

М. А. Черепнёв, д-р физ.-мат. наук, доц., cherepniov@gmail.com,
МГУ имени М. В. Ломоносова

Оценка скорости работы нового параллельного блочного алгоритма для решения задач в области больших разреженных систем над большим простым полем*

Рассмотрен вопрос о том, как оценить время работы программы, реализующей алгоритм нахождения решения большой разреженной системы линейных уравнений над большим простым полем, изложенный в работах [1, 2], на вычислительной системе из K независимых вычислительных узлов. Блочный характер предлагаемого алгоритма позволит уменьшить время работы последовательной скалярной версии (алгоритм Монтгомери [3]) примерно в $\sqrt{K}$ раз, при K , растущем до размеров исходной задачи. Общее время работы всей программы нахождения решения разреженной системы линейных уравнений в пространстве размерности N снижено до $O\left(\frac{N^2}{\sqrt{K}}\right)$ при произвольном K , от 1 до N . Таким образом, теоретически, получено оптимальное время $O(N^{3/2})$.

Ключевые слова: дискретное логарифмирование, метод Ланцоша, параллельные вычисления, блочные алгоритмы

Введение

Задача нахождения некоторого количества (не обязательно всех) решений большой разреженной системы уравнений над большим простым полем возникает при решении задачи дискретного логарифмирования, которая связана со стойкостью очень многих систем защиты информации. Похожие исследования для случая $GF(2)$ описаны в работе [4]. Однако в настоящей статье предлагается ряд принципиально новых приемов, позволяющих существенно уменьшить время работы программы. Таким образом, настоящая работа имеет прежде всего важное прикладное значение. Цель работы — минимизация времени за счет увеличения числа используемых вычислительных узлов.

Рассмотрена топология, связывающая узлы вычислительной сети типа "тор". При этом считается, что в каждом из двух направлений этого тора действуют связи с разной пропускной способностью. Реально это может соответствовать крупным вычислительным кластерам, связанным относительно медленной сетью Интернет, либо несколькими многоядерным (возможно, с виртуальными ядрами) процессорам одного кластера. При этом в оценках времени работы учитывается не только время, затрачиваемое на счет, как это сделано в работе [2], но и время на пересылку. Около половины всех коммутаций допускается медленными, откуда получена возможность набрать необходимое количество

вычислительных узлов из разных кластеров. Предложено специфическое распределенное хранение промежуточных результатов вычислений, в результате получена экономия не только вычислительных ресурсов, но и времени на передачу, которая осуществляется параллельно по независимым ветвям заданной сети. Кроме того, решение плотных систем линейных уравнений, которое необходимо делать на каждом шаге представленного алгоритма, а также обращения плотных матриц предлагается осуществлять с помощью параллельных вычислений. Это позволит снизить время обращения матриц размера $s \times s$ до $O(s^2)$, а время работы последовательной скалярной версии (алгоритм Монтгомери [3]) примерно в $\sqrt{K}$ раз, при K , растущем до размеров исходной задачи. Общее время работы всей программы решения разреженной системы линейных уравнений в пространстве размерности N до $N^2/\sqrt{K}$ при произвольном K , от 1 до N . Таким образом, теоретически, получено оптимальное время $O(N^{3/2})$.

Основные операции, нуждающиеся в распараллеливании, это умножение разреженной матрицы на блок, умножение этих блоков друг на друга (скалярное произведение) и на относительно маленькие матрицы (линейные комбинации), а также решение линейных систем с матрицами относительно маленького, но все же достаточно большого размера.

Каждую операцию любого алгоритма можно рассматривать на предмет возможности использования нескольких однотипных узлов для уменьшения времени на ее реализацию. При этом если считать число арифметических операций в рассматриваемом

* Работа поддержана грантом РФФИ 18-29-03124 мк.

8К

алгоритме фиксированным, то самое значительное снижение времени — это падение времени обратно-пропорционально числу вычислительных узлов. Так будет, например, при наборе соотношений в методах дискретного логарифмирования и факторизации на основе факторных баз. Однако для большинства относительно более сложных алгоритмов верхняя оценка времени их работы в зависимости от параметров задачи и используемого оборудования представляется как $T(N, d, n, c...; s)$, где $N, d, n, c...$ будут введены позже, и числа используемых вычислительных узлов s . Эта оценка может вести себя по-разному на разных областях изменения аргумента. Зачастую, при увеличении s время не падает обратнопропорционально s . А при увеличении s выше определенного порогового значения $s_0(N, d, n, c...)$, зависящего от параметров задачи, оно снова начинает увеличиваться вследствие того, что время на обмен между s вычислительными узлами растет быстрее, чем уменьшается время работы каждого вычислительного узла в отдельности.

По мнению автора, правомерно поставить вопрос о вычислении значения величины $s_0(N, d, n, c...)$ и $T(N, d, n, c...; s) = \min_s T(N, d, n, c...; s) = T(N, d, n, c...; s_0(N, d, n, c...))$ для некоторого эталонного кластера. В качестве эталона на настоящее время логично взять кластер с неограниченным числом вычислительных узлов, для которых время выполнения одной арифметической операции с машинными словами, умноженное на некоторую константу c , равно времени передачи одного машинного слова между вычислительными узлами. В современной компьютерной технике обмен между оперативной памятью и процессором осуществляется с $c \approx 5$, а между отдельными частями оперативной памяти — $c \approx 20$ (время на обмен с кэшем процессора в этой статье учитываться не будет).

Насколько известно автору, параметр c определяется соотношением между тактовыми частотами процессора и шины, которая связывает вычислительные узлы, процентом информационных битов в передаваемых по внутренней сети сообщениях и некоторыми другими характеристиками кластера. Будем также предполагать, что на кластере возможна как адресная рассылка между выполняющими конкретные задания вычислительными узлами, так и рассылка с фиксированного узла на узлы некоторой группы по бинарному дереву (бродкаст), т. е. за логарифм от числа элементов этой группы. Пересылки внутри непересекающихся групп вычислительных узлов будем считать независимыми. Все оценки времени проведены в единицах, равных времени на выполнение одной арифметической операции с машинными словами в таком эталонном кластере. Будем считать также незначительным время на преобразование форматов хранения матриц. Все дальнейшие результаты получены при указанных условиях.

