

# ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ РЕШЕНИЯ СЛАУ МЕТОДОМ ГАУССА-ЗЕЙДЕЛЯ

Г.И. Шпаковский, А.Е. Верхотуров  
Беларусь, г. Минск

В докладе рассматривается параллельный алгоритм решения системы линейных алгебраических уравнений (СЛАУ) методом Гаусса-Зейделя [1]. Этот метод характеризуется строго последовательным характером определения корней системы уравнений, поэтому создание эффективной организации параллельных вычислений для него позволяет перенести эту организацию на многие другие численные методы. Параллельный алгоритм основан на принципе упреждающих вычислений, позволяющих минимизировать влияние последовательного вычисления корней.

Параллельному решению СЛАУ методом Гаусса-Зейделя посвящены работы [2-4]. Настоящая работа отличается тем, что в ней оценка эффективности производится для реальных характеристик современных кластеров. Оценка распараллеливания производится для одной итерации, поэтому вопросы приведения матриц к виду, удобному для выполнения итераций, вопросы сходимости, выбора начального приближения, оценки достигнутой точности – не рассматриваются.

Обычно для выполнения итерационных вычислений исходную систему линейных алгебраических уравнений приводят к виду:

$$\begin{aligned} X_1 &= c_{11}x_1 + c_{12}x_2 + \dots + c_{1m}x_m + d_1 \\ X_2 &= c_{21}x_1 + c_{22}x_2 + \dots + c_{2m}x_m + d_2 \\ \dots & \\ X_m &= c_{m1}x_1 + c_{m2}x_2 + \dots + c_{mm}x_m + d_m \end{aligned}$$

Метод Гаусса-Зейделя состоит в том, что очередное приближение корня вычисляется с учетом ранее полученных значений корней:

$$x_i^{k+1} = \sum_{j=1}^{i-1} c_{ij} \cdot x_j^{k+1} + \sum_{j=i}^m c_{ij} \cdot X_j^k + d_i$$

где  $m$  – размерность матрицы коэффициентов  $c_{ij}$ ,  $k$  – номер итерации.

Рассмотрим сначала ситуацию, когда  $n=m$ , где  $n$  – число процессоров. В правой части матрицы (буква П) хранятся коэффициенты, используемые для вычислений со «старыми» значениями корней, в левой (буква

Л) – коэффициенты для вычисления новых корней. Рассмотрим начальный момент, когда в строке  $i-1$  вычисляется корень  $x_{i-1}^{k+1}$  (рис. 1,а). Для этого предположим, что каким-то образом ранее вычислены суммы, охватывающие отмеченный стрелками участок коэффициентов:  $S_{s,i-1} = \sum_{j=1}^{i-2} c_{i-1,j} \times x_j^{k+1}$  и  $S_{n,i-1} = \sum_{j=i}^m c_{i-1,j} \times x_j^k + d_{i-1}$ . Тогда  $x_{i-1}^{k+1} = S_{s,i-1} + S_{n,i-1}$ . Полученное значение этого корня через коммуникационную систему рассыпается всем процессорам выше и ниже процессора строки  $i-1$ .

Теперь рассмотрим последовательность операций в процессоре строки  $i$ . В этой строке уже сформировано значение суммы  $s_{s,i} = \sum_{j=1}^{i-2} c_{i,j} \times x_j^{k+1}$  (здесь  $s$  – маленькое). Такие же суммы уже вычислены и во всех низлежащих процессорах для своих коэффициентов. Далее в строке  $i$  необходимо выполнить следующие две операции:  $S_{s,i} = S_{s,i-1} + c_{i,i-1} \times x_{i-1}^{k+1}$  и  $x_i^{k+1} = S_{s,i} + S_{n,i}$ . Эти операции на уровне строки  $i$  требуют времени  $t = t_s + t_+ + t_+$ . Все операции на уровнях  $i+1, i+2$  и так далее до уровня  $m$  будут повторяться, поэтому полное время  $T^m$  параллельного решения СЛАУ методом Гаусса-Зейделя без учета обменов будет  $T^m = m \times t$ .

