2013 Вычислительные методы в дискретной математике №3(21)

DOI 10.17223/20710410/21/12

УДК 519.7

О РЕАЛИЗАЦИИ АЛГОРИТМА КОППЕРСМИТА ДЛЯ ДВОИЧНЫХ МАТРИЧНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ НА ВЫЧИСЛИТЕЛЯХ КЛАСТЕРНОГО ТИПА

А. С. Рыжов

Лаборатория ТВП, г. Москва, Россия

E-mail: ryzhovalexander@mail.ru

Рассматривается задача реализации алгоритма Копперсмита, вычисляющего векторные аннулирующие многочлены для матричных последовательностей, на современных 64-разрядных ЭВМ. Рассмотрены вопросы представления данных для случая последовательностей бинарных матриц с точки зрения снижения трудоёмкости алгоритма. Предложены способы эффективного распараллеливания алгоритма для реализации на ЭВМ с многоядерными процессорами, а также для выполнения алгоритма на вычислителях кластерного типа.

Ключевые слова: *матричные последовательности, алгоритм Копперсмита.*

Введение

Во многих алгоритмах вычислительной алгебры возникает задача нахождения решений сильно разреженной системы однородных линейных уравнений (СОЛУ). Например, при нахождении дискретного логарифма в поле $\mathrm{GF}(2^n)$ (при $p = 2^n - 1$ — простом) методом Копперсмита [1] необходимо найти решения такой системы над простым полем $\mathbb{Z}_p$.

В настоящий момент для решения данной задачи обычно используется один из двух алгоритмов: алгоритм Ланцоша — Монтгомери [2] или алгоритм Видемана — Копперсмита [3]. Оба алгоритма имеют свои преимущества и недостатки, а также ограничения на используемые вычислители. Алгоритм Ланцоша — Монтгомери реализован, например, в пакете GGNFS. Он ориентирован на выполнение на одном вычислителе кластерного типа с высокоскоростным внутренним соединением узлов. Алгоритм Видемана — Копперсмита, напротив, позволяет наиболее трудоёмкие этапы выполнять на нескольких не связанных друг с другом кластерах.

Блочный алгоритм Видемана — Копперсмита является обобщением алгоритма Видемана поиска решения разреженной СОЛУ [4], ориентированным на вычислительную технику, работающую со словами в несколько байтов. Один из этапов алгоритма Видемана — нахождение аннулирующего многочлена двоичной последовательности с помощью алгоритма Берлекэмпа — Мэсси, и его непосредственное обобщение на многомерный случай является, по-видимому, трудной задачей. Копперсмит решил эту проблему, сведя задачу к поиску векторных приближений Паде и разработав новый алгоритм для построения таких приближений.

В данной работе рассматриваются вопросы реализации алгоритма Копперсмита построения векторных приближений Паде при решении СОЛУ над полем $\mathrm{GF}(2)$. Приводится описание алгоритма Видемана — Копперсмита нахождения решений разреженных СОЛУ, подробное описание метода Копперсмита построения векторных приближений Паде, особенности реализации этого алгоритма на ЭВМ. Отдельно рассмотрены особенности реализации алгоритма Копперсмита на вычислителях кластерного типа.

1. Блочный алгоритм Видемана — Копперсмита решения систем однородных линейных уравнений

Описание алгоритма Видемана — Копперсмита нахождения решений разреженных систем однородных линейных уравнений можно найти, например, в [3]. Приведём основные шаги алгоритма, чтобы показать, где используется алгоритм Копперсмита построения векторных приближений Паде. Пусть $P = \mathrm{GF}(2)$ — поле из двух элементов, A — матрица размеров $M \times N$ над полем P, и пусть необходимо найти нетривиальное (т. е. ненулевое) решение СОЛУ

$$Ax = 0. \tag{1}$$

Заметим, что не ставится задача нахождения всех решений системы (1) или приведения матрицы A к специальному виду. Чтобы гарантировать наличие нетривиальных решений, будем считать, что $M < N$.

Пусть $m, n \in \mathbb{N}$ — параметры алгоритма, $m \geqslant n$. Пусть, далее, $Z \in P_{m,M}$ — случайная матрица размера $m \times M$ над полем P, $B \in P_{M,M}$ — матрица из первых M столбцов матрицы A решаемой СОЛУ (1), $Y \in P_{M,n}$ — матрица, составленная из столбцов матрицы A с номерами $M+1, M+2, \ldots, M+n$ (пусть $M + n \leqslant N$). Последовательно выполняем следующие три этапа:

