Использование последовательно-параллельного метода для распараллеливания алгоритмов с ассоциативными операциями*

А.В. Фролов

ИВМ РАН

Автором исследуются возможности использования последовательно-параллельного метода конструирования новых параллельных версий известных численных методов, а также вопросы сбалансированности получаемых методов - как по части распределения вычислений, так и по их устойчивости.

1. Введение

Последовательно-параллельный алгоритм известен давно и используется, главным образом, как эрзац блочного метода[1], там, где есть возможность использовать свойство ассоциативности операций. К последним, например, можно отнести довольно широкий класс алгоритмов, содержащих последовательные рекуррентные вычисления с линейными и дробно-линейными формулами. В данной статье автор исследует и предлагает использовать свойства последовательно-параллельного метода для раскрытия большего потенциала параллелизма, чем обычно принято видеть в ряде алгоритмов.

2. Последовательно-параллельный метод для алгоритмов с одним результатом

Последовательно-параллельный алгоритм изначально придуман для таких подзадач, где из большого поля данных нужно получить всего одно число, характеризующее всё это поле, например, одну из норм – максимальный модуль элемента или сумму модулей всех элементов. Не будем пока конкретизировать операции, а предположим, что нужно вычислить результат

$$x = a_1 \bullet a_2 \bullet a_3 \bullet \dots a_{n-1} \bullet a_n \quad (1)$$

где ● – обозначение некой ассоциативной операции над данными некоторого типа, к которому как раз и принадлежит любой из элементов a_i. Для вычисления этого выражения последовательно-параллельным методом весь диапазон натуральных чисел *от 1 до n* разбивается на q промежутков – *от* $k_0=1$ *до* k_1, *от* k_1+1 *до* k_2, *...,* *от* $k_{q-1}+1$ *до* $k_q=n$. В каждом из этих промежутков выражение

$$S_j = a_{k_{j-1}+1} \bullet a_{k_{j-1}+2} \bullet a_{k_{j-1}+3} \bullet \dots a_{k_j-1} \bullet a_{k_j} \quad (2)$$

вычисляется последовательно, с возрастанием индекса, а потом последовательно же вычисляется

$$x = S_1 \bullet S_2 \bullet S_3 \bullet \dots S_{q-1} \bullet S_q \quad (3)$$

При возможности деления n на количество устройств q обычно используют равномерное разделение на промежутки. В этом случае граф получающегося алгоритма имеет вид, показанный на **Рис. 1**. Нетрудно понять, что длина его критического пути будет равна

$$\frac{n}{q} + q - 2 \quad (4)$$

* Исследование выполнено при частичной финансовой поддержке гранта Российского научного фонда (проект N14-11-00190).

[1] Полноценным блочным методом можно, на взгляд автора, считать только такой, где количество арифметических операций, используемых одной блочной операцией, существенно больше, чем количество входных и выходных данных. В последовательно-параллельном методе это не так, отсюда и использование слова "эрзац".

то есть при $q = \sqrt{n}$ будет достигнут её минимум, равный

$$2\sqrt{n} - 2 \quad (5)$$

Естественно, что этот метод значительно уступает по длине критического пути графа (по организации вычислений и передач между устройствами он всё же проще) методу сдваивания, показанному на **Рис. 1** и имеющему длину критического пути

$$\lceil \log_2 n \rceil \quad (6)$$

Перед тем, как перейти к более сложным задачам, отметим для себя одну важную вещь. В случае вычисления выражения (1) как последовательно-параллельный метод, так и метод сдваивания использовали одно и то же общее количество операций ●, равное $n - 1$. Избыточных по отношению к исходному последовательному алгоритму вычислений оба они не используют.

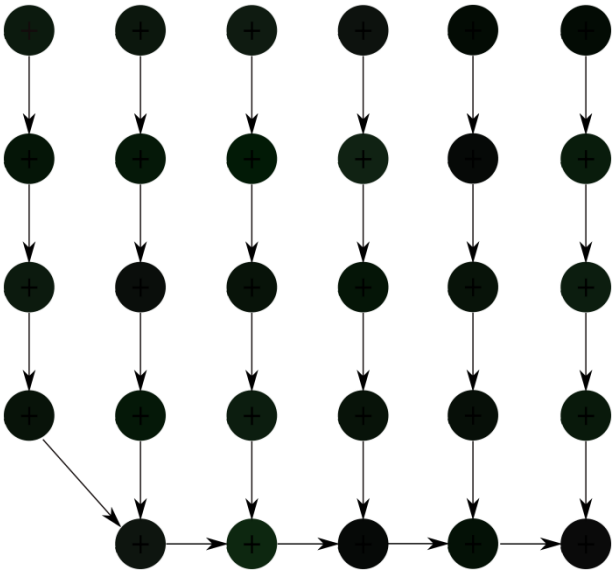

Рис. 1. Последовательно-параллельный метод для вычисления (1) через (2) и (3) при *n=30*