Задачу решения большой разреженной системы линейных уравнений

$$AX = B, B, X \in F^{N \times s},$$

будем характеризовать четырьмя параметрами: $N \geq 2^{20}$ — размер исходной матрицы (максимум из числа строк и столбцов); d — оценка числа ненулевых

элементов в каждой строке этой матрицы; c — параметр, о котором было сказано выше. Таким образом,

пересылка V бит будет осуществлена за $\frac{c}{n}V$ единиц

времени, где n — длина машинного слова. Одна единица времени — это время, необходимое для выполнения одной арифметической операции с машинными словами. Параметр s принято называть блочным фактором. В дальнейшем вместо векторов длины N будем рассматривать блоки, состоящие из s столбцов, а вместо скаляров — матрицы размера $s \times s$.

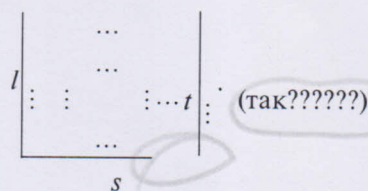
Решение ищем в пространстве Крылова, а именно в линейной оболочке столбцов матриц $\langle B, AB, A^2B, \dots \rangle$. Это решение может быть построено по формуле

$$X = \sum_{i=1}^{N/s} W_i (W_i^T A W_i)^{-1} W_i^T B,$$

где $\langle B, AB, A^2B, \dots \rangle = \langle W_0, W_1, \dots \rangle$, а матрицы W_i образуют базис пространства Крылова, ортогональный относительно скалярного произведения с матрицей A , и обладают некоторыми дополнительными свойствами. При этом матрицу A можно считать симметричной и вырожденной [1, 5].

В настоящей работе предпочтение отдано блочной версии ($s > 1$), поскольку при распараллеливании скалярной версии на первый план выходит время, затрачиваемое на пересылки. Например, чтобы собрать с помощью алгоритма циклической пересылки [6] вектор размера N , потребуется время порядка $O(N)$, а таких векторов в пространстве Крылова также N . Таким образом, получается время порядка N . Блочный подход позволяет распараллелить самую трудную часть алгоритма — построение пространства Крылова, используя тот факт, что после каждого умножения блока на матрицу результат не надо снова собирать в блок. Следует отметить, что в блочном подходе есть свой принципиальный минус — это растущие как квадрат блочного размера вычислительные затраты на скалярное произведение, при том, что число необходимых скалярных произведений уменьшается пропорционально его первой степени. Поэтому приходится увеличивать число вычислительных узлов, чтобы выполнить эту работу. Именно поэтому эффективность использования вычислительных узлов в этом алгоритме определяется знаменателем вида $\sqrt{K}$. Однако, поскольку K может расти с сохранением этой оценки примерно до N , существует возможность посчитать решение линейной системы за время порядка $N^{3/2}$, что невозможно на сегодняшний момент сделать другим алгоритмом на вычислительной системе такого же размера.

Рассмотрим следующую вычислительную систему, состоящую из $ls + t = K$ однотипных вычислительных узлов:



Такая конструкция является общей для данного метода. В настоящей работе эта конструкция рассматривается при условии, что вертикальные связи между узлами быстрее горизонтальных. Коэффициенты, соответствующие коэффициенту c , который упоминался выше, равны c_1 для вертикальных и c_2 для горизонтальных связей, $c_1 < c_2$. Левая часть этой вычислительной системы, состоящая из s вычислительных узлов, будет вычислять куски пространства Крылова, необходимые скалярные произведения, а также решение исходной системы. А правая часть, состоящая из t узлов, будет вычислять коэффициенты этого решения. Далее фактически будет написана программа (алгоритм работы) для каждого из однотипных вычислителей правой и левой частей вычислительной системы, осуществляющая баланс времени на счет и пересылки. Конечно, функции правой части вычислительной системы можно передать узлам левой части, сэкономив при этом время на пересылках. Однако в этом случае каждый узел должен будет проводить вычисления элементов системы одновременно по нескольким программам. Для некоторых (многоядерных) вычислителей это может и не привести к дополнительным затратам времени. Однако в контексте целей настоящей работы будем следить за тем, чтобы все вычислительные ресурсы каждого узла были постоянно задействованы в вычислениях, и на работу параллельной программы уже было нечего выделять.

Параллельное решение линейных систем

Как было отмечено выше, любую процедуру можно раздать на несколько вычислительных узлов и тем самым уменьшить время на вычисления в такое же число раз. Например, для приведения матрицы размера $s \times s$ к треугольному виду или для получения ее LU-разложения требуется не более s^3 операций. Значит на системе из s вычислительных узлов это потребует не более s^2 операций на каждом вычислительном узле. Однако для осуществления этих операций каждому вычислительному узлу может потребоваться s или больше элементов, вычисленных на других узлах, и ускорения не получится. Но если распараллеливать приведение к треугольному виду по столбцам, то для реализации каждого элементарного преобразования узлам необходимо получить всего один элемент и сделать одно умножение и одно вычитание. Поскольку необходимых элементарных преобразований не более $s^2/2$, получаем оценку времени для всего алгоритма приведения к треугольному виду $c_1 s^2/2$. Для решения системы, имеющей треугольный вид, требуется не более $s^2/2$ операций. Это означает, если каждому из s вычислительных узлов поручить в качестве правой части свой базисный вектор, то время на обращение матрицы потребуется в сумме не более $c_1 s^2$ с учетом рассылки всей треугольной матрицы. Аналогичное распараллеливание можно предложить и для получения LU-разложения. Для этого достаточно раздать s вычислительным узлам s -ую долю элементов матрицы для вычисления, равномерно разбросав их по всей матрице для балансировки нагрузки.

Формальное описание алгоритма

Будем пользоваться обозначениями, введенными в работе [1], только для случая большого простого поля F . Алгоритм работает в виде цикла, который строит последовательно часть ортогонального базиса пространства Крылова и соответствующие координаты решения.

