УДК 519.684.4 doi 10.26089/NumMet.v17r322

МЕТОД ДВУХУРОВНЕВОГО РАСПАРАЛЛЕЛИВАНИЯ ПРОГОНКИ ДЛЯ РЕШЕНИЯ ТРЕХДИАГОНАЛЬНЫХ ЛИНЕЙНЫХ СИСТЕМ НА ГИБРИДНЫХ ЭВМ С МНОГОЯДЕРНЫМИ СОПРОЦЕССОРАМИ

А. А. Федоров[1], А. Н. Быков[2]

Приводится описание метода двухуровневого распараллеливания прогонки (на общей памяти средствами OpenMP и на распределенной памяти средствами MPI) для решения трехдиагональных линейных систем, возникающих при моделировании двумерных и трехмерных физических процессов. Анализируются особенности реализации метода как на ЭВМ с универсальными процессорами, так и на гибридных ЭВМ с многоядерными сопроцессорами Intel Xeon Phi. Оценивается арифметическая сложность реализованного метода. Обсуждаются результаты численных экспериментов по исследованию масштабируемости метода.

Ключевые слова: системы линейных алгебраических уравнений, трехдиагональные матрицы, метод прогонки, распараллеливание прогонки, параллельно-конвейерный метод, метод Яненко, параллельные ЭВМ, Intel Xeon Phi.

1. Введение. Неявная конечно-разностная аппроксимация дифференциальных уравнений, описывающих физические процессы, приводит задачу нахождения численного решения к необходимости решения систем линейных алгебраических уравнений (СЛАУ). В случае расщепления многомерного дифференциального оператора на произведение одномерных операторов матрицы СЛАУ, соответствующие каждому одномерному оператору, имеют специальный (трехдиагональный) вид. Такое расщепление осуществимо для регулярных сеток и используется в ряде программных комплексов и методик, например в методике РАМЗЕС-КП [1]. Для нахождения решения рассматриваемых СЛАУ используется метод прогонки.

Метод прогонки является прямым методом и привлекателен своей экономичностью и простотой реализации в последовательном режиме. Появление ЭВМ, использующих многоядерные сопроцессоры [2] или графические ускорители [3], актуализирует задачу распараллеливания метода прогонки. При этом наличие в устройствах возможности использования общей и распределенной памяти порождает необходимость двухуровневого распараллеливания метода прогонки, т.е. распараллеливания как на общей памяти, так и на распределенной памяти.

В англоязычной литературе известно множество статей на данную тему. Среди них можно выделить следующие: в [4, 5] предложены решатели, использующие сочетание параллельной циклической редукции (PCR) [6] и правой прогонки [7] с механизмами принятия решений для переключения с PCR на метод правой прогонки, чтобы сбалансировать параллелизм и вычислительную сложность. Компания NVIDIA выпустила в свет трехдиагональный решатель, основанный на сочетании циклической редукции (CR) [6] и алгоритма PCR библиотеки CUSPARSE [8]. В [9–11] предложены трехдиагональные решатели для графических ускорителей, использующих CR, а в [12] — решатель на основе PCR. В [13] введено сочетание алгоритмов PCR и правой прогонки. В [14] предложен гибридный метод, включающий в себя метод правой прогонки, CR, PCR и метод рекурсивного удвоения (RD) [15]. В [16] предложен вычислительно устойчивый масштабируемый решатель на основе алгоритма SPIKE [17, 18]. В [19] тоже на основе алгоритма SPIKE представлен решатель для сопроцессоров Intel Xeon Phi без использования распараллеливания на распределенной памяти.

В русскоязычной литературе известен метод, предложенный Н. Н. Яненко [20]. Этот метод позволяет редуцировать исходную систему с большим числом неизвестных к системе с числом неизвестных, равным числу процессоров. Редуцированная система, состоящая из гранично-процессорных точек, решается методом прогонки. Чтобы распараллелить процесс решения этой системы, можно применить такие методы, как метод встречной прогонки и метод параллельно-циклической редукции. Кроме того, известен

[1] Российский федеральный ядерный центр — Всероссийский научно-исследовательский институт экспериментальной физики (РФЯЦ–ВНИИЭФ), просп. Мира, 37, 607188, г. Саров; мл. науч. сотр., e-mail: android.f1@yandex.com

[2] Российский федеральный ядерный центр — Всероссийский научно-исследовательский институт экспериментальной физики (РФЯЦ–ВНИИЭФ), просп. Мира, 37, 607188, г. Саров; начальник отдела, e-mail: ban3101@mail.ru

параллельно-конвейерный метод [21, 22], предназначенный для решения нескольких трехдиагональных СЛАУ, возникающих при решении двумерных и трехмерных задач.