1) Вычислим первые $L+1$ элементов последовательности матриц $a = a_0 a_1 \ldots$ размера $m \times n$ над полем P, где $a_i = ZB^iY$ (здесь L — параметр метода; см. ниже).
2) Найдём *векторный аннулирующий многочлен* для последовательности a, т. е. такие векторы $f_0, \ldots, f_d \in P^n$, что $\sum\limits_{j=0}^{d} a_{i+j} f_j = 0 \in P^{(m)}$ для всех $i \geqslant 0$.
3) Из последнего равенства получаем, что вектор $\sum\limits_{j=0}^{d} B^j Y f_j$ лежит в ядре любой матрицы ZB^i, $i = 0, 1, \ldots$ Отсюда с большой вероятностью $\sum\limits_{j=0}^{d} B^j Y f_j = 0$, т. е. $B \sum\limits_{j=0}^{d-1} B^j Y f_{j+1} + Y f_0 = 0$. Следовательно, вектор-столбец $w = \begin{pmatrix} \sum\limits_{j=0}^{d-1} B^j Y f_{j+1} \\ f_0 \end{pmatrix}$, дополненный нулями до длины N, является решением системы (1) (напомним, Y состоит из столбцов матрицы A, идущими после подматрицы B). Если вектор w равен нулю, то $\sum\limits_{j=0}^{d-1} B^j Y f_{j+1} = 0$, и выполняются аналогичные рассуждения.

Замечание 1. Необходимо отметить, что для последовательности a указанного вида векторный аннулирующий многочлен, вычисляемый на втором этапе алгоритма, всегда существует. При $m = n = 1$ таким многочленом является любой аннулирующий многочлен матрицы B — например характеристический или минимальный. В общем случае равенство $\sum\limits_{j=0}^{d} B^j Y f_j = 0$ можно рассматривать как систему из M однородных линейных уравнений от $(d+1)n$ неизвестных элементов векторов $\{f_j\}$ над полем P, и при $d \geqslant M/n$ эта система имеет ненулевое решение — искомый векторный аннулирующий многочлен.

Исходный алгоритм Видемана, опубликованный в работе [4], является частным случаем приведённого выше блочного алгоритма Видемана — Копперсмита при $m = n = 1$. В алгоритме Видемана на втором этапе необходимо найти аннулирующий многочлен

последовательности $a = a_0 a_1 \ldots$ элементов поля P, при этом достаточно вычислить первые $2M + 1$ элементов, так как ранг последовательности не превосходит ранг матрицы B. Для поиска аннулирующего многочлена можно использовать, например, алгоритм Берлекэмпа — Мэсси.

Для нахождения векторного аннулирующего многочлена последовательности матриц Копперсмит предложил новый алгоритм, основанный на последовательном вычислении приближений Паде. При этом в результате работы алгоритма может получиться до n различных независимых аннулирующих многочленов, что в результате дает до n линейно независимых решений системы (1).

Для нахождения векторного аннулирующего многочлена достаточно иметь первые $L + 1$ элементов последовательности a, где $L = \lfloor M/m \rfloor + \lfloor M/n \rfloor + \Delta$, $\Delta \approx 100$. Это следует из набора линейных соотношений, а также из теоретико-вероятностных соображений [3].

По сравнению с методом Видемана, в алгоритме Видемана — Копперсмита на первом и третьем этапах требуется выполнить существенно меньшее число итераций. Для удобства реализации на современных ЭВМ числа m и n выбирают, как правило, кратными длине машинного слова: например, $m = 128$, $n = 64$. Расчет трудоёмкости приведён автором метода в [3]. Алгоритм дает выигрыш только в случае сильной разреженности матрицы системы: для первого и третьего этапов трудоёмкость пропорциональна $M^2 w$, где w — средний вес строки матрицы B, т. е. среднее число ненулевых элементов в строке. Трудоёмкость второго этапа при использовании алгоритма Копперсмита равна $\mathrm{O}(M^2 m)$.

Далее приводится подробное изложение алгоритма Копперсмита и рассматриваются вопросы, связанные с его эффективной реализацией на вычислителях кластерного типа.

2. Алгоритм Копперсмита вычисления векторных аннулирующих многочленов

Описание алгоритма Копперсмита вычисления векторных аннулирующих многочленов приведено автором алгоритма в [3]. Алгоритм сформулирован достаточно громоздко в терминах отдельных элементов коэффициентов степенных рядов над кольцом матриц. Здесь приводится описание алгоритма, в основном схожее с описанием в работе [5] и более удобное для дальнейшего изложения.

Итак, для последовательности матриц над конечным полем (не обязательно $\mathrm{GF}(2)$) необходимо найти векторный аннулирующий многочлен. Поскольку алгоритм Копперсмита, как было сказано ранее, выдаёт сразу несколько решений, будем рассматривать задачу нахождения *матричного аннулирующего многочлена*: для последовательности $a = a_0 a_1 \ldots$ матриц размера $m \times n$ над конечным полем P необходимо найти матрицы $f_0, \ldots, f_d \in P_{n,r}$ размера $n \times r$, одновременно не равные нулю, такие, что

$$\forall i \geqslant 0 \ \sum_{j=0}^{d} a_{i+j} f_j = 0_{m \times r}. \tag{2}$$

Будем изначально предполагать существование решения.