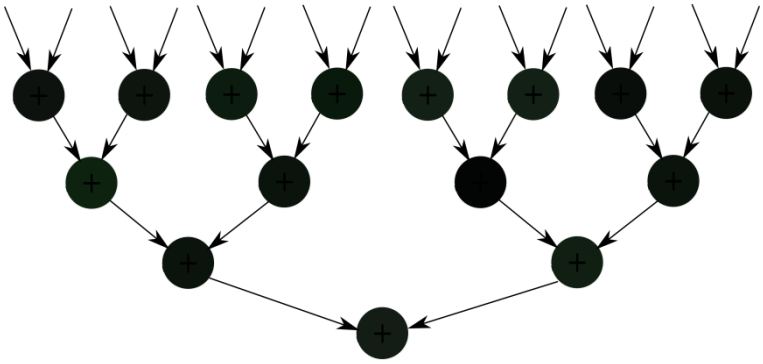

Рис. 2. Метод сдваивания для вычисления (1) при *n=16*

3. Последовательно-параллельный метод для алгоритмов с нужными промежуточными результатами

Перейдём теперь к рассмотрению задачи более сложного типа. Пусть опять ● – обозначение некой ассоциативной операции над данными некоторого типа, к которому принадлежит любой из элементов a_i, и пусть нам нужно вычислить все выражения

$$x_m = a_1 \bullet a_2 \bullet a_3 \bullet \ldots a_{m-1} \bullet a_m \quad (7)$$

для всех возможных значений *m от 1 до n*. Для решения такой задачи с помощью последовательно-параллельного метода мы опять весь диапазон натуральных чисел *от 1 до n* разбиваем на q промежутков – *от* k_0*=1 до* k_1*, от* k_1*+1 до* k_2*, ..., от* k_{q-1}*+1 до* k_q*=n*. В каждом из этих промежутков последовательно определяем выражения

$$\hat{x}_{k_{j-1}+1} = a_{k_{j-1}+1} \quad (8)$$

$$\hat{x}_l = \hat{x}_{l-1} \bullet a_l \quad (8')$$

где l меняется в диапазоне от $k_{j-1}+2$ до k_j. После окончания этого этапа на первом промежутке

$$x_l = \hat{x}_l \quad (9)$$

а на остальных

$$x_l = \hat{x}_{k_{j-1}} \bullet \hat{x}_l \quad (9')$$

где l также меняется в диапазоне от $k_{j-1}+2$ до k_j. Граф вычислений показан на **Рис. 1** Нетрудно видеть, что он, как и в случае с одним нужным результатом, имеет длину критического пути, выведенную в (4) и (5).

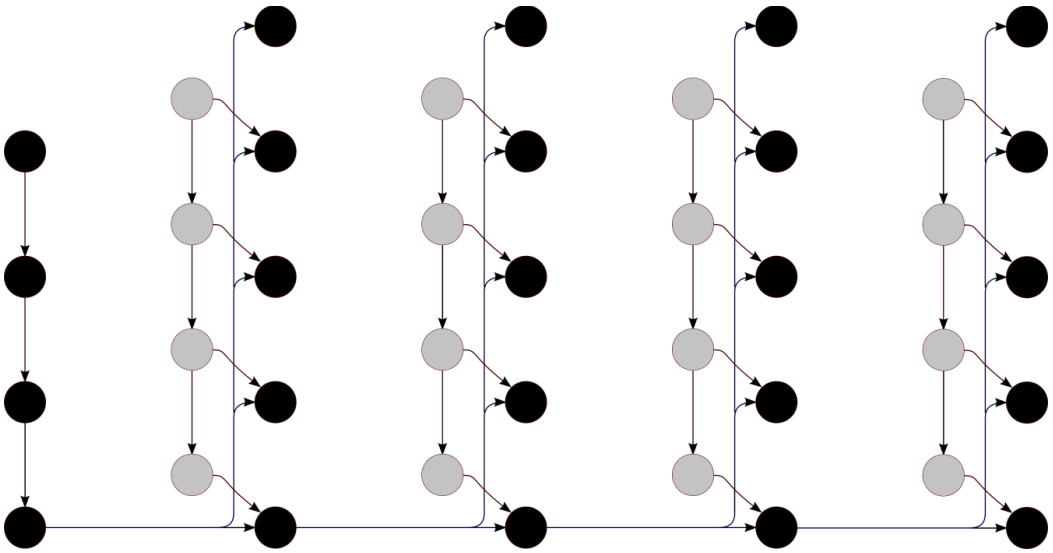

Рис. 3. Алгоритм последовательно-параллельного метода для вычисления всех частичных результатов при *n=25* и равномерном разделении интервала, чёрным обозначены операции, результаты которых нужны на выходе алгоритма. Операции (8) и (9), как пустые, не показаны.