При входе в очередной цикл алгоритма имеем k старших коэффициентов многочленов — матричных приближений Паде $Q^{(k)}(\lambda)$, $Q^{(k-1)}(\lambda)$ к ряду $\alpha(\lambda)$. Сначала на левой части вычислительной системы вычисляем блочные векторы

$$A^i Q^{(k)}(A, B), A^i Q^{(k-1)}(A, B), i = 0, 1, \dots, k, \quad (1)$$

где k — параметр, который будет выбран ниже. Эти векторы, также как и вектор B , будут иметь s столбцов, которые в соответствии с их номерами при всех $i = 0, 1, \dots, k$ хранятся на соответствующих группах вычислительных узлов левой вычислительной системы. Внутри группы будем хранить их распределенно по высоте на каждом по $\frac{N}{l}$ элементов каждого столбца.

Кроме того, в результате предыдущей синхронизации вычислены начальный отрезок решения (до k -го слагаемого включительно) и матрицы $W_i^T B$ при $i = k, k+1$, а также $(W_k^T A W_k)^{-1}$.

Вычислим коэффициенты γ_i и μ_i по следующим формулам:

$$\begin{aligned} (Q^{(k-1)}(A, B))^T A^i Q^{(k-1)}(A, B) &= \sum_{i=k}^{k+j-1} (Q_{i-j}^{(k-1)})^T \gamma_i, \\ (Q^{(k)}(A, B))^T A^i Q^{(k)}(A, B) &= \sum_{i=k+1}^{k+j} (Q_{i-j}^{(k)})^T \mu_i. \end{aligned} \quad (2)$$

Не дожидаясь окончания этих вычислений, запускаем циклический процесс, состоящий из повторяющихся двух частей, описанных ниже. Обозначим коэффициенты $Q_{j,i}^{(t)}$ в соответствии с формулами

$$Q^{(t)}(\lambda) = \sum_{i=0}^{t-k} \lambda^i Q_{k,i}^{(t)} + \sum_{i=0}^{t-k-1} \lambda^i Q_{k-1,i}^{(t-1)}, \quad t \geq k+1. \quad (3)$$

Начальные условия определим так: $Q_{k,0}^{(k-1)} = O_n$, $Q_{k-1,0}^{(k-1)} = I_n$, $Q_{k,0}^{(k)} = I_n$, $Q_{k-1,0}^{(k)} = O_n$ (остальные коэффициенты нулевые).

Для получения $Q_{j,i}^{(t+1)}$, сначала по формулам

$$\begin{aligned} \tau_j &= \sum_{i=0}^{t-k-1} \mu_{i+j} Q_{k,i}^{(t-1)} + \sum_{i=0}^{t-k-2} \gamma_{i+j} Q_{k-1,i}^{(t-1)}, \\ \tau_j &= \mu_j \quad \text{для } t = k+1, \\ \rho_j &= \sum_{i=0}^{t-k} \mu_{i+j} Q_{k,i}^{(t)} + \sum_{i=0}^{t-k-1} \gamma_{i+j} Q_{k-1,i}^{(t)}. \end{aligned} \quad (4)$$

вычисляются соответствующие коэффициенты левых частей в очередных формулах

$$\begin{pmatrix} \rho_t & O_n & \tau_{t-1} \\ \rho_{t+1} & \rho_t & \tau_t \\ O_n & O_n & O_n \\ O_n & & \\ O_n & & \\ O_n & \vdots & \vdots \\ O_n & & \\ O_n & & \\ Q_{k,t-k}^{(t)} & O_n & O_n \\ Q_{k,t-k-1}^{(t)} & Q_{k,t-k}^{(t)} & O_n \\ Q_{k,t-k-2}^{(t)} & Q_{k,t-k-1}^{(t)} & Q_{k,t-k-1}^{(t-1)} \\ Q_{k,t-k-3}^{(t)} & Q_{k,t-k-2}^{(t)} & Q_{k,t-k-2}^{(t-1)} \\ \vdots & \vdots & \vdots \\ Q_{k,1}^{(t)} & \cdot & \cdot \\ Q_{k,0}^{(t)} & Q_{k,1}^{(t)} & \cdot \\ O_n & Q_{k,0}^{(t)} & Q_{k,0}^{(t-1)} \\ O_n & & \\ O_n & & \\ O_n & \vdots & \vdots \\ O_n & & \\ O_n & & \\ Q_{k-1,t-k-1}^{(t)} & O_n & O_n \\ Q_{k-1,t-k-2}^{(t)} & Q_{k-1,t-k-1}^{(t)} & O_n \\ Q_{k-1,t-k-3}^{(t)} & Q_{k-1,t-k-2}^{(t)} & Q_{k-1,t-k-2}^{(t-1)} \\ Q_{k-1,t-k-4}^{(t)} & Q_{k-1,t-k-3}^{(t)} & Q_{k-1,t-k-3}^{(t-1)} \\ \vdots & \vdots & \vdots \\ Q_{k-1,0}^{(t)} & \cdot & \cdot \\ O_n & Q_{k-1,0}^{(t)} & Q_{k-1,0}^{(t-1)} \end{pmatrix} \begin{pmatrix} I \\ v_0 \\ v_{-1} \end{pmatrix} = \begin{pmatrix} O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ Q_{k,t-k+1}^{(t+1)} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Q_{k,0}^{(t+1)} \\ O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ Q_{k-1,t-k}^{(t+1)} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Q_{k-1,0}^{(t+1)} \end{pmatrix}, \quad (5)$$

$$Q^{(t+1)} = \begin{pmatrix} O \\ O_n \\ O_n \\ O_n \\ O_n \\ Q_{k,t-k+1}^{(t+1)} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Q_{k,0}^{(t+1)} \\ O_n \\ O_n \\ O_n \\ O_n \\ O_n \\ Q_{k-1,t-k}^{(t+1)} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Q_{k-1,0}^{(t+1)} \end{pmatrix}.$$

На выходе из этой части:

- рекуррентные формулы (т.е. коэффициенты v_i);
- коэффициенты вектора $Q^{(t+1)}$;
- в начале следующего цикла находим старший коэффициент $Q_{t+1}^{(t+1)}$, вычисленный по формулам (3) для вычисления скалярных произведений $W_{t+1}^T A W_{t+1} = Q_{t+1}^{(t+1)T} \sigma_{t+2}$, где σ_i — коэффициенты правой части Паде разложений:

$$\alpha(\lambda)Q^{(t+1)}(\lambda) - P^{(t+1)}(\lambda) = \sum_{i=t+2}^{\infty} \sigma_i \lambda^{-i},$$

которые вычисляют по формулам (4).