Замечание 2. Последовательность $a = a_0 a_1 \ldots$, обладающую свойством (2), в общем случае нельзя называть линейной рекуррентной последовательностью, поскольку при $m \neq n$ множество возможных значений коэффициентов этой последовательности не образует структуру кольца. Нельзя также говорить о многочлене вида $f_0 x^d + f_1 x^{d-1} + \cdots + f_d$. Тем не менее далее соответствующие объекты будем

называть многочленами и степенными рядами для удобства и наглядности изложения. Например, будем говорить «степенной ряд $H(x) = \sum a_i x^i$» или «многочлен $F(x) = f_0 x^d + f_1 x^{d-1} + \cdots + f_d$», а также рассматривать их «произведение», если размеры коэффициентов допускают перемножение. При этом какие-либо алгебраические свойства этих объектов использоваться не будут.

Замечание 3. Как было отмечено выше, при решении исходной задачи — нахождения решений СОЛУ (1) — построенная последовательность $a = (ZB^iY : i = 0, 1, \ldots)$ заведомо обладает свойством (2), поскольку система $\sum\limits_{j=0}^{d} B^jY f_j = 0$, состоящая из rM уравнений от $r(d+1)n$ неизвестных (коэффициентов матриц $f_0, \ldots, f_d$) при $d \geqslant M/n$ имеет ненулевое решение.

Рассмотрим произведение ряда $H(x) = \sum a_i x^i$ и многочлена $F(x) = f_0 x^d + f_1 x^{d-1} + + \cdots + f_d$:

$$H(x)F(x) = \sum_{i=0}^{\infty} \sum_{j=0}^{d} a_i f_j x^{i+d-j} = \sum_{k=0}^{\infty} \left(\sum_{j=0}^{d} a_{k-d+j} f_j \right) x^k$$

(здесь $a_i = 0$ при $i < 0$). Ясно, что если выполняется условие (2), то ряд $H(x)F(x)$ является многочленом степени не выше $d-1$, то есть его коэффициенты при $x^d, x^{d+1}, \ldots$ равны нулю. При этом $F(x)$ называется *правым порождающим многочленом* последовательности a. Обратно, если для некоторых многочленов $F(x) \in P_{n,r}[x], Q(x) \in P_{m,r}[x]$ выполняется $H(x)F(x) = Q(x)$, то коэффициенты многочлена $F(x)$ являются решением поставленной задачи.

Если для некоторых многочленов $F(x)$, $Q(x)$ и рядов $H(x)$, $C(x)$ выполняется равенство

$$H(x)F(x) = Q(x) + x^t C(x), \tag{3}$$

то многочлены F и Q называют *приближениями Паде* порядка t для степенного ряда $H(x)$. Алгоритм Копперсмита итеративно строит приближения Паде порядка $t = t_0, t_0 + 1, \ldots$ Как было оговорено выше, последовательность a обладает правым порождающим многочленом, поэтому при достаточно большом t получим правый порождающий многочлен этой последовательности.

Приведём описание алгоритма Копперсмита. Через E_m будем обозначать единичную матрицу размера $m \times m$. Положим $A(x) = (H(x)| - E_m)_{m\times(m+n)}$, $G_t(x) = \begin{pmatrix} F_t(x) \\ Q_t(x) \end{pmatrix}_{(m+n)\times(m+n)}$. Тогда $A(x)G_t(x) = H(x)F_t(x) - Q_t(x)$ и равенство (3) равносильно равенству $A(x)G_t(x) = x^t C_t(x)$ (здесь $C(x)$ из равенства (3) заменено на $C_t(x)$, чтобы показать, что этот матричный многочлен определяется номером шага алгоритма $t = 0, 1, 2, \ldots$). Для столбцов матрицы $G_t(x) = \big(G_{t,1}(x), \ldots, G_{t,(m+n)}(x)\big)$ введём целочисленные векторы $\delta_t = (\delta_{t,1}, \ldots, \delta_{t,(m+n)})$, элементы которых вычисляются в процессе работы алгоритма и в некотором роде соответствуют степеням этих векторных многочленов. (Под степенью столбца матрицы над кольцом многочленов будем понимать максимум степеней многочленов, являющихся элементами этого столбца: $\deg\,(g_1(x), \ldots, g_k(x))^T = \max\limits_{i\in\{1,\ldots,k\}} \{\deg g_i(x)\}$.) А именно:

— степень столбца $G_{t,j}(x)$ не превосходит соответствующее значение $\delta_{t,j}$;
— при сложении двух столбцов $G_{t,j_1}(x)$ и $G_{t,j_2}(x)$ результату будет соответствовать максимум их значений $\max\{\delta_{t,j_1}, \delta_{t,j_2}\}$;
— при умножении столбца на x соответствующее ему значение увеличивается на 1.