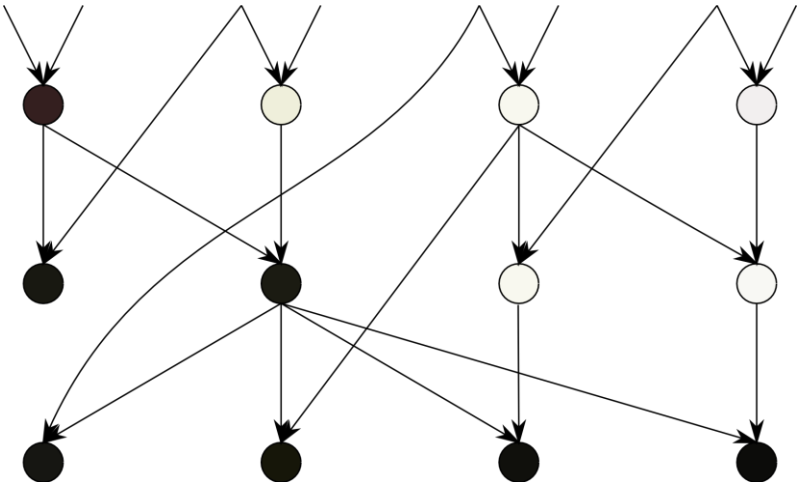

Рис. 4. Алгоритм сдваивания для вычисления всех частичных результатов при *n=8*, чёрным обозначены операции, результаты которых нужны на выходе алгоритма

От внимательного взгляда читателя наверняка не ускользнуло появление в алгоритме[1] «лишних» операций. Для данного метода они необходимы, но по сравнению с исходным – избыточны. При больших *n коэффициент избыточности*[2] данного метода стремится к 2. С одной стороны, это означает, что последовательно-параллельный метод вряд ли кто-то будет приме-

[1] по сравнению с последовательной версией

[2] Коэффициентом избыточности будем называть отношение количества операций нового алгоритма к количеству операций исходного, который мы как раз и распараллеливаем заменой на новый

нять на однопроцессорном компьютере (в отличие от задачи из части 2, где его применение может быть обусловлено не распараллеливанием, а другими соображениями, типа минимизации промахов кэша), и что для его реализации нужно хотя бы несколько параллельных устройств. Однако сравнение с методом сдваивания для данной задачи показывает, что 2 – не такое уж большое число. У метода сдваивания *коэффициент избыточности* равен с точностью до главного члена $\frac{\log_2 n}{2}$ (это несложно показать, если вспомнить, что в методе сдваивания всего $\frac{n \log_2 n}{2}$ операций, а в исходном последовательном – только *n-1*). Вкупе с более простой организацией распределения и пересылки данных это, несмотря на сравнительно большую длину критического пути, делает последовательно-параллельный метод более предпочтительным, чем метод сдваивания, для задач, где нужны и промежуточные результаты.

