УДК 519.615.5

О. В. Белякова, А. М. Наумов, И. А. Ветров

РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ МЕТОДОВ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Для решения систем линейных алгебраических уравнений с плотной матрицей представлен параллельный алгоритм метода последовательной верхней релаксации. Показано применение OpenMP-технологии для реализации этого алгоритма. Приведены результаты численных экспериментов.

For solving systems of linear equations with dense system matrix there is presenting a parallel algorithm of SOR-method. There is showing OpenMP-programm for the parallel algorithm. There are giving the results of numerical experiments.

Ключевые слова: параллельные алгоритмы, системы линейных алгебраических уравнений, метод последовательной верхней релаксации, OpenMP-программирование.

Keywords: parallel algorithms, systems of linear equation, SOR-method, OpenMP-programming.

Введение

Актуальность параллельных алгоритмов решения систем линейных алгебраических уравнений (СЛАУ) объясняется потребностью быстро решать СЛАУ больших порядков во многих современных задачах, например задачах моделирования физических процессов в 3- и 4-мерных пространствах. Часто наиболее подходящими методами решения СЛАУ являются итерационные методы, которые позволяют достаточно быстро находить приближенное решение требуемой точности.

Для организации параллельных вычислений в соответствии с выбранными итерационными методами со стороны программиста решается дополнительная задача — распределения вычислений по процессорным элементам многопроцессорной вычислительной системы и синхронизации работы соответствующих процессов. Эта задача рассматривается многими авторами, например [1—3]. Часто в литературе параллельные алгоритмы описывают на естественном языке, но для их эффективной реализации этого может оказаться недостаточно.

В данной работе на примере известного метода последовательной верхней релаксации решения СЛАУ (см. [4]) подробно описана методика разработки параллельных алгоритмов и сам параллельный алго-

ритм для выполнения в мультипроцессоре, приведен листинг реализации с помощью OpenMP-технологии основных операций в алгоритме, дан пример анализа полученных численных результатов.

Последовательный алгоритм

Метод последовательной верхней релаксации (ПВР) применим для решения СЛАУ, в которой матрица системы является симметричной $A = A^T$, положительно определенной A > 0,

$$Ax = f,\ A = A^T,\ A > 0, \tag{1}$$

где f — некоторый вектор; x — решение системы, искомый вектор. Будем полагать далее, что $A \in R^{n \times n}$, $f, x \subset R^n$.

Метод последовательной верхней релаксации является итерационным одношаговым методом, в котором при помощи предыдущего приближенного решения $y^{(k)}$ находят новое приближенное решение $y^{(k+1)}$ по формуле (см. [4])

$$y_i^{(k+1)} = (1-w)\cdot y_i^{(k)} + \frac{w}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}\cdot y_j^{(k+1)} - \sum_{j=i+1}^{n} a_{i,j}\cdot y_j^{(k)} \right),\ i = 1,...,n,\ \ w \in (1;2), \tag{2}$$

где k — номер итерации; w — числовой параметр.

Известно, что метод (2) сходится к точному решению системы (1) при любом начальном приближении $y^{(0)}$. Основное преимущество этого метода в том, что часто возможно определить параметр w так, чтобы метод ПВР сходился существенно быстрее некоторых других итерационных методов, например методов Якоби и Зейделя. Однако выбор этого параметра — весьма трудная задача, которая во многих случаях решается экспериментальным путем [4]. Итерационный процесс можно остановить при выполнении условия

$$\left\| y^{(k+1)} - y^{(k)} \right\| \le \varepsilon \cdot \left\| y^{(k)} - y^{(k-1)} \right\|, \tag{3}$$

где ε — заданная точность.

Если на текущем шаге условие (3) не выполняется, то полагают k = k + 1, повторяют вычисления (2) и процесс продолжается. Приближенным решением полагают вектор $y^{(k+1)}$.

Из формулы (2) следует, что сложность одной итерации метода ПВР составляет $(2n^2 + 5n)$. Обозначим через K номер последней итерации. Тогда сложность алгоритма метода получения приближенного решения с заданной точностью можно записать в виде

$$C(n) \approx (2n^2 + 5n)\cdot K. \tag{4}$$

При реализации метода ПВР нет необходимости хранить вектор с предыдущим приближенным решением $y^{(k)}$. Достаточно использовать один вектор, в котором будут пересчитываться значения компонент текущего приближенного решения $y^{(k+1)}$.

В случае плотной матрицы системы метод ПВР содержит большое количество последовательных операций. Из расчетной формулы (2) можно заметить, что основной потенциал параллелизма содержится в параллельном вычислении скалярного произведения двух векторов большой размерности: строки матрицы A и приближенного решения $y^{(k)}$ при вычислении *i*-й компоненты нового приближения $y^{(k+1)}$.