На каждом шаге алгоритма Копперсмита последовательно вычисляются пары $(G_t(x), C_t(x))$, для которых выполняются следующие три условия:

$$A(x)G_t(x) = x^t C_t(x); \tag{4}$$

$$\operatorname{rank} C_t(0) = m; \tag{5}$$

$$\deg G_{t,j}(x) \leqslant \delta_{t,j},\ j = 1, 2, \ldots, m+n,\ \sum_{j=1}^{m+n} \delta_{t,j} = tm. \tag{6}$$

Инициализация значений $(G_0(x), C_0(x))$ выполняется следующим образом: $G_0(x) = E_{(m+n)\times(m+n)}$, $C_0(x) = (H(x)|E_m)$. Значения $\delta_0 = (\delta_{0,1}, \ldots, \delta_{0,m+n})$ задаются равными нулю. Выполнение условий (4) – (6) для $t = 0$ очевидно.

При выполнении очередного шага t необходимо так изменить матрицу $C_t(x)$, чтобы можно было выделить одну степень x. Для этого достаточно n столбцов младшего коэффициента (т. е. $C_t(0)$) сделать нулевыми, а остальные m столбцов домножить на x.

Формально, на шаге t по паре $(G_t(x), C_t(x))$ вычисляется пара $(G_{t+1}(x), C_{t+1}(x))$ следующим образом:

1) Вычислим матрицу перехода τ_t размера $(m+n)\times(m+n)$, которая представляет собой невырожденные линейные преобразования столбцов, приводящие матрицу $C_t(0)$ к ступенчатому виду: $C_t(0)\tau_t = (0|E)$. При этом к столбцу $G_{t,j}(x)$ можно прибавлять только те столбцы $G_{t,j_2}(x)$, соответствующие значения δ_{t,j_2} которых не больше $\delta_{t,j}$ (этого можно добиться за счёт перестановки столбцов; при этом значения $\delta_{t,j}$ также переставляются).
2) Вычислим $G_{t+1}(x) = G_t(x)\tau_t D$, где $D = \operatorname{diag}(1, \ldots, 1, x, \ldots, x)$ (единицы стоят на первых n элементах диагонали).
3) Вычислим $C_{t+1}(x) = C_t(x)\tau_t D/x$.
4) Положим $\delta_{t+1,j} = \begin{cases} \delta_{t,j}, & \text{если } j \in \{1, \ldots, n\} \\ \delta_{t,j} + 1, & \text{если } j \in \{n+1, \ldots, m+n\}. \end{cases}$

Заметим, что вычисление матрицы τ_t возможно в силу выполнения условия (5). Деление на x при вычислении $C_{t+1}(x)$ возможно в силу того, что первые n столбцов матрицы $C_t(0)\tau_t$ равны нулю, т. е. делятся на x, а оставшиеся m столбцов домножаются на x при умножении на матрицу D.

Проверим выполнение условий (4) – (6). Выполнение условия (4) для $t+1$ очевидно (если оно выполняется для t): левая и правая части равенства (4) для t домножены справа на одну и ту же полиномиальную матрицу $\tau_t D$. Условие (5) для $t+1$ следует из того, что последние m столбцов матрицы $C_t(0)\tau_t$ являются единичными. Условие (6) выполняется в силу особенностей построения матрицы перехода τ_t, а также умножения $G_t(x)\tau_t$ на матрицу D, содержащую ровно m элементов x на диагонали.

Признаком останова алгоритма является наличие в матрице $G_t(x)$ достаточного (требуемого) числа столбцов $G_{t,j}(x)$, для которых выполняется неравенство

$$t - \delta_{t,j} > \frac{N}{m} + \Delta_2, \tag{7}$$

где Δ_2 — наперёд заданный постоянный параметр; $\Delta_2 \approx 10 - 100$. Если необходимо найти n решений, то в силу условия (6) останов выполнится по крайней мере через $t \sim N/m + N/n$ шагов.

Столбцы, для которых выполняется условие (7), представляют собой решение задачи. Поскольку нас интересуют только сами правые порождающие многочлены $F_{t,j}(x)$, нет необходимости вычислять на каждом шаге всю матрицу $G_t(x) =$

$= \begin{pmatrix} F_t(x) \\ Q_t(x) \end{pmatrix}_{(m+n)\times(m+n)}$, а достаточно выполнять действия только с первыми n строками.

