Параллельное формирование предобуславливателя на основе обращения Шермана-Моррисона*

Н.С. Недожогин, С.П. Копысов, А.К. Новиков

Институт механики УрО РАН

Исследуются возможности ускорения предобусловленных итерационных методов бисопряженных градиентов (BiCGStab) с предобуславливателем на основе разложения матрицы по формуле Шермана-Моррисона. Рассмотрена эффективность распараллеливания наиболее ресурсоёмких операций данного предобуславливателя на многоядерных центральных процессорах и графических процессорах.

1. Введение

Построение эффективных методов решения больших систем линейных алгебраических уравнений на основе предобусловленных итерационных методов, особенно в контексте параллельных вычислений, является достаточно трудной задачей. Матрица предобуславливатель не только должна быть в определенном смысле близка к обратной матрице коэффициентов системы, но и должна допускать эффективно распараллеливаемый алгоритм ее формирования и умножения на вектор.

К широко используемым сегодня можно отнести методы, ориентированные на разреженные матрицы и основанные на неполном разложении на треугольные составляющие, такие как метод неполного LU-разложения и метод неполного разложения Холецкого [1]. Имея высокую эффективность и популярность данные алгоритмы сталкиваются с известными проблемами при их параллельной реализации.

Высокий потенциал распараллеливания имеют предобуславливатели на основе аппроксимации обратной матрицы: полиномиальные (TNS [1] и др.), разреженные аппроксимации обратной матрицы (например, AINV), аппроксимации обратной матрицы в факторизованной форме (такие как FSAI, SPAI и др.) [2], а также метод AISM, основанный на обращении матриц Шермана-Моррисона (Sherman-Morrison formula) [3].

В работе ставятся следующие задачи: разработка методов параллельного построения предобуславливателей AISM; реализация и практическая оценка параллельной эффективности предложенных методов для гибридных вычислительных систем.

2. Предобуславливатель AISM

Рассмотрим предобуславливатель, основанный на обращении матриц методом Шермана-Моррисона. Параллельные свойства алгоритма обращения оказались достаточно хорошими, как и ожидалось из теоретических построений [4]. В настоящей работе исследуется техника предобусловливания задачи, предложенная в [5], схема которой изложена далее.

За основу построения A^{-1} возьмем матрицу B той же размерности, что и A, но с известной обратной матрицей.

Теорема [3]. Пусть B — невырожденная матрица и вектора u и v такие, что $r = 1 + v^T B^{-1} u \neq 0$, матрица $A = B + uv^T$ является обратимой и её обращение находится как

$$A^{-1} = B^{-1} - r^{-1} B^{-1} u v^T B^{-1}. \quad (1)$$

Обозначим u_k, v_k и A_0 — невырожденная матрица, обращение которой просто вычисляется (т.е. A_0 может быть диагональной или даже единичной матрицей). Тогда $A_k = A_0 + \sum_{i=1}^{k} u_i v_i^T$, где $k = 1, n$ и $A = A_n$. Если A_k, u_k,v_k удовлетворяют выражению (1), тогда

*Работа выполнена при поддержке РФФИ (проекты 14-01-00055-а, 14-01-31066-мол_а)

обращение матрицы A может быть вычислено при n-кратном использованием (1)

$$A^{-1} = A_0^{-1} - \sum_{k=1}^{n} r_k^{-1} A_{k-1}^{-1} u_k v_k^T A_{k-1}^{-1}. \tag{2}$$

Представим (2) в матричной форме

$$A_0^{-1} - A^{-1} = \Phi \Omega^{-1} \Psi^T, \tag{3}$$

где $\Phi = [A_0^{-1}u_1, A_1^{-1}u_2, \ldots, A_{n-1}^{-1}u_n]$, $\Psi = [v_1^T A_0^{-1}, u_2^T A_1^{-1}, \ldots, v_n^T A_{n-1}^{-1}]$
и $\Omega^{-1} = diag[r_1^{-1}, r_2^{-1}, \ldots, r_n^{-1}]$. Покажем, что факторизация (3) записывается без явного вычисления A_k^{-1} через вектора u_k, v_k как

$$s_k = u_k - \sum_{i=0}^{k-1} \frac{t_i^T A_0^{-1} u_k}{r_i} s_i, \quad t_k = v_k - \sum_{i=0}^{k-1} \frac{v_k^T A_0^{-1} s_i}{r_i} t_i, \quad k = 1, \ldots, n. \tag{4}$$