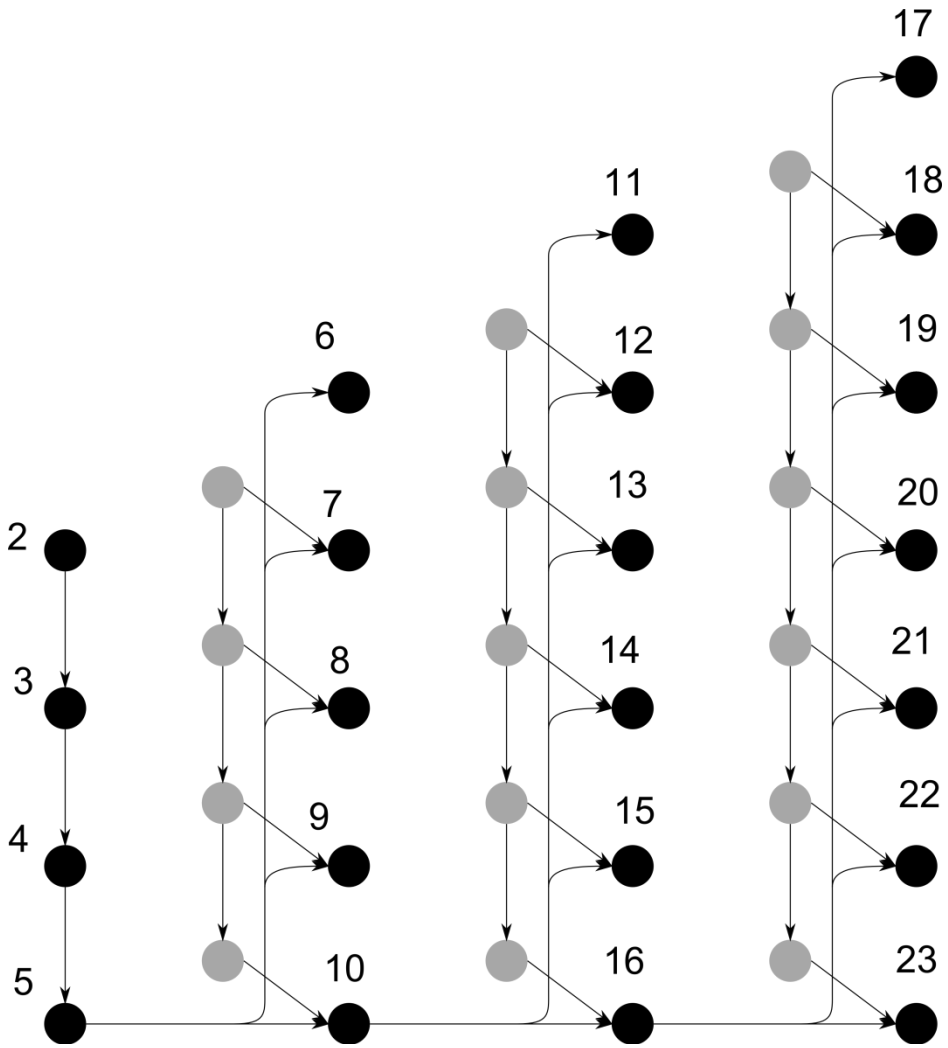

Рис. 5. Алгоритм последовательно-параллельного метода для вычисления всех частичных результатов при *n=23*, чёрным обозначены операции, результаты которых нужны на выходе алгоритма, числа рядом с ними − номера вычисляемых частных выражений

4. Перераспределение интервалов в последовательно-параллельном методе

На **Рис. 1** нетрудно заметить, что при равномерном разделении интервала на части операции, находящиеся на нижней линии (и на критическом пути графа), начиная с третьей, вынуждены "простаивать" в ожидании результата операции слева от себя. Поэтому последовательно-параллельный метод можно оптимизировать, перераспределив интервалы так, чтобы их длины

росли на 1 при возрастании номера. При таком распределении граф алгоритма будет выглядеть как на **Рис. 1**5. Если на первом интервале вычисляется *i* частных выражений, а всего *k* интервалов, то

$$n - 1 = \frac{k(k + i - 1)}{2} \tag{10}$$

а длина критического пути графа алгоритма равна $q = k + i - 1$. Если проделать ряд выкладок, то, как и следовало ожидать, окажется, что наименьшим значение q=k будет при i=1. При таком распределении граф алгоритма будет выглядеть как на **Рис. 16**.