Таким образом, на каждом шаге необходимо выполнить следующие действия:

— с помощью невырожденных линейных преобразований столбцов привести матрицу $C_t(0)$ к ступенчатому виду $C_t(0)\tau_t = (0|E)$;
— умножить полиномиальную матрицу $G_t(x)$ на матрицу $\tau_t D$;
— умножить полиномиальную матрицу $C_t(x)$ на $\tau_t D/x$.

Замечание 4. За счёт выбора матрицы $\tau_t D$ на каждом шаге ровно m столбцов матрицы $G(x)$ увеличивают свою степень. Именно это позволяет к определённому моменту набрать достаточное число столбцов, удовлетворяющих условию (7).

Оценим трудоёмкость алгоритма Копперсмита. На каждом шаге наиболее трудоёмким действием является умножение полиномиальных матриц $G_t(x)$ и $C_t(x)$ на скалярную матрицу перехода τ_t. Степень матрицы $G_t(x)$, очевидно, не превосходит t. Степень матрицы $C_t(x)$ не превосходит $L - t$, где $L = \deg C_0(x) = \deg H(x)$ — длина отрезка последовательности, для которой ищется правый порождающий многочлен. Таким образом, при $m \geqslant n$ трудоёмкость алгоритма имеет порядок $\mathrm{O}(L^2 m^3)$. При $L = \lfloor M/m \rfloor + \lfloor M/n \rfloor + \Delta$, как в алгоритме Видемана — Копперсмита нахождения решений СОЛУ, получаем трудоёмкость порядка $\mathrm{O}(M^2 m)$.

3. Вопросы реализации алгоритма Копперсмита на ЭВМ

Рассмотрим некоторые подходы, позволяющие эффективно реализовать алгоритм Копперсмита нахождения векторных аннулирующих многочленов матричной последовательности над полем $\mathrm{GF}(2)$ на современной 64-разрядной вычислительной технике. Будем считать, что размеры последовательности m и n кратны 64: $m = 64m'$, $n = 64n'$.

3.1. Представление данных

С точки зрения оптимизации реализации алгоритма на ЭВМ большую роль играет способ представления данных. Естественным выбором является представление, позволяющее упростить наиболее трудоёмкие операции, выполняемые на каждом шаге алгоритма. Кроме того, при использовании вычислителей кластерного типа необходимо минимизировать объём данных, которыми обмениваются узлы кластера на каждом шаге.

Наиболее трудоёмкими операциями, выполняемыми на каждом шаге алгоритма Копперсмита, являются:

— умножение (справа) матрицы $C_t(x)$ на матрицу $\tau_t D/x$;
— умножение (справа) матрицы $G_t(x)$ на матрицу $\tau_t D$.

Выделение нулевого коэффициента $C_t(0)$ полиномиальной матрицы $C_t(x)$ при любом её естественном представлении не составляет труда. Операция приведения $C_t(0)$ при небольших значениях m и n также не является трудоёмкой по сравнению с умножением матриц $C_t(x)$ и $G_t(x)$ на матрицу перехода τ_t. Вектор степеней $\delta_{t+1} = (\delta_{t+1,1}, \ldots, \delta_{t+1,m+n})$ вычисляется за $\mathrm{O}(m+n)$ операций.

Матрица перехода τ_t является скалярной матрицей, поэтому умножение полиномиальных матриц $C_t(x)$ и $G_t(x)$ на τ_t справа сводится к отдельному умножению каждой строки каждого коэффициента этих матриц на τ_t. Поэтому с точки зрения снижения трудоёмкости этой операции, а также с учётом использования 64-разрядной ЭВМ естественно представлять полиномиальные матрицы $C_t(x)$ и $G_t(x)$ в виде массивов из m и n (соответственно) полиномиальных строк (длины этих массивов соответству-

ют степеням матриц $C_t(x)$ и $G_t(x)$), каждую полиномиальную строку — в виде массива строк-коэффициентов, а каждую строку-коэффициент (длины $m+n=64(m'+n')$) — в виде массива из $m'+n'$ 64-разрядных машинных слов.

Умножение на x выполняется над последними $m=64m'$ столбцами матрицы $G_t(x)$ (соответственно деление на x — над первыми $n=64n'$ столбцами матрицы $C_t(x)$), поэтому эту операцию можно выполнить автоматически, записывая результат умножения строки-коэффициента на матрицу перехода τ_t со сдвигом: последние m' машинных слов результата умножения строки-коэффициента при x^k записываются в массив, соответствующий коэффициенту при x^{k+1} (для $C_t(x)$ — первые n' машинных слов результата умножения строки-коэффициента при x^k записываются в массив, соответствующий коэффициенту при x^{k-1}).

Предложенный способ хранения данных позволяет записывать результаты очередного шага алгоритма на место предыдущих результатов: $G_{t+1}(x)$ вместо $G_t(x)$, а $C_{t+1}(x)$ вместо $C_t(x)$, что снимает необходимость постоянного выделения дополнительных объёмов оперативной памяти и, в свою очередь, снижает нагрузку на системный диспетчер памяти.