Вертикальный размер в формуле (5) соответствует максимальному числу коэффициентов в формуле (3) плюс две блочных строки на коэффициенты Паде разложений, которые надо обнулить на текущем шаге.

Обработка координат

С помощью рекуррентных формул последовательно в правой части вычислительной системы проводятся следующие далее действия.

1. Построение $W_{t+1} = A W_t + W_t v_0 + W_{t-1} v_{-1}$, здесь $W_t = Q^{(t)}$, а $A W_t$ обозначает вектор, ненулевые координаты которого в текущем базисе представляют собой сдвинутые на одну позицию вверх координаты вектора W_t . На первых двух шагах алгоритма вычисляем элементы напрямую. Полученный новый блок

Вертикальный размер этой матрицы равен $2k + 3$ блокам. Отметим, что при $t = k + 1$ строки, отмеченные подчеркиванием, следует опустить, а элементы той же матрицы, помеченные точками, заменить нулевыми матрицами.

Решая однородную систему в верхних двух блочных строках, получаем коэффициенты рекуррентных формул v_i , а затем коэффициенты очередных приближений в текущем базисе. Указанные коэффициенты снова используются в формулах (4) на следующих шагах.

Для блочных векторов длины $2k + 1$, расположенных в этих равенствах справа, начиная с третьей позиции сверху, введем следующие обозначения:

по построению будет ортогонален всем предыдущим. Заметим, что $W_i = Q^{(i)}$ означает, что доортогонализация, необходимая в работе [1], в данном случае не нужна, так как все матрицы размера $s \times \frac{1}{2} s$ считаем невырожденными, ввиду очень большой вероятности этого в большом простом поле.

2. Вычисление $(W_{i+1}^T A W_{i+1})^{-1}$ с помощью параллельного алгоритма и решения s линейных систем с правыми частями в виде базисных единичных векторов.

3. Вычисление $W_{i+1}^T B = v_0^T W_i^T B + v_{-1}^T W_{i-1}^T B$, здесь используется то, что $W_i^T A B = 0$ по построению, а на первых двух шагах алгоритма вычисления проводятся напямую.

Элементы $(W_i^T A W_i)^{-1}$, W_i , $W_i^T B$ запоминаются на правой части вычислительной системы. Затем проводится синхронизация, суть которой заключается в описанных далее действиях.

Считаем, что матрица $Q_{k,k}^{(2k)}$, а также матрица τ_{2k} в представлении

$$\alpha(\lambda) Q^{(2k-1)}(\lambda) - P^{(2k-1)}(\lambda) = \sum_{i=2k}^{\infty} \tau_i \lambda^{-i}, \quad (6)$$

которые вычисляются по формулам (2), (4) — невырождены.

На самом деле в этой версии алгоритма на каждом шаге нужна невырожденность старших коэффициентов разложения в правых частях равенств (6) для решения линейных систем в верхних двух блочных строках системы (5), будем ее предполагать, ввиду большой вероятности этого события при работе в большом простом поле.

Перевычисляется решение

$$X = X + A_1 \sum_{i=k+1}^{2k} W_i (W_i^T A W_i)^{-1} W_i^T B = X + A_1 X, \quad (7)$$

где матрица A_1 состоит из блочных векторов (1). Для этого слагаемые правой суммы вычисляются на правой части вычислительной системы и передаются в левую часть построчно, каждая строка к столбцу с тем же номером. Сложение происходит по методу циклической пересылки по медленным горизонтальным каналам.

Аналогично строятся новые образующие блоки по формулам

$$Q^{(i)}(A, B) = A_i Q^{(i)}, \quad i = 2k-1, 2k, \quad (8)$$

с последующей раздачей результата по столбцам на s/l вычислительных узлов левой части вычислительной системы.

По формулам (3) на правой части вычислительной системы при $i = 2k, 2k-1$ вычисляются k старших коэффициентов многочленов $Q^{(2k)}(\lambda), Q^{(2k-1)}(\lambda)$ из k старших коэффициентов многочленов предыдущих образующих для вычислений по формулам (2).

Далее на левой части вычислительной системы вычисляются левые части формул (2). Для этого на каждом шаге после вычисления очередного вектора с номером i по формуле (1) узлы фиксированного столбца проводится умножение на него слева матрицами

$(Q^{(k-1)}(A, B))^T, (Q^{(k)}(A, B))^T$, которые хранятся на них распределенно по вертикали на l одинаковых частей.

Рассмотренная версия алгоритма, основы которого были предложены в работе [1], делает практически те же действия, что и алгоритм Видемана-Копперсмита [7], но частями для сравнительно небольших отрезков базиса пространства Крылова. Это позволяет не проводить повторного вычисления, а запоминать. Уже использованные для построения решения отрезки этого базиса забываются. Но важно то, что это позволяет избежать работы с многочленами большой степени, как следствие, приводит не только к экономии памяти, но и к увеличению скорости работы. В качестве платы за эти дополнительные удобства выступает необходимость периодически проводить "синхронизацию", т. е. фактически собирать результат умножения матрицы на блок. Однако оптимальное число необходимых синхронизаций, как будет показано ниже, невелико, и для текущих размеров задач составляет чуть больше 100.

Работа левой части вычислительной системы

Прежде чем перейти к формулировкам теорем, отметим следующее: в последовательных реализациях указанных алгоритмов максимальное время тратится на многократно повторяющуюся операцию умножения матрицы на блок векторов. Поэтому распараллеливание должно быть прежде всего применено к этой операции.

Можно рассматривать два подхода к распараллеливанию умножения матрицы на блок векторов, связанных с распределенным хранением самой матрицы и (или) распределенным хранением блока.