Тогда выполняются соотношения

$$A_{k-1}^{-1} u_k = A_0^{-1} s_k, \quad u_k^T A_{k-1}^{-1} = t_k^T A_0^{-1}, \tag{5}$$

$$r_k = 1 + v_k^T A_0^{-1} s_k = 1 + t_k^T A_0^{-1} u_k. \tag{6}$$

С учетом (5) соотношение (3) запишем в виде

$$A_0^{-1} - A^{-1} = A_0^{-1} S \Omega^{-1} T^T A_0^{-1}, \tag{7}$$

где матрицы $S = [s_1, s_2, \ldots, s_n]$, $T = [t_1, t_2, \ldots, t_n]$ столбцы которых вычисляются по u_k,v_k.

Выбор A_0, u_k, v_k следующим образом [5]

$$A_0 = gI_n, \quad u_k = e_k, \quad v_k = (a^k - a_o^k)^T, \quad k = 1, \ldots, n.$$

где I_n, e_k — единичная матрица и ее k- столбец, векторы a^k, a_0^k — k-ая строка матриц A, A_0. Тогда аппроксимация обратной матрицы и соответствующий предобуславливатель примут вид

$$P_1 = A^{-1} = gI_n - g^{-2} U \Omega^{-1} V^T.$$

Рассмотрим вариант когда матрица представима в виде разложения $A = W - Z$, где W — обратимая матрица, $Z = UV^T = \sum\limits_{k=1}^{n} u_k v_k^T$, а v_k, u_k такие, что $d_k = 1 - v_k^T W_{k-1}^{-1} u_k \neq 0$, где $W_k = W_0 - \sum_{i=1}^{k} u_u v_i^T$. Задавая выбор матриц $W = \beta \cdot diag(A)$, $\beta > 0$, $U = I$, $V = Z^T$ и следуя соотношениям (4), (5) и (7), получим выражения для вычисления столбцов матриц S и T

$$s_k = u_k - \sum_{i=1}^{k-1} \frac{t_i^T W^{-1} u_k}{d_i} s_i, \quad t_k = v_k - \sum_{i=1}^{k-1} \frac{v_k^T W^{-1} s_i}{d_i} t_i$$

и обратной матрицы в виде

$$A^{-1} = W^{-1} - W^{-1} S D^{-1} T^T W^{-1}. \tag{8}$$

Используя стратегию фильтрации по значениям элементов (оставляем элементы значения которых больше некоторой величины τ) при вычислении матриц S, T и основываясь на (8) выпишем предобуславливатель, аппроксимирующий обратную матрицу в виде

$$P = W^{-1} - W^{-1} \tilde{S} D^{-1} \tilde{T}^T W^{-1}. \tag{9}$$

Последовательный процесс формирования рассматриваемого предобуславливателя представлен в виде алгоритма 1.

Алгоритм 1 Построение предобуславливателя AISM

1: $A = W - Z$
2: $W = \beta \cdot diag(A); Z = W - A$
3: $U = I; V = Z^T$
4: **for** $k = 1$ **to** n **do**
5: $s_k = u_k$, $t_k = v_k$
6: **for** $i = 1$ **to** $k - 1$ **do**
7: $\delta = (t_i^T W^{-1}, u_k)$
8: **if** $|\frac{\delta}{d_i}| > \tau_u$ **then**
9: $s_k = u_k - \frac{\delta}{d_i} \cdot s_i$
10: **end if**
11: $\delta = (v_k^T W^{-1}, s_i)$
12: **if** $|\frac{\delta}{d_i}| > \tau_v$ **then**
13: $t_k = v_k - \frac{\delta}{d_i} \cdot t_i$
14: **end if**
15: **end for**
16: **for** $j = 1$ **to** n **do**
17: **if** $|(s_k)_j| < \tau_u$ **then**
18: $(s_k)_j = 0$
19: **end if**
20: **if** $|(t_k)_j| < \tau_v$ **then**
21: $(t_k)_j = 0$
22: **end if**
23: **end for**
24: $d_k = 1 - (t_k^T W^{-1}, u_k)$
25: **end for**
26: $\tilde{S} = \{s_1, s_2, \ldots, s_n\}, \tilde{T} = \{t_1, t_2, \ldots, t_n\}$
27: $D = \{d_1, d_2, \ldots, d_n\}$
28: $P = W^{-1} - W^{-1}\tilde{S}D^{-1}\tilde{T}^T W^{-1}$

3. Параллельное построение предобуславливателя

Рассмотрим процесс построения предобуславливателя в модели параллелизма по данным, которая присуща алгоритму 1 и связана с вычислением скалярных и матрично-векторных, матричных произведений.