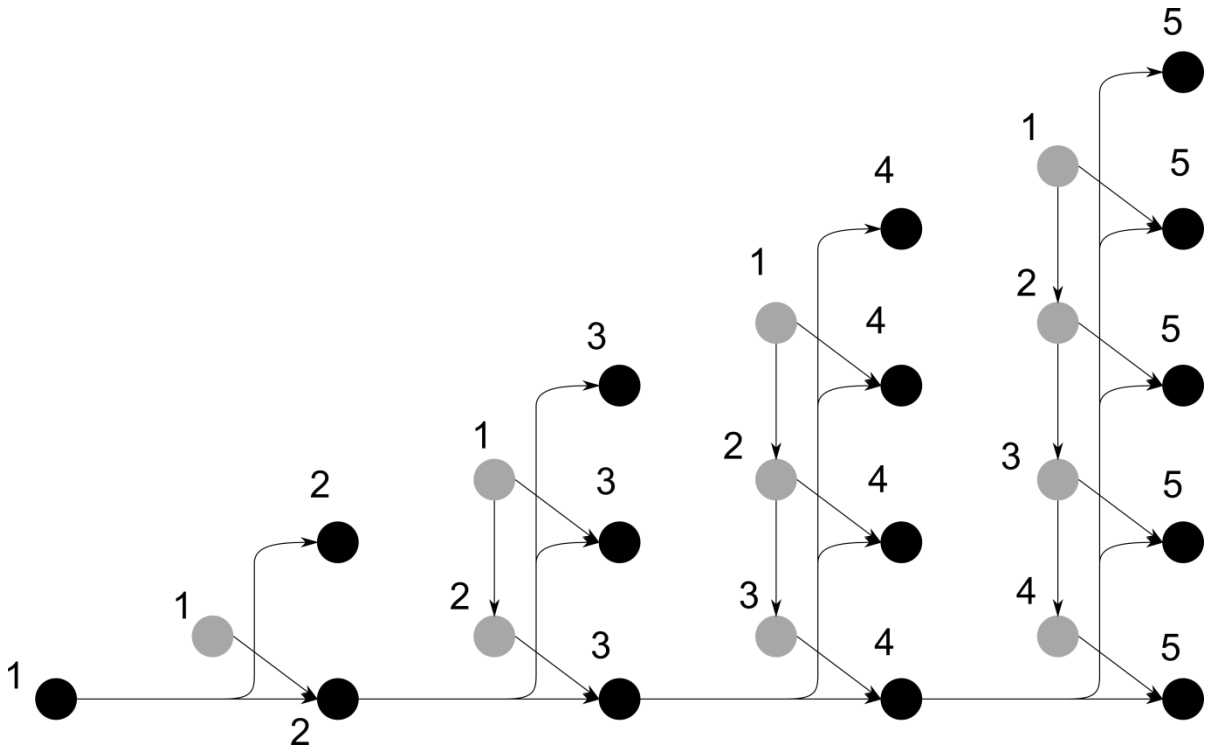

Рис. 6. Алгоритм последовательно-параллельного метода для вычисления всех частичных результатов при *n=16*, чёрным обозначены операции, результаты которых нужны на выходе алгоритма. Числа при вершинах обозначают номер яруса в наискорейшей ярусно-параллельной форме

При указанной разбивке уравнение, связывающее q *с* n, в предположении, что такая разбивка существует, можно записать как

$$n - 1 = \frac{q(q + 1)}{2} \tag{11}$$

или

$$q^2 + q - 2(n - 1) = 0 \tag{12}$$

Решая его, получаем для положительного корня

$$q = \frac{-1 + \sqrt{8(n - 1)}}{2} = \sqrt{2(n - 1)} - \frac{1}{2} \tag{13}$$

что даёт экономию в $\sqrt{2}$ раз по сравнению с равномерным разбиением интервала. Коэффициент избыточности при таком делении интервала остаётся менее 2. Кроме экономии на длине критического пути, при таком разбиении можно выбрать такое распределение операций по ярусам параллельной формы, что ширина всех ярусов окажется одинаковой (это хорошо видно на **Рис. 16**, где количество вершин с одинаковым номером яруса равно 5), что тоже даёт преимущества при реализации метода.

5. Исследование свойств последовательно-параллельного метода и их приложения

Рассмотрим теперь свойства последовательно-параллельного метода, которые можно как-то использовать при конструировании новых алгоритмов. В [6] автором предложен на основе старого метода Стоуна [7], которым на основе приёма сдваивания были распараллелено старое *LU*-разложение трёхдиагональной матрицы [1,2], новый параллельный алгоритм, с помощью которого можно разложить на двухдиагональные множители трёхдиагональную матрицу. При этом характеристики его устойчивости, в отличие от метода сдваивания Стоуна, не хуже, чем у последовательного варианта разложения. Рассмотрим, что именно позволило автору сохранить устойчивость при использовании последовательно-параллельного метода.

Как видно на **Рис. 13, Рис. 15** и **Рис. 16**, последовательно-параллельный метод содержит в себе набор ветвей последовательных вычислений. Это те участки последовательности операций исходного последовательного алгоритма, которые либо оставили без изменений, либо заменили на последовательности более сложных ассоциативных операций. В эти части и можно вставить типичный для последовательных алгоритмов приём обеспечения устойчивости вычислений – нормировку. Её добавление в обычный последовательный алгоритм можно видеть на примере обратной подстановки с нормировкой, например, в [2]. Посмотрим, можно ли использовать её, сконструировав с её помощью по последовательно-параллельной схеме вычислений новый параллельный алгоритм.