В работе [23] показано, как метод Яненко удалось применить при работе на ЭВМ с графическими ускорителями. В работе [24] приведено сравнение параллельно-конвейерного метода с фиксированным числом порций и метода Яненко, на втором этапе которого работал метод правой прогонки, и рассмотрены результаты эффективности распараллеливания не более чем на 20 универсальных процессорах.

В настоящей статье приводится описание метода Яненко, параллельно-конвейерного метода и комбинированного метода, представляющего собой метод Яненко, на втором этапе которого работает параллельно-конвейерный метод встречной прогонки с автоматическим подбором числа порций. Кроме того, обсуждаются оценки арифметической сложности методов и особенности их реализации для работы на ЭВМ с многоядерными сопроцессорами, а также анализируются результаты численного исследования эффективности распараллеливания комбинированного метода и сравнения его с параллельно-конвейерным методом.

2. Параллельно-конвейерный метод (ПКМ). Для удобства дальнейшего изложения материала статьи назовем линейные системы с трехдиагональными матрицами системами трехточечных линейных алгебраических уравнений. Основная идея метода состоит в том, что процессор, выполнив прямой или обратный ход прогонки по части точек, находящихся на нем, передает данные соседнему процессору, чтобы тот мог продолжить прогонку, а сам в это время начинает обрабатывать следующую порцию точек. Если исключить время разгона и торможения конвейера, то все остальное время в счете задействованы все процессоры. Поясним вышесказанное с помощью рис. 1. На рисунке конвейер состоит из 3 процессоров, строки на рисунке (сверху вниз) соответствуют последовательным моментам времени, стрелки вправо — протягиванию прямого хода прогонки в одном направлении, а влево — в другом. Светло-серым цветом отмечены процессоры, протянувшие прогонку в одном направлении, темно-серым — в обоих.

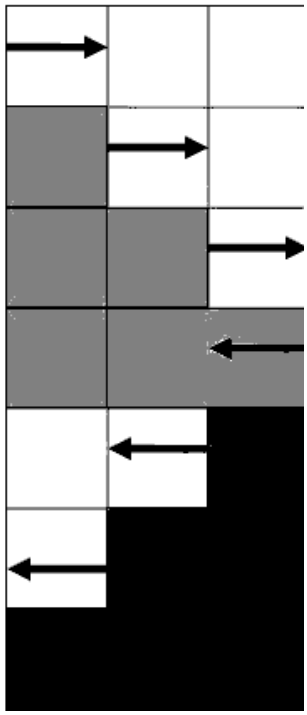

Рис. 1. Схема параллельно-конвейерного алгоритма

Следует отметить, что число порций, определяющее число тактов конвейера, подбирается автоматически во время решения задачи на ЭВМ исходя из времени счета на предыдущем временно́м шаге задачи. В случае, когда число порций становится равным единице, ПКМ вырождается в метод правой прогонки, когда в один момент времени работает только один процессор. Подробнее этот метод описан в [22].

3. Метод Яненко. В отличие от параллельно-конвейерного метод Яненко можно применить не только для решения нескольких систем трехточечных уравнений, но и для решения одной системы.

Рассмотрим систему линейных уравнений с трехдиагональной матрицей следующего вида:

$$\begin{aligned} &a_i y_{i-1} + b_i y_i + c_i y_{i+1} = d_i, \quad b_i \neq 0, \quad i = 1, 2, \ldots, N-1, \\ &b_0 y_0 + c_0 y_1 = d_0, \qquad\qquad a_N y_{N-1} + b_N y_N = d_N. \end{aligned} \tag{1}$$

Для простоты пусть на каждом процессоре будет одинаковое количество точек $m = K/M$, где K — число неизвестных (в нашем случае $K = N+1$) и M — число процессоров, индексация будет глобальная. Таким образом, на отдельном процессоре с номером j будет находиться лишь часть уравнений системы (1) с номерами от $(j-1)*m+1$ до $j*m$, где j — номер процессора. Обозначим y_{j*m} через z_j, $j = 0, \ldots, M$, и будем искать решение системы (1) в виде

$$y_{(j-1)*m+i} = z_{j-1} u_i + z_j v_i + w_i, \quad i = 1, \ldots, m-1, \quad j = 1, \ldots, M, \tag{2}$$