Принципы разработки параллельного алгоритма

Разработку параллельных алгоритмов рекомендуется выполнять в следующем порядке [1]:

1) провести анализ последовательного алгоритма и разделить его на подзадачи, которые могут быть вычислены существенно независимо друг от друга;

2) определить необходимые взаимодействия процессов, исполняющих образованный набор подзадач для решения общей задачи, и назначить способы синхронизации процессов;

3) задать количество рабочих процессов и распределить образованный набор подзадач между процессами так, чтобы обеспечить балансировку вычислительной нагрузки (примерно одинаковый объем вычислений во всех процессах) и минимум коммуникационных взаимодействий между процессами;

4) выполнить предварительную оценку эффективности разработанного параллельного алгоритма, вычислив ускорение S_p и эффективность использования процессоров E_p (основные характеристики параллельного алгоритма), например по формулам [5]

$$S_p = \frac{T_1}{T_p},\ E_p = \frac{S_p}{p}, \tag{5}$$

где p — количество рабочих процессов; T_1 — время выполнения последовательного алгоритма; T_p — время вычислений самого нагруженного процесса.

Время выполнения последовательного алгоритма оценивается по формуле $T_1 = C(n)\cdot\tau$, τ — среднее время выполнения одной базовой операции алгоритма. В качестве базовых операций рассматривают наиболее массовые операции в алгоритме, например арифметические операции.

Параллельный алгоритм ПВР

В литературе редко встречается параллельный алгоритм для реализации метода ПВР решения системы с плотной матрицей. Представленный далее параллельный алгоритм основан на параллельном вычислении скалярного произведения векторов. Такой подход рекомендуется в [6].

Придерживаясь принципов, указанных в предыдущем разделе, составим параллельный алгоритм метода последовательной верхней релаксации. Будем предполагать, что количество процессоров $p << n$ и $n \% p = 0$, каждый процесс исполняется в отдельном процессорном элементе (ядре), $y^{(0)} = 0$. Обозначим через $A[i]$ строку с номером i матрицы A, $[A[i]]_m$ — m-й блок-вектор этой строки размера n/m, $[y^{(k)}]_m$ — блок вектора $y^{(k)}$ размера n/m.

I. На каждой итерации с номером k ($k = 0, 1, \ldots$) выполняется

Алгоритм ПВР_Par:

1. Пусть $i = 1$.

2. Каждый процесс P_m, $m = 0, \ldots, p - 1$ вычисляет скаляр

$$temp_m = [A[i]]_m \cdot [y^{(k)}]_m .$$

3. Процесс-мастер (процесс с номером $m = 0$) суммирует вычисленные всеми процессами скаляры и корректирует сумму в соответствии с формулой (2):

$$c = \sum_{m=0}^{p-1} temp_m - a_{i,i} \cdot y_i^{(k)} .$$

4. Процесс-мастер вычисляет i-ю компоненту текущего приближения $y^{(k+1)}$ по формуле

$$y_i^{(k+1)} = (1 - w) \cdot y_i^{(k)} + \frac{w}{a_{i,i}} (b_i - c).$$

5. Процесс-мастер увеличивает значение i : $i = i + 1$ и переходит к пункту 2 при $i <= n$.

6. Процесс-мастер проверяет критерий продолжения / остановки итерационного процесса в соответствии с формулой (3) и либо увеличивает номер итерации $k = k + 1$ и переходит к пункту 1, либо выводит результат $y^{(k+1)}$.

II. После операции в пункте 2 необходима барьерная синхронизация, чтобы процесс-мастер смог вычислить вспомогательный скаляр c. Других синхронизаций нет, поскольку остальные пункты алгоритма выполняются последовательно одним процессом — процессом-мастером.

III. Вычислительная нагрузка в целом сбалансирована, поскольку все рабочие процессы перемножают блоки-векторы одинакового размера n/p, а в силу простоты основной расчетной формулы (2) процессу-мастеру дополнительно приходится выполнить сравнительно небольшое количество арифметических операций.

IV. Получим теоретические оценки для ускорения и эффективности использования процессоров представленного параллельного алгоритма, используя грубые схемы оценивания (4), (5) без учета времени на накладные расходы. Поскольку

$$T_1 = (2n^2 + 5n) \cdot K \cdot \tau, \quad T_p = (2n^2 / p + (p + 7)n) \cdot K \cdot \tau,$$

то

$$S_p = \frac{2n^2 + 5n}{2n^2 / p + (p+7)n}.$$

По этим формулам легко получить сравнительную таблицу 1.

Таблица 1

Зависимость теоретических значений S_p и E_p параллельного алгоритма ПВР от порядка матрицы и количества процессоров

Порядок матрицы, n	p = 2		p = 4		p = 10	
	S_p	E_p	S_p	E_p	S_p	E_p
3000	1,996	0,998	3,974	0,994	9,732	0,973
5000	1,997	0,999	3,984	0.996	9,838	0,984
7000	1,998	0,999	3,989	0,997	9,884	0,988

Из таблицы 1 видно, что ускорение увеличивается с ростом порядка матрицы и увеличением количества процессов. Однако с увеличением количества процессов ускорение оказывается все дальше от максимально возможного значения *p*, а эффективность использования процессоров уменьшается. В целом характеристики параллельного алгоритма в грубом приближении хорошие, но на практике следует ожидать меньших значений характеристик алгоритма и усиления отрицательных трендов изменения характеристик из-за большого количества не учтенных в формулах (4), (5) синхронизаций и обращения к оперативной памяти компьютера.