Необходимость использования первого из отмеченных подходов продиктована также невозможностью хранения в оперативной памяти одного вычислительного узла всей исследуемой матрицы. Для использования второго подхода рассмотрим блок из s векторов, а именно — будем считать, что во всех рассматриваемых алгоритмах решается система линейных уравнений вида $DX = 0$, где $D \in F^{N \times N}$, $X \in F^{N \times s}$.

Рассмотрим задачу параллельного распределенного умножения разреженной матрицы на блочный вектор.

Для обеспечения баланса нагрузки необходимо делить разреженную матрицу на части, содержащие приблизительно одинаковое число единиц. Таким образом, без дополнительных преобразований матрицу можно разрезать только в одном направлении, по соотношениям. Разделим матрицу $D \in F^{N \times N}$ рассматриваемой системы ($A = D^T D$) линейных однородных уравнений на горизонтальные полосы по числу l используемых вычислительных узлов. Таким образом, i -му процессору достанется матрица D_i , в которой ненулевыми оставлены лишь соответствующие N/l строк матрицы D .

На первом этапе нового алгоритма требуется вычислить $(D^T D)^i B$, $B^T (D^T D)^i B$, $B \in F^{N \times s}$, $i = 1, 2, \dots$. Тогда

$$D^T D = \sum_{i=1}^l D_i^T D_i; \quad D^T D B_j = \sum_{i=1}^l D_i^T D_i B_j, \quad j = 1, \dots, s, \quad (9)$$

где $B_j \in F^N$ — это j -я вертикальная полоска блока $B \in F^{N \times s}$. Для хранения этой полоски и матрицы D_i на соответствующем вычислительном узле потребуется память $Nn + \frac{nNd}{l}$ бит (считаем, что для каждого ненулевого элемента матрицы D_i хранится его номер строки, номер столбца и содержание в одном "большом" машинном слове длины n).

Таким образом, распределенное вычисление блочного вектора $D^T DB$ на ls вычислительных узлах требует $\frac{N}{l}d$ операций в F для вычисления элементов $D_i B_j$ и еще $\frac{Nd}{l}$, по числу ненулевых элементов

в матрице D_i^T , для завершения вычисления $D_i^T D_i B_j \in F^N$. Отметим здесь, что для первого умножения (на D_i) при такой схеме расчетов не нужны пересылки, поэтому оно занимает существенно меньше времени, чем второе, требующее получения вектора AB_j . Для этого требуется пересылка со сложением по формуле (9) и обратная рассылка полученной левой части этого равенства на все l задействованных в ее вычислении вычислительных узлов. Используя алгоритм циклической пересылки [6] для вычисления сложения в формуле (9), получим общий объем последовательных пересылок машинных слов со сложением не более $\frac{c_1}{n} 2Nn = 2Nc_1$. Действительно,

в соответствии с алгоритмом циклической пересылки, для получения итоговой суммы на каждом вычислительном узле группы требуется, чтобы этот узел дважды получил и передал вектор, занимающий память с размерами не более чем $N \times 1$ по двустороннему симметричному каналу связи.

Суммируя, получим итоговую оценку времени на вычисление $(D^T D)B$ на ls вычислительных узлах:

$$\frac{2Nd}{l} + 2Nc_1. \quad (10)$$

В предлагаемой версии распределенного вычисления векторов $(D^T D)^i B$ на каждом из l вычислительных узлов j -й группы, $1 \leq j \leq s$, в некоторый момент содержится блочный вектор $(D^T D)^i B_j$, $i \in \{1, \dots, k\}$. Совокупность этих векторов будем называть куском пространства Крылова. При вычислении этого куска, в случае нового алгоритма, все их можно хранить распределенно на вычислительных узлах группы с номером j по $\frac{N}{l}$ строк на каждом.

Поскольку все векторы $(D^T D)^i B_j$, $i \in \{1, \dots, k\}$ в ходе вычислений появляются на каждом узле группы, часть их строк для этого можно просто забыть.

Для распределенного по строкам (полоскам) хранения на l вычислительных узлах всего блока B^T потребуется еще $\frac{Nns}{l}$ бит. После вычисления вектор-столбца $(D^T D)^i B_j$ на каждом из l вычислительных

узлов j -й группы вычисляется своя часть скалярного произведения $B^T (D^T D)^i B_j$ не более, чем за $\frac{Ns}{l}$ операций в F .

Все результаты вычисления j -й группы после concatenation на некотором вычислительном узле правой части вычислительной системы, содержащем k старших коэффициентов $Q_j^{(i)}$, дают $B^T (D^T D)^i B_j$, которые и будем использовать при решении уравнений (2) относительно столбцов переменных μ_i на правой части вычислительной системы (т. е. $\geq s$). Предварительно на l вычислительных узлах правой части вычислительной системы параллельно обращаются матрицы $Q_k^{(k)}$, $Q_{k-1}^{(k-1)}$, для подстановки их в систему (2), а затем и матрицы $W_i^T A W_i = Q_i^{(i)T} P_{i+1}$, $i = 1, \dots, k$, для дальнейшей подстановки их в формулу для решения X . Пересылки при этом незначительные ввиду маленьких размеров пересылаемых матриц. Добавляя к оценке (10), получим оценку времени на вычисление очередного блока векторов из пространства Крылова и попарных скалярных произведений на кластере из sl вычислительных узлов:

$$\frac{2dN}{l} + \frac{Ns}{l} + 2Nc_1.$$

Если

$$\max \left\{ \frac{d}{c_1}, \frac{s}{2c_1} \right\} < l, \quad (11)$$

то первое и второе слагаемые этой оценки меньше третьего. Учитывая, что общее число шагов всего алгоритма будет $2\frac{N}{s}$, получаем общую оценку времени работы первого этапа (вычисление пространства Крылова и скалярных произведений):

$$\frac{8N^2 c_1}{s}. \quad (12)$$

Необходимая память на каждом из sl вычислительных узлов левой части вычислительной системы (с учетом распределенного хранения куска пространства Крылова и исходной матрицы D) определяется размерами соответствующих матриц и форматом хранения их элементов, который были введены выше:

$$\begin{array}{ccccccc} \frac{nNd}{l} & Nn & \frac{Nns}{l} & \frac{Nnk}{l} & & & \\ \text{Матрица } D_i & \text{Часть } B^T & \text{Текущее } B_j & \text{Кусок пр. Крылова} & & & \\ & & & & = & & \\ & & & & = & \frac{Nn}{8 \cdot 10^9} \left(1 + \frac{d+s+k}{l} \right) \text{ Гб.} & \end{array}$$

В случае $N \approx 2^{24}$ это составляет примерно $\frac{n}{128} \left(1 + \frac{d+s+k}{l} \right)$ Гб.