В алгоритме 2 при формировании предобуславливателя для GPU, для замены скалярных произведений внутри цикла (шаг 7 алгоритма 1) введём матрицы $S_k = \{s_1, ..., s_{k-1}\}$ и $T_k = \{t_1, ..., t_{k-1}\}$. Тогда $k - 1$ скалярных произведений заменяется на матрично-векторные произведения, которые представлены на шагах 6 и 12 алгоритма 2. Здесь x — вектор, результат матрично-векторного произведения, а x_i — компоненты этого вектора. Для вычислений на графических ускорителях оставшегося скалярного произведения векторов на шаге 26 использовалась функция `cublasSdot`. Остальные векторные операции были реализованы с помощью ядер (kernel) CUDA собственной разработки.

Алгоритм 2 Формирование предобуславливателя AISM на GPU

1: $A = W - Z$
2: $W = \beta \cdot diag(A); Z = W - A$
3: $U = I; V = Z^T$
4: **for** $k = 1$ **to** n **do**
5: $s_k = u_k$, $t_k = v_k$
6: $x = u_k T_k^T W^{-1}$
7: **for** $i = 1$ **to** $k - 1$ **do**
8: **if** $|\frac{x_i}{d_i}| > \tau_u$ **then**
9: $s_k = u_k - \frac{x_i}{d_i} \cdot s_i$
10: **end if**
11: **end for**
12: $x = v_k^T S_k W^{-1}$
13: **for** $i = 1$ **to** $k - 1$ **do**
14: **if** $|\frac{x_i}{d_i}| > \tau_v$ **then**
15: $t_k = v_k - \frac{x_i}{d_i} \cdot t_i$
16: **end if**
17: **end for**
18: **for** $j = 1$ **to** n **do**
19: **if** $|(s_k)_j| < \tau_u$ **then**
20: $(s_k)_j = 0$
21: **end if**
22: **if** $|(t_k)_j| < \tau_v$ **then**
23: $(t_k)_j = 0$
24: **end if**
25: **end for**
26: $d_k = 1 - (t_k^T W^{-1}, u_k)$
27: **end for**
28: $\tilde{S} = \{s_1, s_2, \ldots, s_n\}, \tilde{T} = \{t_1, t_2, \ldots, t_n\}$
29: $D = \{d_1, d_2, \ldots, d_n\}$
30: $P = W^{-1} - W^{-1}\tilde{S}D^{-1}\tilde{T}^T W^{-1}$

В рамках одной итерации цикла Алгоритма 2, при вычислении векторов s_k и t_k, не возникает ситуации блокировки памяти, что позволяет выполнять операции матрично-

векторного и скалярных произведений (шаги 6, 12) независимо в параллельных нитях OpenMP. Однако, такой подход требует больших затрат при реализации для multiGPU варианта, так как каждая последующая итерация цикла зависит от данных, полученных на предыдущем шаге, и требуется производить обмен s_k и t_k между памятью различных GPU.

Таблица 1. Ускорение при формировании предобуславливателя P_{AISM}

Матрица	$A(n/nnz)$	$Cond(A)$	P_{AISM}	
			OpenMP	GPU
nasa2910	2910 / 174296	$9.53 \cdot 10^{64}$	1.67	38.3
bcsstk15	3948 / 117816	$6.64 \cdot 10^{9}$	1.81	36.2
Kuu	7102 / 340200	$15.75 \cdot 10^{3}$	1.61	32.1
msc10848	10848 / 1229778	$9.97 \cdot 10^{7}$	1.88	32.1
vibrobox	12328 / 301700	$1.04 \cdot 10^{19}$	1.7	22.9
cdde5	961/ 4681	$1.64 \cdot 10^{4}$	1.64	19.9
ex37	3565 / 67591	$1.79 \cdot 10^{2}$	1.7	30.1
rajat03	7602 / 32653	$1.26 \cdot 10^{7}$	1.67	23.8
flowmeter5	9669 / 67391	$7.1 \cdot 10^{6}$	1.7	23.6
ex19	12005 / 259577	$2.15 \cdot 10^{12}$	1.7	21.9
sme3Da	12504 /874887	$5.22 \cdot 10^{7}$	3.5	40.4
poisson3Da	13514 / 352762	$1.12 \cdot 10^{3}$	1.65	20.9

Кроме этого, для сравнения все операции над векторами (инициализация, умножение вектора на скаляр, сложение векторов) были реализованы также в модели общей памяти OpenMP с помощью `#pragma omp for`.