где u, v, w — решения следующих систем уравнений:

$$\begin{aligned} &a_i u_{i-1} + b_i u_i + c_i u_{i+1} = 0, \quad u_{(j-1)*m} = 1, \quad u_{j*m} = 0; \quad i = (j-1)*m+1, \ldots, j*m-1, \\ &a_i v_{i-1} + b_i v_i + c_i v_{i+1} = 0, \quad v_{(j-1)*m} = 0, \quad v_{j*m} = 1; \quad j = 1, \ldots, M, \\ &a_i w_{i-1} + b_i w_i + c_i w_{i+1} = d_i, \quad w_{(j-1)*m} = 0, \quad w_{j*m} = 0; \end{aligned} \tag{3}$$

Решения этих трех систем можно найти методом прогонки, причем независимо на каждом процессоре. Будем называть их предрешениями, а этот этап решения задачи — этапом нахождения предрешений.

В уравнения с номерами $j*m$ из системы (1), имеющих вид

$$a_{j*m}y_{j*m-1} + b_{j*m}y_{j*m} + c_{j*m}y_{j*m+1} = d_{j*m}, \quad j = 0, \ldots, M,$$

подставляем вместо y комбинации (2). Таким образом, получаем систему трехточечных уравнений для нахождения z_j, имеющую следующий вид:

$$A_j z_{j-1} + B_j z_j + C_j z_{j+1} = D_j, \quad j = 1, \ldots, M-1, \quad B_0 z_0 + C_0 z_1 = D_0, \quad A_M z_{M-1} + B_M z_M = D_M \tag{4}$$

с коэффициентами
$$\begin{cases} B_0 = b_0 + c_0 u_1, \quad C_0 = c_0 v_1, \quad D_0 = d_0 - c_0 w_1, \\ A_j = a_{j*m} u_{j*m-1}, \quad B_j = a_{j*m} v_{j*m-1} + b_{j*m} + c_{j*m} u_{j*m+1}, \\ C_j = c_{j*m} v_{j*m+1}, \quad D_j = d_{j*m} - a_{j*m} w_{j*m-1} - c_{j*m} w_{j*m+1}, \quad j = 1, \ldots, M-1, \\ A_M = a_{M*m} u_{M*m-1}, \quad B_M = b_{M*m} + a_{M*m} v_{M*m-1}, \quad D_M = d_{M*m} - a_{M*m} w_{M*m-1}. \end{cases}$$

Назовем этот этап этапом нахождения гранично-процессорных решений. Размерность этой системы уравнений равна количеству процессоров, что существенно меньше количества точек задачи.

На последнем этапе восстанавливаем окончательное решение по формуле (2).

Итак, метод Яненко содержит три этапа: 1) нахождение предрешений, 2) нахождения гранично-процессорных решений, 3) восстановление решения.

Первый и третий этапы выполняются независимо на каждом устройстве (универсальном процессоре, ускорителе или сопроцессоре), а второй этап требует коммуникаций между MPI-процессами. Таким образом, эффективность распараллеливания метода Яненко напрямую зависит от эффективности распараллеливания этого этапа.

Возникает вопрос, каким методом решать систему (4), состоящую из гранично-процессорных точек? В качестве возможных методов рассматривались метод правой прогонки, метод встречной прогонки [7] и метод параллельно-циклической редукции [6]. Рассмотрим эти методы подробнее.

Формулы для метода правой прогонки выглядят следующим образом:

$$\begin{gathered} \alpha_0 = -\frac{c_0}{b_0}, \quad \beta_0 = \frac{d_0}{b_0}, \quad \alpha_i = -\frac{c_i}{b_i + a_i\alpha_{i-1}}, \quad \beta_i = \frac{d_i - a_i\beta_{i-1}}{b_i + a_i\alpha_{i-1}}, \quad i = 1, \ldots, M-1, \\ y_M = \frac{d_M - a_M\beta_{M-1}}{b_M + a_M\alpha_{M-1}}, \quad y_i = \alpha_i y_{i+1} + \beta_i, \quad i = M-1, \ldots, 0. \end{gathered} \tag{5}$$