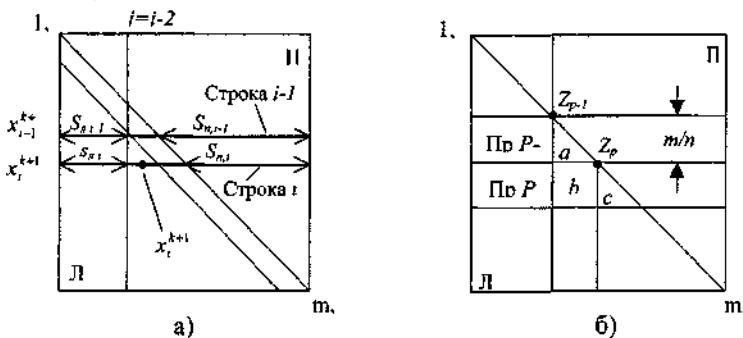


Рис.1. Организация параллельных вычислений СЛАУ методом Гаусса-Зейделя,  
а)  $n=m$ ; б)  $n < m$ .

Как уже говорилось, значение  $x_{i-1}^{k+1}$  посыпается также и в процессоры, лежащие выше строки  $i-1$ . Эти процессоры уже освободились в данной итерации от вычисления корней  $x_l^{k+1}$ , где  $l < i-1$ , поэтому они используются для вычисления в каждой  $l$ -ой строке частных сумм  $S_{s,l}^{k+1} = \sum_{j=1}^{i-2} a_{l,j} \times x_j^{k+1}$ . Поэтому одновременно с вычислением последнего кор-

ия  $x_m^{k+1}$  в правой части уже будут получены все суммы  $S_n$ , для следующей итерации.

Приведенная выше ситуация для  $n=m$  использовалась для пояснения принципа распараллеливания. Реально  $n < m$  или даже  $n \ll m$ . Рассмотрим процесс параллельных вычислений для двух смежных процессоров  $p-1$  и  $p$  (рис. 1,б). Каждый из процессоров вычисляет  $m/n$  корней  $x_i^{k+1}$ .

Для каждой строки в процессоре  $p-1$  как и прежде (рис. 1,а) на основе вычисленных ранее корней  $x_i^{k+1}$  сформированы суммы  $S_n$  и  $S_{n-1}$ . Таким образом, для получения  $m/n$  корней в зоне процессора  $p-1$ , ему необходимо произвести вычисления в треугольнике  $a$ . Эти корни через коммуникационную систему будут переданы в процессор  $p$ , который вычислит очередные  $m/n$  корней. Назовем шагом вычислений процесс вычислений на участке одного процессора. Следовательно, сумма затрат времени на всех  $n$  шагах и составит время решения СЛАУ  $T_{\text{реш}} = m \times t_m$ .

Пусть время одного шага  $t_m$  состоит из времени перехода от точки  $z_{p-1}$  до точки  $z_p$ . Чтобы процессор  $p$  мог начать вычисления в точке  $z_p$ , должны выполниться следующие действия:

- Ожидание вычисления первой части корней треугольника  $a$ :

$$t_{\text{ож}} = \left( \frac{m}{n \cdot k} \right)^2 \cdot \frac{1}{2} \cdot t$$

Здесь:  $t = t_s + t_+$ ,  $k = 1, \dots, m/n$  – доля корней, передаваемых от процессора  $p-1$  в процессор  $p$  в первой итерации. При  $k=1$  передача производится только после вычисления всех корней процессором  $p-1$ ; в случае  $k=m/n$  передача осуществляется после вычисления каждого очередного корня.