Последний шаг Алгоритма 2, содержащий матричные операции (умножение, сложение) был так же распараллелен как в рамках технологии CUDA, так и OpenMP. При использовании CUDA, было реализовано ядро, представляющее матрично-матричное произведение в виде последовательных матрично-векторных произведений.

При построении предобуславливателя P_{AISM} разреженность матрицы неизвестна. Промежуточные вычисления на этапе формирования выполнялись над векторами s_k,t_k, являющиеся столбцами матриц S,T, хранящихся по строкам. Расходы по памяти увеличивались, но сокращалось время обращения к элементам векторов. Для преобразования из сжатого формата хранения матриц CSR в формат хранения полных строк (этап построения предобуславливателя) и обратное преобразование (матрично-векторное произведение при решении СЛАУ) были разработаны эффективные параллельные алгоритмы, позволяющие пренебречь затратами на преобразование матриц. Затраты по памяти при хранении матриц S, T и P составляют $18 * n^2$ байт для варианта с двойной точностью и $12 * n^2$ — с одинарной, что накладывает ограничение на максимальный размер рассматриваемых систем для решения на GPU ($n \sim 13000$). Хотя возможны варианты сокращения затрат памяти при хранении матриц S и T в одном из разреженных форматов.

Сравнение эффективности распараллеливания предобуславливателя P_{AISM} и характеристики матриц представлены в таблице 1.

4. Результаты численных экспериментов

Для тестирования были использованы предобусловленные методы сопряженных градиентов и бисопряженных стабилизированных градиентов. Численные эксперименты прово-

дились на GPU-ускорителе GeForce GTX 780 с 3 ГБ графической памяти и на восьми ядрах CPU (два четырехядерных процессора Intel Xeon E5-2609, 2.4 ГГц) и 64 ГБ оперативной памяти.

Часть тестовых задач выполнялась в пакете OpenFoam, в который была успешно выполнена интеграция программной реализации на GPU решения СЛАУ итерационным методом BiCGStab с параллельным предобуславливателем AISM. Кроме того, в численных экспериментах были использованы матрицы из коллекции The University of Florida Sparse Matrix Collection. Решались системы уравнений $Ay = f$ с известным точным решением $y = [1, 1, \ldots, 1]$, матрицы которых хранились в сжатом строчном формате (CSR). В качестве начального приближения выбиралось $y_0 = [0, 0, \ldots, 0]$, а критерий сходимости — $||r_i|| \leqslant 10^{-6}||r_0||$, где $r_i = f - Ay_i$. В численных экспериментах точность фильтрации для матриц S и T выбиралась $\tau_u = \tau_v = 0.01$, $\beta = 100$.

Таблица 2. Вычислительные затраты предобуславливателей при решении систем с несимметричными матрицами, $t_p/t_{its}(its)$

Матрица A	$P_{ILU(0)}$ CPU/GPU	$P_{ILU(1)}$ CPU/GPU	P_{AISM} GPU/GPU	
			$\tau = 0.01$	$\tau = 0.0001$
cdde5	0.0002/0.13 (140)	0.002/0.12 (106)	0.56/0.13 (179)	0.57/0.13(167)
ex37	0.003/0.03 (3)	1.4/0.03 (2)	15.66/0.004 (5)	15.44/0.004 (4)
rajat03	—	—	169/0.07 (84)	169/0.11 (135)
flowmeter5	0.002/0.36 (63)	0.04/0.34 (32)	357.6/0.11 (120)	357.9/0.15 (112)
ex19	0.04/—	699/0.5 (279)	700/0.35 (132)	703/0.39 (128)
sme3Da	0.15/13.23(1700)	495/16.61(158)	814/11.83(2032)	843/13.5(1338)
poisson3Da	0.04/0.13 (24)	56.3/0.58 (11)	1049/0.05 (28)	1066/0.24 (30)

Сравним сначала результаты полученные при решении несимметричных систем. Для решения использовался итерационный алгоритм BiCGStab, в котором в качестве предобуславливателя берется одно из рассматриваемых разложений класса ILU. Предобуславливатель $P_{ILU(p)}$ [1] получается на основе ILU алгоритма расширением шаблона ненулевых элементов для матриц L и U. При добавление элементов в качестве кандидатов рассматриваются лишь те элементы, относительная норма которых превышает τ, и при этом расширятся каждая строка шаблона ненулевых элементов не более чем на p элементов.