Для метода встречной прогонки формулы выглядят так:

$$\begin{gathered} \alpha_{1,0} = -\frac{c_0}{b_0}, \quad \beta_{1,0} = \frac{d_0}{b_0}, \quad \alpha_{1,i} = -\frac{c_i}{b_i + a_i\alpha_{1,i-1}}, \quad \beta_{1,i} = \frac{d_i - a_i\beta_{1,i-1}}{b_i + a_i\alpha_{1,i-1}}, \quad i = 1, \ldots, k, \\ \alpha_{2,M} = -\frac{a_M}{b_M}, \quad \beta_{2,M} = \frac{d_M}{b_M}, \quad \alpha_{2,i} = -\frac{a_i}{b_i + c_i\alpha_{2,i+1}}, \quad \beta_{2,i} = \frac{d_i - c_i\beta_{2,i+1}}{b_i + c_i\alpha_{2,i+1}}, \quad i = M-1, \ldots, k+1, \\ y_k = \frac{\beta_{1,k} + \alpha_{1,k}\beta_{2,k+1}}{1 - \alpha_{1,k}\alpha_{2,k+1}}, \quad y_i = \begin{cases} \alpha_{1,i}y_{i+1} + \beta_{1,i}, & i = k-1, \ldots, 0, \\ \alpha_{2,i}y_{i-1} + \beta_{2,i}, & i = k+1, \ldots, M. \end{cases} \end{gathered} \tag{6}$$

Особенность этого метода заключается в том, что во время его работы в параллельном режиме загружены одновременно два процессора (за исключением этапа разрешения встречной прогонки, когда работает только один процессор) в отличие от метода правой прогонки [7], когда одновременно может работать только один процессор. Это является очевидным преимуществом для повышения эффективности распараллеливания метода Яненко.

Теперь рассмотрим метод параллельно-циклической редукции. Формулы преобразования коэффициентов матрицы системы уравнений (4), описанные в работе [6], на первом шаге редукции примут следующий вид в наших обозначениях:

$$\begin{gathered} B_0^1 = -\frac{C_0}{B_1}A_1 + B_0, \quad C_0^1 = -\frac{C_0}{B_1}C_1, \quad D_0^1 = -\frac{C_0}{B_1}D_1 + D_0, \quad C_i^1 = -\frac{C_i}{B_{i+1}}C_{i+1}, \quad A_i^1 = -\frac{A_i}{B_{i-1}}A_{i-1}, \\ D_i^1 = -\frac{A_i}{B_{i-1}}D_{i-1} - \frac{C_i}{B_{i+1}}D_{i+1} + D_i, \quad B_i^1 = -\frac{A_i}{B_{i-1}}C_{i-1} - \frac{C_i}{B_{i+1}}A_{i+1} + B_i, \quad i = 1, \ldots, M-1, \\ A_M^1 = -\frac{A_M}{B_{M-1}}A_{M-1}, \quad B_M^1 = -\frac{A_M}{B_{M-1}}C_{M-1} + B_M, \quad D_M^1 = -\frac{A_M}{B_{M-1}}D_{M-1} + D_M. \end{gathered} \tag{7}$$

После такого преобразования мы будем иметь две системы трехточечных уравнений, которые можно решать независимо на разных процессорах: одна из них для четных неизвестных, а другая для нечетных неизвестных. Применяя к этим двум системам преобразования по формулам (7), мы будем иметь четыре системы трехточечных уравнений, которые можно решать одновременно. Этот процесс продолжается до тех пор, пока в каждой из получившихся систем не останется по одному уравнению с одним неизвестным, из которого мы и найдем решение системы.

Особенность метода параллельно-циклической редукции заключается в том, что во время работы в параллельном режиме одновременно загружены почти все процессоры (за исключением тех, где уже найдено решение системы уравнений).

4. Вычисление арифметической сложности методов. Оценим арифметическую сложность описанных выше методов при решении одной системы линейных уравнений.

Поскольку параллельно-конвейерный метод применяется при решении множества систем методом правой прогонки по формулам (5), то число арифметических операций в алгоритме при решении одной системы совпадает с числом арифметических операций в методе правой прогонки, а именно равно $8K-7$, где K — число неизвестных в системе (1) [25].

Для того чтобы оценить, насколько хорошо метод Яненко будет работать в параллельном режиме, вычислим количество операций, которые нужно сделать при нахождении решений системы (1) методом Яненко, а при нахождении решений системы (4) на втором этапе метода Яненко — методом правой прогонки.