После того, как на правой части вычислительной системы будут получены все необходимые коэффициенты, на левой части вычислительной системы вычисляется часть решения по формуле (7) и новые образующие по формуле (8). Пересылки при этом происходят парал-

тельно по медленным каналам. Для загрузки левой части вычислительной системы, занимающейся умножением матрицы на блочный вектор и вычислением скалярных произведений, потребуется всего не более чем

$$c_2 3 \frac{N}{l} s \frac{N}{ks} = 3 \frac{N^2 c_2}{kl} \quad (13)$$

← пересылок на передаче (рассылка частей (по вертикали) векторов $Q^{(k-1)}(A, B)$, $Q^{(k)}(A, B)$ по медленным горизонтальным связям и получение решения по формуле (7)). Сам параллельный пересчет двух образующих и отрезка решения по формулам (7) и (8) с помощью умножения на A_1 , т. е. умножения матрицы из $F^{\frac{N}{l} \times k}$ на матрицу из $F^{k \times s}$ займет $3 \frac{Nks}{l}$, а по всему алгоритму $3 \frac{N^2}{l}$, что выбором l может быть сделано достаточно маленьким, хотя и не зависящим от k . Поскольку параметр l не увеличивает время ни в какой операции рассматриваемого алгоритма, то его можно наращивать, пока имеются свободные вычислительные узлы. Как следствие, теоретически время на выполнение этого последнего в цикле шага можно уменьшить до нуля без ущерба для остальных шагов.

Работа правой части вычислительной системы

Теорема 1. Оценка времени работы параллельной реализации с использованием блочного фактора нового алгоритма при достаточно большом N и достаточно большом числе вычислительных узлов имеет вид

$$48c_1 N^{3/2}$$

Доказательство. Для правой части вычислительной системы возьмем $t = ks$. Для удобства будем рассматривать k групп по s узлов. Пусть на каждой группе вычислительных узлов правой части вычислительной системы при вычислении $t + 1$ -го блока пространства Крылова хранятся коэффициенты $Q_{k,i}^{(T)}$, $Q_{k-1,i}^{(T)}$, $Q_i^{(T)}$ для $T = t, t - 1$, и некоторого фиксированного значения i , а также все μ и γ по мере их вычисления. Поскольку μ и γ используются последовательно, их можно не хранить в одном месте, а получив и использовав, переслать другим группам вычислительных узлов. Для того чтобы хранить все указанные коэффициенты потребуется на ks вычислительных узлах необходимо $O(ns^2)$ свободных бит оперативной памяти на каждой группе из s вычислительных узлов. На втором этапе вычислительные узлы правой части вычислительной системы будут заниматься вычислениями по формулам (2), распределенно по столбцам, и по формуле (3), вычисляя только старший коэффициент $Q_i^{(t)}$ многочлена на каждом шаге и k старших коэффициентов для последних двух многочленов в цикле между синхронизациями. Вычислениями по формулам (4) и нахождением v_i по формуле (5) занимается еще один выделенный узел правой части вычислительной системы, который также вычисляет коэффициенты

при элементах с крышками $((W_i^T A W_i)^{-1} W_i^T B \quad (7))$. При этом обращение и произведение матриц размера $s \times s$ можно проводить параллельно на всех t узлах правой части вычислительной системы. Пересылки при этом незначительные и учитывать их не будем.

Каждая группа с номером i_0 из s узлов правой части вычислительной системы выполняет перечисленные далее операции.

1. На выделенной группе узлов с использованием s узлов правой части вычислительной системы вычисляется вспомогательный коэффициент $(W_{t+1}^T A W_{t+1})^{-1} W_{t+1}^T B$ с помощью обращения матрицы $W_{t+1}^T A W_{t+1} = Q_{t+1}^{(t+1)T} \sigma_{t+2}$ и рекуррентных формул для $W_i^T B$ (см. формальное описание алгоритма). Итого три умножения и одно обращение за время не превышающее $10,5s^2 c_1$.

2. Получает по бинарному дереву коэффициенты рекуррентных формул (два коэффициента v_0, v_{-1}) и вспомогательный коэффициент $(W_{t+1}^T A W_{t+1})^{-1} W_{t+1}^T B$.

Пересчитываем свои (при фиксированном i) коэффициенты $Q_{k,i_0}^{(t+1)}$, $Q_{k-1,i_0}^{(t+1)}$ по рекуррентным формулам (5) (для этого один дополнительный коэффициент получает с соседнего узла, учитывая сдвиг вверх крайнего левого столбца в формуле (5)), и затем свою координату вектора X (см. (7)). Всего не более шести умножений. Произведение матриц $s \times s$ один узел может проводить с использованием s узлов своей группы правой части вычислительной системы за время, не превышающее $3s^2 c_1$ (передать элементы множителей, получить элементы результата).

3. Вычисляет сумму слагаемых в формулах вида (4):

$$\sum_{i=0}^{t+1-k} \mu_{i+j} Q_{k,i}^{(t+1)} + \sum_{i=0}^{t-k} \gamma_{i+j} Q_{k-1,i}^{(t+1)}, \quad j = t + 2, t + 3,$$

содержащих коэффициенты $Q_{k,i_0}^{(t+1)}$, $Q_{k-1,i_0}^{(t+1)}$, которые хранятся на данном узле (четыре умножения).