3.2. Быстрое умножение полиномиальных матриц на матрицу перехода

Поскольку основной элементарной операцией на каждом шаге алгоритма Копперсмита является умножение большого числа строк длины $m+n$ на одну и ту же матрицу перехода τ_t размера $(m+n)\times(m+n)$, для снижения трудоёмкости всего алгоритма необходимо прежде всего оптимизировать эту операцию.

Один из эффективных методов умножения строки на матрицу небольших размеров связан с использованием так называемых lookup-таблиц. Идея эта используется давно [6, гл. 6]; она достаточно проста и заключается в следующем. Пусть необходимо реализовать умножение строки a длиной $8k$ битов на двоичную матрицу C размеров $8k\times 8k$. Будем считать, что строка a хранится в массиве байтов длины k. Представим массив a в виде суммы k байтовых массивов длины k, в которых только один байт может быть ненулевым: $a=\sum\limits_{i=1}^{k} a_i$, где $a_i=(0,\ldots,0,*,0,\ldots,0)\in\mathbb{Z}_{256}^k$ (ненулевой элемент — на i-й позиции). Тогда произведение строки на матрицу раскладывается в сумму k произведений: $aC=\sum\limits_{i=1}^{k} a_iC$. Если для матрицы C заранее вычислить все возможные произведения вида a_iC — а их число равно $256k$, — то умножение строки длины $8k$ битов на матрицу $8k\times 8k$ сводится к сложению k строк длины $8k$ битов.

В нашем случае роль матрицы C играет матрица перехода τ_t размером $(m+n)\times$ $\times(m+n)$ битов и $k=8(m'+n')$. Для реализации приведённого алгоритма необходимо составить таблицу из $2^{11}(m'+n')$ строк длины $m+n$ битов. Составление этой таблицы является предварительным этапом и выполняется один раз на каждом шаге алгоритма Копперсмита после вычисления матрицы перехода τ_t, поэтому вклад этой процедуры в общую трудоёмкость алгоритма ничтожно мал по сравнению с последующим использованием таблицы при умножении строк $C_t(x)$ и $G_t(x)$ на τ_t. Умножение одной строки на матрицу τ_t в этом случае выполняется за $k=8(m'+n')$ побитовых сложений массивов из $m'+n'$ 64-разрядных слов.

3.3. Многопоточная реализация алгоритма

Рассмотрим вопросы, связанные с параллельным выполнением алгоритма Копперсмита несколькими потоками на многоядерной/многопроцессорной ЭВМ с общей памятью.

Наиболее трудоёмкими операциями алгоритма являются операции умножения полиномиальных матриц $C_t(x)$ и $G_t(x)$ на скалярную матрицу перехода τ_t, причём эти операции выполняются независимо для различных строк матриц $C_t(x)$ и $G_t(x)$, а также для матриц-коэффициентов при различных степенях x. Это предоставляет практически неограниченные возможности для распараллеливания данных операций.

При организации многопоточных вычислений применительно к рассматриваемому случаю необходимо учитывать следующие особенности:

— выбранный способ представления данных (строки матриц $C_t(x)$ и $G_t(x)$ представляются в виде отдельных массивов скалярных строк-коэффициентов при $1, x, x^2, \dots$);
— число строк матриц $C_t(x)$ и $G_t(x)$ равно, как правило, $m = 64m'$ и $n = 64n'$ соответственно, где m' и n' — натуральные числа;
— число ядер в современных ЭВМ с серийными процессорами составляет порядка 12–16 ядер на 1 процессор.

С учётом сказанного, достаточно реализовать параллельное умножение различных строк матриц $C_t(x)$ и $G_t(x)$, так как их общее количество даже при $m' = n' = 1$ в разы превышает возможное число вычислительных ядер. В этом случае многопоточная реализация одного шага алгоритма Копперсмита выглядит следующим образом:

1) управляющий поток вычисляет матрицу $C_t(0)$, а также матрицу перехода τ_t;
2) каждый рабочий поток выполняет умножение выделенных ему строк матрицы $C_t(x)$ ($G_t(x)$) на $\tau_t D/x$ ($\tau_t D$);
3) управляющий поток вычисляет вектор степеней δ_{t+1} и дожидается завершения умножения строк матриц $C_t(x)$ и $D_t(x)$ всеми рабочими потоками.

Важно отметить, что при большой длине L входной матричной последовательности трудоёмкость вычисления матрицы перехода τ_t и вектора степеней δ_{t+1} ничтожно мала по сравнению с умножением матриц $C_t(x)$ и $G_t(x)$ на τ_t, поэтому «простой» рабочих потоков является незначительным.