6. Пример использования последовательно-параллельного метода

Пусть нам нужно решить систему линейных алгебраических уравнений (здесь и далее будем использовать сокращение СЛАУ)

$$Gx = b \tag{14}$$

где

$$G = \begin{pmatrix} 1 & 0 & 0 & & & & \\ -a_1 & 1 & 0 & \cdots & & 0 & \\ -c_1 & -a_2 & 1 & & & & \\ 0 & -c_2 & -a_3 & & & & \\ & \vdots & & \ddots & & \vdots & \\ & & & & -a_{n-3} & 1 & 0 & 0 \\ & 0 & & \cdots & -c_{n-3} & -a_{n-2} & 1 & 0 \\ & & & & 0 & -c_{n-2} & -a_{n-1} & 1 \end{pmatrix} \tag{15}$$

– ленточная нижняя треугольная матрица с единичной диагональю и с поддиагональной лентой ширины 2, и

$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{pmatrix} \tag{16}$$

– вектор правой части. Если теперь расписать прямую подстановку, то мы получим

$$x_1 = b_1,\ x_2 = b_2 + a_1 x_1, \qquad x_k = b_k + a_{k-1}x_{k-1} + c_{k-2}x_{k-2},\ k = 3, \dots, n \tag{17}$$

или, вводя вектор

$$y_k = \begin{pmatrix} x_k \\ x_{k-1} \\ 1 \end{pmatrix} \tag{18}$$

соотношение

$$y_k = F_k y_{k-1}, \qquad k = 3, \dots, n \tag{19}$$

где

$$F_k = \begin{pmatrix} a_{k-1} & c_{k-2} & b_k \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (20)$$

Если теперь выполнить в (19) подстановку до некоторого $i<k$, то получается

$$y_k = F_k F_{k-1} \dots F_{i+1} y_i \quad (21)$$

Введём обозначение

$$F_k F_{k-1} \dots F_{i+1} = H_{ki} \quad (22)$$

Из вида матрицы F_k видно, что эти матрицы имеют вид

$$H_{i+1\,i} = F_{i+1} = \begin{pmatrix} a_i & c_{i-1} & b_{i+1} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (23)$$

$$H_{i+2\,i} = F_{i+2} F_{i+1} = \begin{pmatrix} a_{i+1}a_i + c_{i-1} & a_{i+1}c_{i-1} & a_{i+1}b_{i+1} + b_{i+2} \\ a_i & c_{i-1} & b_{i+1} \\ 0 & 0 & 1 \end{pmatrix} \quad (24)$$

и далее

$$H_{ki} = \begin{pmatrix} u_{ki} & v_{ki} & w_{ki} \\ u_{k-1\,i} & v_{k-1\,i} & w_{k-1\,i} \\ 0 & 0 & 1 \end{pmatrix} \quad (25)$$

где величины элементов первой и второй строк матрицы связаны рекуррентно по формулам