4. Передает эту сумму на выделенную группу процессоров в правой части вычислительной системы для решения очередных систем линейных однородных уравнений в верхних частях матричных равенств вида (5), а выделенная группа решает указанную систему с использованием всех $t = ks$ узлов правой части вычислительной системы за время, не превышающее $3/2$ от $(3s)^2 c_1$ (привести к треугольному виду и решить на s вычислительных узлах). Здесь подразумевается, что t больше, чем $3s$ (размер системы), так что время на решение с помощью параллельного алгоритма меньше указанного значения. Полученное рассылается обратно на k групп узлов по бинарному дереву. Поскольку эти коэффициенты небольшие их пересылками пренебрежем.

Объем необходимой памяти на одной группе из s узлов правой части вычислительной системы оценивается значением величины $3kns^2$. Здесь учтено по два вычисляемых коэффициента разложений для приближений $Q^{(T)}$, для $T = t, t - 1$, один вспомогательный коэффициент, а также все μ и v (k матриц размера $s \times s$), а также резерв для пересчета. Все μ и v можно не хранить, а получать, использовать (умножать на соответствующие коэффициенты $Q_i^{(T)}$, хранящиеся на i -м вычислительном узле) и пересылать дальше, получая новые, в этом случае память оценивается как $O(ns^2)$.

Соответствующие оценки времени вычислений по перечисленным пунктам и операциям следующие:

- 1) $10,5Nsc_1$;
- 2) $18s^2c_1 \frac{N}{s} = 18Nsc_1$;
- 3) $12s^2c_1 \frac{N}{s} = 12Nsc_1$;
- 4) $\frac{27}{2}s^2c_1 \frac{N}{s} = \frac{27}{2}Nsc_1$.

Складывая все значения, получим следующую оценку времени работы:

$$54Nsc_1. \quad (14)$$

Перед очередной синхронизацией осуществляется пересылка. Поскольку решение и образующие хранятся частями вида $F^{\frac{N}{l}}$, то получаемые для пересчета коэффициенты должны быть в F^{ks} . Пересылка здесь незначительная.

В сумме с (14) и (13) время вычислений не более

$$54Nsc_1 + 3 \frac{N^2c_2}{kl}. \quad (15)$$

Здесь же учтем вычисления μ_j, γ_j по формулам (2) (обращение двух матриц размера $s \times s$ и решение блочнотреугольной линейной системы уже без обращений) при известных левых частях этих формул на правой части вычислительной системы с относительно быстрыми связями по всему алгоритму $\left(2\frac{3}{2}s^2c_1 + 3s^2c_1k2\right) \frac{N}{sk} < 9Nsc_1$. Здесь принят во внимание тот факт, что каждый коэффициент $Q_{i_0}^{(T)}$ используется в системе (2) не более k раз.

Вычисление и обмен, в результате которого узлы, вычисляющие скалярные произведения, полностью получают старшие k коэффициентов многочленов $Q^{(k)}(\lambda)$, аналогично займет не более $(3s^2c_1k) \frac{N}{sk} = 3Nsc_1$.

Общая оценка с учетом выражений (12), (15) — $12Nsc_1 + 54Nsc_1 + 3 \frac{N^2c_2}{kl} + \frac{8N^2c_1}{s}$.

Для минимизации получившейся функции по k возьмем $k > 3c_2$, получим общую оценку времени:

$$66Nsc_1 + \frac{8N^2c_1}{s}. \quad (16)$$

Естественным ограничением сверху на k является l — чтобы число вычислительных узлов на правой части вычислительной системы не стало слишком большим. Это обстоятельство позволяет ограничить память узлов левой части вычислительной системы, на которой хранится кусок пространства Крылова до реальных размеров (см. выше).

Выбрав

$$s = \sqrt{\frac{8N}{66}}, \quad (17)$$

получим $48c_1N^{3/2}$,

Теорема доказана.

В процессе доказательства суммировалось время по двум этапам, хотя для начала второго этапа достаточно первых элементов рассматриваемого отрезка пространства Крылова, что следует из равенств (2), (3), (4). С учетом этого факта первый и второй этапы можно проводить практически параллельно. При согласованной работе узлов, занятых на первом и втором этапах, общее время можно уменьшить примерно вдвое.

Число арифметических операций на втором этапе алгоритма Видемана — Копперсмита-Томе [7] оценено в работе [8] значением

$$O(Nns(ns + \log_2 N) \log_2 N \log_2 \log_2 N).$$

Складывая с (12) и оптимизируя по s , получаем для всего алгоритма оценку

$$O(N^{1+\frac{2}{3}}(\log_2 N \log_2 \log_2 N)^{\frac{1}{3}}).$$

Кроме того, работа программы при постановке рекорда целочисленной факторизации в 2012 г. потребовала использования критического объема оперативной памяти, около 1 ТБ. Это связано с ростом коэффициентов многочленов большой степени при использовании алгоритма быстрого преобразования Фурье для их умножения. Совсем недавно, используя высокопроизводительную технику, группой под руководством Томе были поставлены рекорды целой факторизации (RSA-250) и дискретного логарифмирования по модулю простого числа в 240 десятичных знаков (подробных публикаций пока нет). В рассматриваемом алгоритме произведение многочленов есть, но их степени меньше в число синхронизаций раз. Таким образом, предложенный алгоритм лучше алгоритма Видемана-Копперсмита асимптотически по времени работы и по используемой оперативной памяти.

Здесь следует сделать следующее замечание. Оптимальное число вычислительных узлов получается приблизительно равным $\frac{N}{33c_1}$ (см. (11)), что при ак-

туальных значениях параметров приблизительно равно 13000 узлам. Поэтому совсем не обязательно выбирать s оптимальным согласно формуле (17). Если s выбрано меньше, то оценка времени работы алгоритма (см. (16)) будет $\frac{8N^2c_1}{s}$ при $K = sl + t \approx \frac{s^2}{2c_1}$

(см. (16) при $k = 3c_2, t \approx 3c_2s$).

Заключение

Были получены оптимальные параметры настройки вычислительной системы для успешной работы программы решения большой разреженной системы над большим простым полем. Кроме того, получены оценки времени работы такой программы, лучшие из известных на сегодняшний момент.