- Передача первой части корней из процессора  $p-1$  в процессор  $p$  через коммуникационную систему:

$$t_{\text{неп}} = t_s + \frac{m}{n \cdot k} \cdot t_{\text{ср}}$$

Здесь  $t_s$  – латентность (начальная задержка) коммуникатора,  $t_{\text{ср}}$  – время передачи одного слова коммуникационной системой.

- Вычисление прямоугольника  $b$ :

$$t_{\text{ср}} = \left( \frac{m}{n} \right)^2 \cdot t$$

Так как треугольник  $b$  равен треугольнику  $a$ , то для процессора  $p+1$  и всех последующих вычисления выполняются аналогично.

Ускорение вычисляется по формуле  $R = T_{\text{пар}} / T_{\text{пос}}$ , где  $T_{\text{пар}} = m^2 \cdot t$  - время последовательного, а  $T_{\text{пар}} = n \cdot (t_{\text{осн}} + t_{\text{пар}} + t_{\text{сн}})$  - время параллельного вычисления СЛАУ методом Гаусса-Зейделя, поэтому получаем:

$$R = \frac{m^2 \cdot t}{n \cdot \left[ \left( \frac{m}{n \cdot k} \right)^2 \cdot \frac{1}{2} \cdot t + \left( t_s + \frac{m}{n \cdot k} \cdot t_{\text{сн}} \right) + \left( \frac{m}{n} \right)^2 \cdot t \right]}$$

Это выражение в стандартной форме приобретает вид:

$$R = \frac{1}{\frac{1}{n} \cdot \left( 1 + \frac{1}{2 \cdot k^2} \right) \cdot t + n \cdot \frac{1}{m^2} \cdot \frac{t_s}{t} + \frac{1}{m \cdot k} \cdot \frac{t_{\text{сн}}}{t}}$$

Первый член знаменателя этого выражения соответствует процессорным вычислениям, два следующих члена определяют вклад коммуникационной системы – ее задержку и пропускную способность.

Приведенное выражение использовалось для оценки ускорения:

- Для небольшого кластера типа Beowulf, построенного на базе низкоскоростной сети Fast Ethernet с числом процессоров до 20. Расчет производился для:  $m=10^3$ ,  $t=50$  нс (учитываются промахи кэша, вспомогательные операции и др.),  $t_s=150$  мкс,  $t_{\text{сн}}=1$  мкс.
- Для большого кластера типа СКИФ с быстродействующей сетью SCI и характеристиками:  $m=10^4$ ,  $t=50$  нс,  $t_s=2$  мкс,  $t_{\text{сн}}=20$  нс.

В обоих случаях для пересылки вычисленных корней использовалась известная операция коллективного обмена *bcast*, обеспечивающая одновременную передачу данных от одного процессора всем другим.

Соответствующие графики представлены на рис. 2 и рис.3.

### Выходы

1. Предложенный алгоритм обеспечивает предельно возможный уровень распараллеливания для метода Гаусса-Зейделя
2. Для кластера на базе сети Fast Ethernet низкие результаты объясняются в первую очередь очень большой латентностью коммуникационной системы – 150 мкс. Результаты будут существенно лучше для матриц большего размера.
3. Для кластера на SCI и больших матриц достигается почти линейный рост ускорения.



Рис.2. Ускорение для кластера типа Beowulf с сетью Fast Ethernet.



Рис.3. Ускорение для большого кластера с сетью SCI

### Литература

1. Орtega Дж. Введение в параллельные и векторные методы решения линейных систем Пер. С англ – М.: Мир, 1991. –367 с.
2. В.Д. Корнеев. Параллельное программирование в MPI. Новосибирск: Изд-во СО РАН, 2000. 213 с.
3. Г.И. Шпаконский, Н.В. Серикова. Программирование для многопроцессорных систем в стандарте MPI. Мн : БГУ, 2002. 323 с.
4. Д.Л. Головашкин, О.Е. Горбунов. Параллельное решение СЛАУ методом Зейделя. Вестн. Самар. Гос. Техн. Ун-та. Сер. Физико-математические науки. 2004. №27