$$u_{ki} = a_{k-1}u_{k-1\,i} + c_{k-2}u_{k-2\,i} \quad (26)$$

$$v_{ki} = a_{k-1}v_{k-1\,i} + c_{k-2}v_{k-2\,i} \quad (27)$$

$$w_{ki} = b_k + a_{k-1}w_{k-1\,i} + c_{k-2}w_{k-2\,i} \quad (28)$$

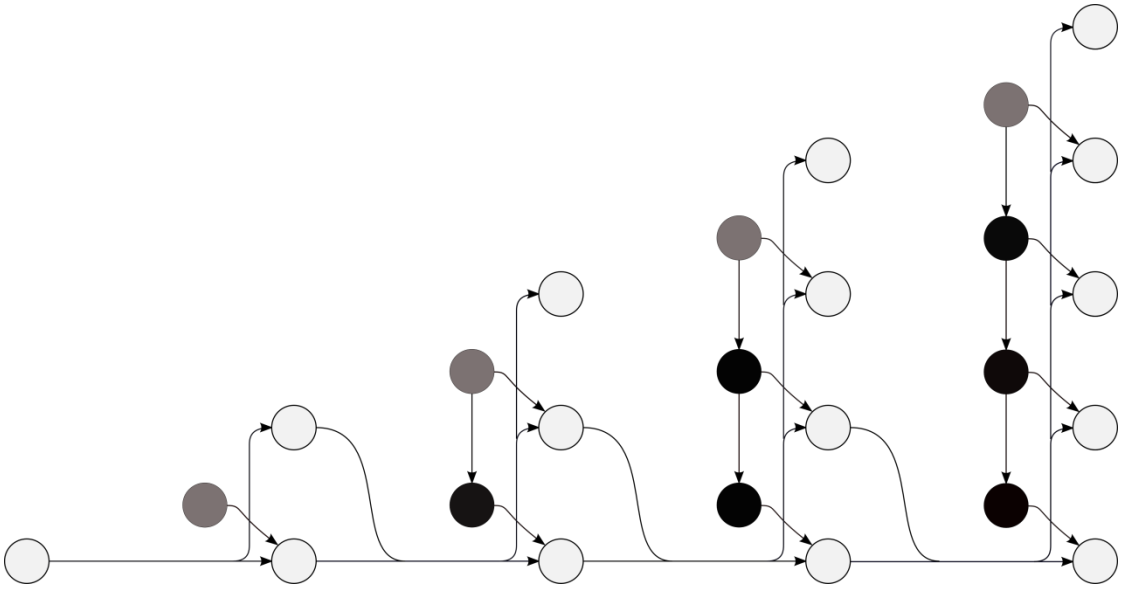

Рис. 7. Алгоритм последовательно-параллельного метода для вычисления решения СЛАУ (1) при *n=16*, степень «тяжести» операций показана градациями серого цвета. Под «тяжестью» здесь и на следующем рисунке автор понимает, как и в [5], реальное количество выполняемых арифметических операций.

Формулы (26) – (28) показывают, что даже при небольших отличиях коэффициентов a_{k-1} или c_{k-2} от единицы (в большую или меньшую сторону), у модулей элементов матриц в длинных ветвях вычислений (когда *k-i* велико) может наблюдаться как рост, так и убывание, что может повредить точности вычислений. В связи с этим целесообразно нормировать вычисления, удерживая хотя бы один из элементов близко к 1. Сделаем это, и теперь можно выполнить вычисления по следующей схеме. Весь диапазон натуральных чисел *от 1 до n* разбиваем на q промежутков – *от* i_0*=1 до* i_1*, от* i_1*+1 до* i_2*, ..., от* i_{q-1}*+1 до* i_q*=n*. В первом промежутке вычисляем значения x_k по формулам (17). В каждом из остальных промежутков последовательно определяем значения коэффициентов матриц (*i считаем равным* i_{j-1}):

$$u_{ii} = 1,\ v_{ii} = 0, w_{ii} = 0,\ z_i = r_i = 1 \quad (29)$$

$$U_{i+1\,i} = a_{\,i},\ V_{i+1\,i} = c_{\,i-1}, W_{i+1\,i} = b_{\,i+1}, \tag{30}$$

Вводим нормировочные коэффициенты

$$z_{i+1} = \frac{1}{U_{i+1\,i}}, \qquad r_{i+1} = U_{i+1\,i} \tag{31}$$

Далее для возрастающих k выполняем

$$U_{ki} = (a_{k-1}U_{k-1\,i} + c_{k-2}u_{k-2\,i})z_k = a_{k-1} + c_{k-2}u_{k-2\,i}z_k \tag{32}$$

$$u_{k-1\,i} = U_{k-1\,i}z_{k-1} = 1 \tag{32'}$$

$$V_{ki} = (a_{k-1}V_{k-1\,i} + c_{k-2}v_{k-2\,i})z_k \tag{33}$$

$$v_{k-1\,i} = V_{k-1\,i}z_{k-1} \tag{33'}$$

$$w_{ki} = b_k + a_{k-1}w_{k-1\,i} + c_{k-2}w_{k-2\,i} \tag{34}$$

И в конце каждого шага обновляем текущие нормировочные коэффициенты

$$z_k = \frac{1}{U_{\,ki}}, \qquad r_k = U_{ki}r_{k-1} \tag{35}$$

После окончания расчётов на промежутках вычисляем все значения x_k по формулам

$$x_k = r_k(x_i + v_{ki}x_{i-1}) + w_{ki} \tag{36}$$

где в качестве i берётся то i_j, которое самое близкое к k снизу. Ясно, что все значения x_{i_j} и x_{i_j-1} можно вычислить только последовательно парами друг за другом. В результате граф алгоритма, если применять неравномерное дробление интервалов, будет как на **Рис. 1**7. При равномерном дроблении интервалов граф будет выглядеть, как на **Рис. 18.**