Список литературы

1. Черепнёв М. А. Блочный алгоритм типа Ланцоша решения разреженных систем линейных уравнений // Дискретная математика. — 2008. — Т. 20, № 1. — С. 145—150.
2. Черепнёв М. А., Замарашкин Н. Л. Универсальный блочный метод Ланцоша—Паде для систем линейных уравнений над

большими простыми полями // Фундаментальная и прикладная математика. — 2014. — Т. 19, Вып. 6. — С. 223–247.

3. Montgomery P. L. A Block Lanczos Algorithm for Finding Dependencies over GF(2) // Advances in Cryptology — EuroCrypt'95 / Eds by L. C. Guillou, J.-J. Quisquater, Berlin: Springer-Verlag, Lect. Notes in Comp. Sci. 1995. — Vol. 921, P. 106–120.

4. Cherepnev M. A. Some estimations of performance of parallel algorithms for solving large linear systems over GF(2) // A Journal of Tambov State University, The works of participants of International conference "ParCA" presented according to the results of reviewing by International Program Committee. — 2010. — Vol. 15, Iss. 4. — P. 134–1353.

5. Черепнев М. А. О некоторых вычислениях в пространствах Крылова над GF(2) // Вестник Тамбовского университета Сер. Естественные и технические науки. — 2009. — Т. 14/ Вып. 4. — С. 833–835.

6. Barnett M., Littlefield R., Payne D. G., van de Geijn R. A. Global combine algorithms on mesh architectures with wormhole routing // Int. Par. Processing Symp. 1993-Apr. P. 156–162.

7. Coppersmith D. Solving homogeneous linear systems over GF(2) via block Wiedemann algorithm // Mathematics of Computation. — 1994. — Vol. 62, No. 205. — P. 333–350.

8. Thome E. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm // Journal of Symbolic Computation. — 2002. — Vol. 33, No. 5. — P. 757–775.

Estimation of the Speed of a New Parallel Block Algorithm for Finding Solutions of the Large Sparse Systems over a Large Prime Field

M. A. Cherepnev, Ph. D., Associate Professor, cherepniiov@gmail.com, Lomonosov Moscow State University, Moscow, 119992, Russian Federation

Corresponding author:

Cherepnev Michail A., Ph. D., Associate Professor, Lomonosov Moscow State University, Moscow, 119992, Russian Federation
E-mail: cherepniiov@gmail.com

Received on June 17, 2020

Accepted on June 22, 2020

The question of how to estimate the running time of a program that implements an algorithm for solving a large sparse system of equations over a large simple field, described in [1,2], on a computer system of K independent computing nodes is considered. The article considers the topology that connects nodes of the computer network, "tor". At the same time, we believe that in each of the two directions of this torus, there are connections with different bandwidth. In reality, this may correspond to large computing clusters connected by a relatively slow Internet network, or multiple multi-core (possibly with virtual cores) processors in the same cluster. At the same time, the estimates of working time take into account not only the time for the invoice, as in [2], but also the time for forwarding. As a result, the overall time with the growth of the number of computing nodes decreases not as fast as the number of operations in [2]. The block nature of the algorithm allows us to reduce the running time of the sequential scalar version (Montgomery's algorithm [3]) by approximately $\sqrt{K}$ times, with K growing to the size of the original problem. At the same time, about half of all switching is allowed to be slow, which makes it possible to dial the necessary number of computing nodes from different clusters. A specific distributed storage of intermediate results of calculations is proposed. This results in saving not only computing resources, but also time for transmission, which is carried out in parallel over independent branches of our network. In addition, the solution of dense systems of linear equations, which must be done at each step of our algorithm, as well as the inversion of dense matrices, we suggest using parallel calculations. This will reduce the conversion time of matrices of size $s \times s$ to $O(s^2)$ and the total running time of the entire program for solving a sparse system of linear equations in a space of dimension N to $O\left(\frac{N^2}{\sqrt{K}}\right)$ for an arbitrary K, from 1 to N. Thus, theoretically, the optimal time $O(N^{3/2})$ is obtained.

Keywords: discrete logarithm method, Lanczos method, parallel computing, block algorithms

Acknowledgements: This work was supported by the Russian Foundation for Basic Research, project nos. 18-29-03421 МК.

For citation:

Cherepnev M. A. Estimation of the Speed of a New Parallel Block Algorithm for Finding Solutions of the Large Sparse Systems over a Large Prime Field, *Programnaya Ingeneria*, 2020, vol. 11, no. 4, pp. 242–250

DOI: 10.17587/prin.11.242-250

References

1. Cherepnev M. A. Block algorithm of lanzosh type for solving sparse systems of linear equations, *Diskretnaya matematika*, 2008, vol. 20, no. 1, pp. 145–150 (in Russian).

2. Cherepnev M. A., Zamarashkin N. L. Universal block method of lanzosh-Pade for systems of linear equations over large simple fields, *Journal of Mathematical Sciences*, 2017, vol. 221, no. 3, pp. 461–478.

3. Montgomery P. L. A Block Lanczos Algorithm for Finding Dependencies over GF(2), *Advances in Cryptology — EuroCrypt'95* / Eds by L. C. Guillou, J.-J. Quisquater, Berlin: Springer-Verlag, Lect. Notes in Comp. Sci., 1995, vol. 921, pp. 106–120.

4. Cherepnev M. A. Some estimations of performance of parallel algorithms for solving large linear systems over GF(2), *A Journal of Tambov State University, The works of participants of International conference "ParCA" presented according to the results of reviewing by International Program Committee*, 2010, vol. 15, iss. 4, pp. 1342–1353.

5. Cherepnev M. A. Some calculations in Krylov space over GF(2). *Vestnik Tambovskogo universiteta Ser. Estestvennye i tekhnicheskie nauki*, 2009, vol. 14, no. 4, pp. 833–835 (in Russian).

6. Barnett M., Littlefield R., Payne D. G., van de Geijn R. A. Global combine algorithms on mesh architectures with wormhole routing, *Int. Par. Processing Symp.*, 1993-Apr., pp. 156–162.

7. Coppersmith D. Solving homogeneous linear systems over GF(2) via block Wiedemann algorithm, *Mathematics of Computation*, 1994, vol. 62, no. 205, pp. 333–350.

8. Thome E. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm, *Journal of Symbolic Computation*, 2002, vol. 33, no. 5, pp. 757–775.