Для проведения численных экспериментов матрицы, свойства которых приведены в таблице 1, хранились в сжатом формате CSR. Предобуславливатель $ILU(p)$ формировался на центральном процессоре, а итерационный процесс решения BiCGStab на ускорителе вычислений. В таблице 2 приведено соотношение времени формирования предобуславливателя и времени итерационного процесса и числа итераций, необходимых для решения систем линейных уравнений с использованием $ILU(p)$ разложений и рассматриваемого алгоритма AISM. Использование $ILU(p)$ в случае матрицы "ex19"не привело к решению системы за приемлемое время. По времени работы алгоритмов решения систем BiCGStab видны преимущества предобуславливателя P_{AISM}. Затраты на одну итерацию в этом случае существенно ниже, чем при $ILU(p)$. Достигнутое ускорение при формировании явного предобуславливателя P_{AISM} все-таки сохраняет достаточно большие вычислительные затраты и требует дальнейших исследований.

Исключительно для целей тестирования в таблице 3 приведены результаты для явных предобуславливателей $AINV$, $FSAI$, TNS при решении симметричных систем уравнений методом сопряженных градиентов с формированием матрицы предобуславливателя на центральных процессорах и графических ускорителях.

По скорости сходимости результаты многих тестов для различных предобуславливателей сопоставимы, а для матриц "bcsstk15", "vibrobox"применение P_{AISM} позволило получить меньшее число итераций. Затраты на решения систем уравнений с предобуславливателями P_{AISM} и P_{AINF} примерно одинаковы.

Потенциально сокращение затрат на формирования P_{AISM} для случая симметричных матриц представляется возможным и перспективным. Так для достаточно большой и заполненной матрицы "mcs10848"затраты на построение предобуславливателя P_{FSAI} практически втрое превышают время формирования предлагаемого алгоритма.

Таблица 3. Вычислительные затраты явных предобуславливателей при решении систем с симметричными матрицами, $t_p/t_{its}(its)$

Матрица	P_{AINV}	P_{FSAI}	P_{TNS}	$P_{AISM}(\tau = 0.0001)$
A	CPU/GPU	CPU/GPU	GPU/GPU	GPU/GPU
nasa2910	2.5/0.65 (314)	19.5/0.04 (129)	0.002/0.32 (962)	8.78/0.62 (387)
bcsstk15	0.15/0.6 (293)	1.47/0.03 (109)	0.002/0.08 (259)	20.37/0.17 (81)
Kuu	4.03/0.16 (75)	11.7/0.02 (45)	0.004/0.1 (241)	142.03/0.18 (103)
msc10848	1.48/2.68 (1190)	1168/0.02 (35)	0.006/14.7 (21871)	505.5/5.12 (846)
vibrobox	1.29/1.42 (683)	85/0.05 (92)	0.003/1.98 (4682)	814/0.81 (52)

Алгоритм $AISM$ обладает определенными свойствами при распараллеливании, например, при замене скалярных произведений на матрично-векторные операции. Время вычисления предобуславливателя остается значительным и превышает затраты итерационных приближений при решении СЛАУ, в отличии от предобуславливателя $AINV$, где затраты во многих случаях соизмеримы. Дальнейшее повышение параллельной эффективности предобуславливателя $AISM$ должно быть связано с тем или иным блочным представлением алгоритма и выделением крупноблочной декомпозиции матрицы.

Литература

1. Saad Y Iterative Methods for Sparse Linear Systems. SIAM, 2003.
2. Benzi M. Preconditioning Techniques for Large Linear Systems: A Survey // Journal of Computational Physics. 2002. V. 182, № 2. P. 418–477.
3. Sherman J., Morrison W.J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix // Ann. Math. Statistics. 1950. V. 21, № 1. P. 124–127.
4. Недожогин Н. С., Сармакеева А. С., Копысов С. П. Высокопроизводительный алгоритм Шермана-Моррисона обращения матриц на GPU // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2014. Т. 3, № 2. С. 101–108.
5. Bru M., Cerdán J., Marín J., Mas J. Preconditioning sparse nonsymmetric linear systems with Sherman-Morrison formula // SIAM J. Sci. Comput. 2003. V. 28, № 2. P. 701 –715.
6. Копысов С. П., Кузьмин И. М., Недожогин Н. С., Новиков А.К. Параллельные алгоритмы формирования и решения системы дополнения Шура на графических ускорителях // Ученые записки Казанского университета. Серия «Физико-математические науки.» 2012. Т. 154, № 3. С. 202–215.