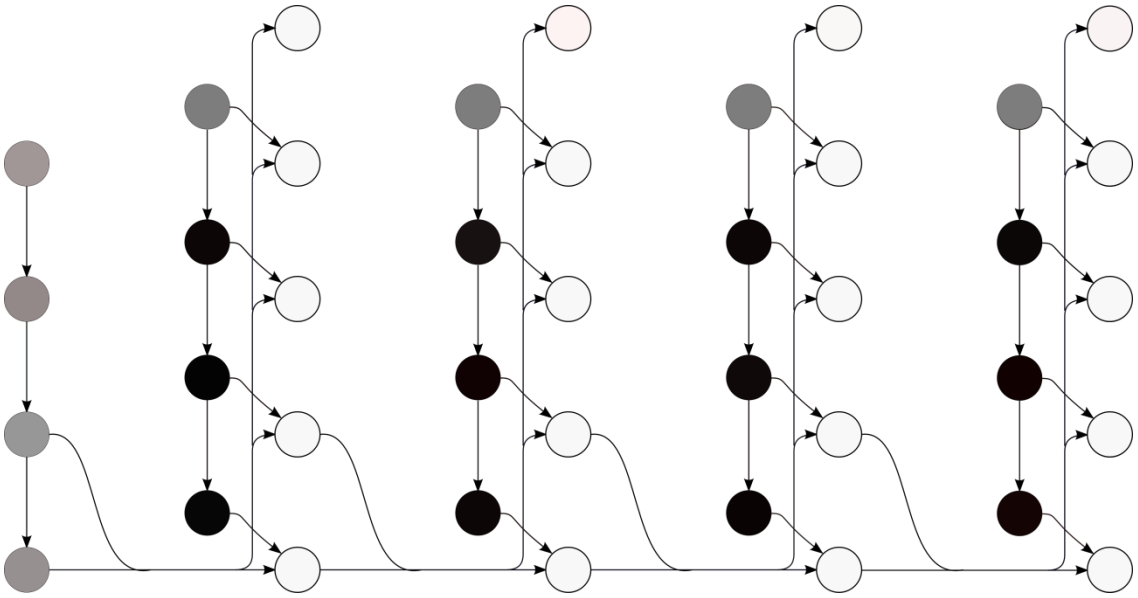

Рис. 8. Алгоритм последовательно-параллельного метода для вычисления решения СЛАУ (1) при *n=30*, степень "тяжести" операций показана градациями серого цвета.

7. Заключение

Использование последовательно-параллельной схемы при конструировании новых параллельных методов позволяет использовать некоторые приёмы работы, которые характерны для традиционных последовательных алгоритмов. Это связано с наличием в схеме достаточно длинных последовательных ветвей, которые имеют вычислительную структуру, во многом заимствованную из последовательного метода.

Поэтому автор рекомендует читателю обратить внимание на последовательно-параллельную схему вычислений там, где схемы сдваивания не дают возможности построить устойчивые алгоритмы. Не исключено, что подобные замены могут помочь и в других аналогичных случаях, где применяется целочисленная дихотомия диапазонов, а не только для ассоциативных операций. Сам автор предполагает заняться ревизией других алгоритмов, опирающихся на сдваивание [4], с целью получения на их основе, возможно, и не столь быстрых, но

более устойчивых параллельных методов, либо алгоритмов с более регулярными графами. Тут важен ещё тот момент, что даже при равных характеристиках устойчивости алгоритмы, опирающиеся на последовательно-параллельную схему, про мнению автора, будут иметь графы, более близкие к регулярным [3], что может позволить более эффективно отображать их на параллельные архитектуры вычислительных систем.

Литература

1. Воеводин В.В. Вычислительные основы линейной алгебры. М.: Наука, 1977.
2. Воеводин В.В., Кузнецов Ю.А. Матрицы и вычисления. М.: Наука, 1984.
3. Воеводин В.В. Математические основы параллельных вычислений // М.: Изд. Моск. ун-та, 1991.
4. Открытая энциклопедия свойств алгоритмов. URL: http://algowiki-project.org (дата обращения: 28.05.2015).
5. Фролов А.В. Принципы построения и описание языка Сигма. Препринт ОВМ АН №236. М.: ОВМ АН СССР, 1989.
6. Фролов А.В. Ещё один метод распараллеливания прогонки с использованием ассоциативности операций // Представлена в качестве доклада на первую объединенную международную конференцию "Суперкомпьютерные дни в России", Москва, 28-29 сентября 2015 г.
7. Stone H.S. An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations // J. ACM, Vol. 20, No. 1 (Jan. 1973), P. 27-38.

Serial-parallel method using for partial associative operation parallelizing

Alexey Frolov

Keywords: Serial-parallel method, associative operations, parallelizing

Serial-parallel method using for partial associative operations is discussed in this paper.