При использовании большего числа вычислительных ядер — скажем, порядка 1000 — вычисления отдельных строк матриц $C_{t+1}(x)$ и $G_{t+1}(x)$ можно также распределять по нескольким ядрам, разделяя между ними вычисления по диапазону степеней x. Например, при умножении $G_t(x)$ на матрицу перехода τ_t каждый коэффициент каждой строки $G_t(x)$ умножается на τ_t независимо от остальных: если разделить $G_t(x)$ по строкам $G_t(x) = (G_{t,1}(x), \dots, G_{t,n}(x))^T$, а каждую строку $G_{t,i}(x)$ по степеням x: $G_{t,i}(x) = \sum\limits_{j=0}^{t} G_{t,i,j}x^j$, то умножение выполняется отдельно для каждого коэффициента при различных степенях x: $G_{t,i}(x)\tau_t = \sum\limits_{j=0}^{t}(G_{t,i,j}\tau_t)x^j$. В этом случае необходимо лишь учитывать перекрытия на границах диапазонов степеней x, выделенных различным вычислительным ядрам, поскольку одновременно с умножением на τ_t выполняется умножение последних m столбцов $G_t(x)\tau_t$ на x, что достигается за счёт записи результата умножения на τ_t со сдвигом на одну степень x; аналогичным образом выполняется деление первых n столбцов $C_t(x)\tau_t$ на x (см. п. 3.1).

Практически единственным препятствием для эффективного использования большого количества вычислительных ядер при реализации алгоритма Копперсмита яв-

ляется большое число обращений к оперативной памяти. Действительно, с вычислительной точки зрения на каждом шаге выполняются простейшие операции, связанные с умножением строки на матрицу, а при использовании lookup-таблиц это умножение сводится к побитному сложению предварительно вычисленных массивов машинных слов. В то же время выполняется полное чтение и перезапись больших по объёму матриц $G_t(x)$ и $C_t(x)$. Другими словами, основными операциями являются чтение из памяти и запись в память. Поскольку современные модули памяти не могут обрабатывать запросы от нескольких ядер одновременно, распараллеливание алгоритма на большое число ядер становится неэффективным.

Одним из способов решения этой проблемы является использование вычислителей, в которых отдельные модули памяти соответствуют различным процессорам. В этом случае ядра одного процессора обращаются к своей доли оперативной памяти и выполнение алгоритма происходит более эффективно. Частным случаем является использование вычислителей кластерного типа.

3.4. Реализация алгоритма на вычислителях кластерного типа

При организации высокопроизводительных вычислений обычно используются вычислители кластерного типа, или кластеры. Кластер — это разновидность распределённой вычислительной системы, состоящей из нескольких связанных между собой компьютеров (узлов) и используемой как единый ресурс. В качестве связующего элемента выступают Ethernet, InfiniBand или любые другие относительно недорогие сети. Распределение вычислений по отдельным узлам выполняет один или несколько выделенных управляющих узлов.

В отличие от обычных многопоточных реализаций, при реализации алгоритмов на кластерах необходимо учитывать отсутствие общей для всех узлов оперативной памяти. Тем не менее в случае распараллеливания алгоритма Копперсмита всё достаточно просто: каждый узел хранит свою порцию строк матриц $G_t(x)$ и $C_t(x)$, а один шаг алгоритма выглядит следующим образом:

1) управляющий узел получает от каждого рабочего узла часть строк матрицы $C_t(0)$, вычисляет матрицу перехода τ_t и рассылает её копии каждому рабочему узлу;
2) каждый рабочий узел выполняют умножение выделенных ему строк матрицы $C_t(x)$ ($G_t(x)$) на $\tau_t D/x$ ($\tau_t D$);
3) управляющий узел вычисляет вектор степеней δ_{t+1} и дожидается завершения вычислений на всех рабочих узлах.

Объём данных, передаваемых по сети на каждом шаге, минимален и составляет несколько килобайтов: $512m'(m'+n')$ байтов при пересылке матрицы $C_t(0)_{m\times(m+n)}$ и $512(m'+n')^2$ байтов при пересылке матрицы $(\tau_t)_{(m+n)\times(m+n)}$. Трудоёмкость приведения $C_t(0)$ к ступенчатому виду мала по сравнению с остальными вычислениями, поэтому простой узлов кластера при этой операции является несущественным.

На отдельных узлах кластера возможно дальнейшее распараллеливание вычислений по отдельным ядрам, как это описано в предыдущем пункте. Тем не менее из-за особенностей работы оперативной памяти распределение вычислений по большому количеству ядер внутри одного узла не является эффективным.

Необходимо также отметить, что на каждом шаге алгоритма используется весь объём данных, полученных на предыдущем шаге, поэтому выполнение алгоритма на нескольких не связанных между собой вычислителях не представляется возможным.

