).
11. PETSc (Portable, Extensible Toolkit for Scientific Computation). URL: https://www.mcs.anl.gov/petsc (accessed: 15.06.2016).