Реализация параллельного алгоритма ПВР

Для реализации параллельного алгоритма в мультипроцессоре наиболее простой является OpenMP-технология. Одними из основных ее инструментов являются директивы компилятору *parallel* и *for* с параметром *schedule* для разбивки на блоки множества значений переменной цикла и параметром *reduction* для накопления в разделяемой переменной суммы значений, получаемых разными процессами. Использование параметра *reduction* позволяет вычислить скаляр *c* из алгоритма ПВР_Par внутри блока параллельных вычислений. Поэтому явной барьерной синхронизации в программе не потребуется. Также будем использовать функцию *omp_set_num_threads(p),* устанавливающую количество рабочих процессов в блоках параллельных вычислений. Приведем листинг с реализацией алгоритма ПВР_Par с помощью OpenMP-технологии:

```
int p=2;
omp_set_num_threads(p);
double temp, c, dif=0;
for (int i = 0; i < n; i++)
```

```
{       //параллельное умножение строки матрицы на вектор приближения
        temp = 0;
    #pragma omp for schedule(static, n / p) reduction(+: temp)
        for (int j = 0; j < n; j++)
        temp += A[i][j] * y[j];
            //параллельные вычисления закончились, далее до конца внешнего цикла //операции выполняет процесс-мастер; находим i-ый элемент нового //приближения
        {c=temp-A[i][i] * y[i];
        y_prev[i]=y[i]; y[i] = (1 - w) * y[i] + w * (b[i] - c) / A[i][i];
        // в dif накапливается значение максимальной нормы ||y^(k+1) - y^(k)||
        if (abs(y_prev[i]-y[i])>dif) dif= abs(y_prev[i]-y[i]);
}
```

Вычислительные эксперименты проводились при $\varepsilon = 10^{-4}, w = 1{,}7$. Результаты расчетов собраны в таблице 2.

Таблица 2

Экспериментальные ускорение и эффективность использования процессоров в зависимости от порядка матрицы и количества процессов

Порядок матрицы, n	p = 2		p = 4	
	S_p	E_p	S_p	E_p
3000	1,151	0,576	1,297	0,324
5000	1,431	0,716	1,493	0.373
7000	1,300	0,650	1,775	0,444

Из таблицы 2 видно, что рассмотренный параллельный алгоритм оказывается более эффективным по сравнению с последовательным алгоритмом лишь при достаточно большом порядке СЛАУ. Закономерности изменения характеристик параллельного алгоритма соответствуют указанным для таблицы 1, но абсолютные значения ускорения и эффективности использования процессоров довольно пессимистические. Возможно, такие результаты связаны с реализацией параллельного алгоритма и неполным использованием инструментов параллельных вычислений, содержащихся в OpenMP-технологии.

Список литературы

1. *Гергель В.П.* Высокопроизводительные вычисления для многоядерных многопроцессорных систем. М., 2010.

2. *Эндрюс Г.Р.* Основы многопоточного, параллельного и распределенного программирования. М., 2003.

3. *Старченко А.В.* Высокопроизводительные вычисления на кластерах. Томск, 2008.

4. *Голуб Дж., Ван Лоун Ч.* Матричные вычисления. М., 1999.

5. *Гергель В.П.* Теория и практика параллельных вычислений. М., 2007.

6. *Гергель В.П., Воеводин В.В., Сысоев А.В., Баркалов К.А.* Intel Parallel Programming Professional. М., 2016.

Об авторах

Ольга Владиславовна Белякова — канд. физ.-мат. наук, доц., Балтийский федеральный университет им. И. Канта, Россия.
E-mail: OBelyakova@kantiana.ru

Александр Максимович Наумов — студент, Балтийский федеральный университет им. И. Канта, Россия.
E-mail: alex_newmov@mail.ru

Игорь Анатольевич Ветров — канд. техн. наук, доц., Балтийский федеральный университет им. И. Канта, Россия.
E-mail: vetrov.gosha209@yandex.ru

The authors

Dr Olga V. Belyakova, Associate Professor, I. Kant Baltic Federal University, Russia.
E-mail: OBelyakova@kantiana.ru

Alexander M. Naumov, Undergraduate Student, I. Kant Baltic Federal University, Russia.
E-mail: alex_newmov@mail.ru

Dr Igor A.Vetrov, Associate Professor, I. Kant Baltic Federal University, Russia.
E-mail: vetrov.gosha209@yandex.ru