4. Результаты экспериментов

Приведём результаты экспериментальных исследований времени работы алгоритма Копперсмита. Реализация алгоритма выполнена на языке C++, для компиляции программного кода использован 64-разрядный компилятор среды разработки Microsoft Visual Studio 2008. В качестве вычислителя использована ЭВМ с процессором Intel Xeon (2 процессора 2,53 ГГц по 4 ядра каждый) под управлением 64-разрядной ОС Windows 7. В таблице представлено время работы алгоритма при различных размерах задачи. Рассматривалась последовательность, получаемая в качестве результата работы первого этапа блочного алгоритма Копперсмита при решении СОЛУ из M уравнений; m и n — параметры алгоритма; L — длина последовательности; threads — число потоков.

Время работы алгоритма

M	m	n	L	threads	Время, с
100 000	2	1	2443	1	71
100 000	2	1	2443	2	91
100 000	2	1	2443	4	90
100 000	2	1	2443	8	93
1 000 000	2	1	23537	1	6578
1 000 000	4	2	11818	1	12969
1 000 000	4	2	11818	2	8849
1 000 000	4	2	11818	4	5726
1 000 000	4	2	11818	8	4700

Из таблицы видно, что при небольших размерах задачи распараллеливание не приводит к ускорению работы алгоритма, а наоборот, замедляет его. Это связано с тем, что на каждом шаге алгоритма необходимо выполнить небольшой объём вычислений, и синхронизация потоков на каждом шаге отнимает время, сравнимое с временем вычислений. При увеличении размера входа видно, что параллельная реализация работает быстрее. Однако увеличение числа потоков в 2 раза не приводит к соответствующему снижению времени работы алгоритма из-за особенностей работы оперативной памяти.

5. Субквадратичные алгоритмы

Трудоёмкость алгоритма Копперсмита составляет $\mathrm{O}(m^3L^2)$ элементарных операций, где L — длина матричной последовательности, а m и n — размеры её элементов ($m \geqslant n$). В то же время предложено большое число алгоритмов построения векторных аннулирующих многочленов с асимптотически меньшей трудоёмкостью. Например, если поле P допускает быстрое преобразование Фурье (БПФ), то изложенный в работе [7] алгоритм решает эту задачу за $\mathrm{O}\big(m^3L\log^2 L\big)$ операций. В работе [5] предложена субквадратичная версия алгоритма Копперсмита — рекурсивный алгоритм Копперсмита — Томэ, который также использует БПФ и имеет трудоёмкость $\mathrm{O}\big(m^3L\log^2 L\big)$ операций.

Несмотря на то, что эти алгоритмы имеют в асимптотике меньшую трудоёмкость, при относительно небольших размерах входа алгоритм Копперсмита работает значительно быстрее, поскольку не имеет большой константы в формуле трудоёмкости. Кроме того, у алгоритма Копперсмита на порядок ниже требуемый объём оперативной памяти, что также является немаловажным.

Заключение

Рассмотренный в данной работе алгоритм Копперсмита используется для построения векторных аннулирующих многочленов для матричных последовательностей над

полем. В частности, этот алгоритм является составляющей частью блочного алгоритма Видемана — Копперсмита нахождения решений больших разреженных систем линейных уравнений над полем.

Проведены исследования, связанные с эффективной реализацией алгоритма Копперсмита на современной вычислительной технике. Предложен способ представления входных и промежуточных данных, позволяющий эффективно распараллеливать наиболее трудоёмкие операции каждого шага алгоритма. Рассмотрен подход к оптимизации основной операции — умножения полиномиальных матриц большой степени на скалярную матрицу — за счёт выполнения предварительных вычислений.

Отдельное внимание уделено многопоточной реализации алгоритма, а также выполнению алгоритма на вычислителях кластерного типа. Приведены экспериментальные результаты.

ЛИТЕРАТУРА

1. *Coppersmith D.* Fast evaluation of logarithms in fields of characteristic two // IEEE Trans. Inform. Theory. 1984. V. IT-30(4). P. 587–594.
2. *Montgomery P. L.* A block Lanczos algorithm for finding dependencies over GF(2) // EUROCRYPT'95. LNCS. 1995. V. 921. P. 106–120.
3. *Coppersmith D.* Solving linear equations over GF(2) via block Wiedemann algorithm // Math. Comp. 1994. No. 62(205). P. 333–350.
4. *Wiedemann D. H.* Solving sparse linear equations over finite fields // IEEE Trans. Inform. Theory. 1986. V. IT-32(1). P. 54–62.
5. *Thomé E.* Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm // J. Symbolic Comput. 2002. No. 33. P. 757–775.
6. *Ахо А., Хопкрофт Д., Ульман Дж.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.
7. *Beckerman B. and Labahn G.* A uniform approach for the fast computation of matrix-type Padé approximants // SIAM J. Matrix Anal. Appl. 1994. No. 15(3). P. 804–823.