Число арифметических операций в методе складывается как сумма операций на всех этапах метода, описанных ранее. Искомое число есть $S = s_1 + s_{2,1} + s_{2,2} + s_3$, где s_1 — число операций на этапе нахождения предрешений, $s_{2,1}$ — число операций в расчете коэффициентов матрицы системы для нахождения гранично-процессорных решений, $s_{2,2}$ — число операций в методе прогонки на этом этапе, s_3 — число операций при восстановлении решения исходной системы.

Теперь вычислим количество арифметических операций, которые нужно выполнить в методе Яненко при условии использования метода правой прогонки на втором этапе.

Сначала заметим, что если решать системы (3) методом прогонки, не учитывая особенностей этих систем, то мы получим число арифметических операций $s_1' = 3 * (8K-7) = 24K-21$.

Покажем, как можно упростить нахождение предрешений, которые будем искать методом правой прогонки по формулам (5). Обозначим прогоночные коэффициенты первой системы (относительно u) через $\alpha_{1,i}$, $\beta_{1,i}$, второй системы (относительно v) через $\alpha_{2,i}$, $\beta_{2,i}$, третьей системы (относительно w) через $\alpha_{3,i}$, $\beta_{3,i}$.

Заметим, что все коэффициенты $\alpha_{1,i}$, $\alpha_{2,i}$, $\alpha_{3,i}$ равны между собой, так как

$$\alpha_{1,i} = \alpha_{2,i} = \alpha_{3,i} = -\frac{c_i}{b_i + a_i\alpha_{1,i-1}},$$

при этом $\alpha_{1,(j-1)*m} = \alpha_{2,(j-1)*m} = \alpha_{3,(j-1)*m} = 0$ (из (3)). Обозначим $\alpha_{1,i} = \alpha_{2,i} = \alpha_{3,i} = \alpha_i$.

Кроме того, заметим, что $\beta_{2,i} = 0$. Покажем это методом индукции. Сначала вычислим этот коэффициент на левой границе задачи:

$$\beta_{2,(j-1)*m+1} = \frac{d_{(j-1)*m+1} - a_{(j-1)*m+1}\beta_{2,(j-1)*m}}{b_{(j-1)*m+1} + a_{(j-1)*m+1}\alpha_{(j-1)*m}} = \\ = \frac{0 - a_{(j-1)*m+1}v_{(j-1)*m}}{b_{(j-1)*m+1} + a_{(j-1)*m+1}\alpha_{(j-1)*m}} = \frac{-a_i \cdot 0}{b_{(j-1)*m+1} + a_i\alpha_{(j-1)*m}} = 0.$$

Пусть теперь $\beta_{2,i} = 0$, тогда найдем значение $\beta_{2,i+1}$:

$$\beta_{2,i+1} = \frac{d_{i+1} - a_{i+1}\beta_{2,i}}{b_{i+1} + a_{i+1}\alpha_i} = \frac{0 - a_i \cdot 0}{b_{i+1} + a_{i+1}\alpha_i} = 0, \quad i = (j-1)*m+1, \ldots, j*m-2.$$

Используя полученные выше утверждения, формулы прогоночных коэффициентов для нахождения предрешений будут выглядеть следующим образом:

$$\alpha_i = -\frac{c_i}{b_i + a_i\alpha_{i-1}}, \quad \beta_{1,i} = \frac{d_i - a_i\beta_{1,i-1}}{b_i + a_i\alpha_{i-1}} = \frac{-a_i\beta_{1,i-1}}{b_i + a_i\alpha_{i-1}}, \quad \beta_{2,i} = 0, \quad \beta_{3,i} = \frac{d_i - a_i\beta_{3,i-1}}{b_i + a_i\alpha_{i-1}}.$$

При этом формулы нахождения предрешений принимают вид

$$u_i = \alpha_i u_{i+1} + \beta_{1,i}, \quad v_i = \alpha_i v_{i+1}, \quad w_i = \alpha_i w_{i+1} + \beta_{3,i}, \quad i = (j-1)*m+1, \ldots, j*m-1, \quad j = 0, \ldots, M.$$

Таким образом, для прямого хода прогонки необходимо число операций, равное

$$s_{1,1} = (2 + 1 + 2 + 0 + 3) \cdot (m - 1) = 8(m - 1),$$

а для обратного хода прогонки — $s_{1,2} = (2 + 1 + 2) \cdot (m - 1) = 5(m - 1)$.

Для всего этапа требуется $s_1 = s_{1,1} + s_{1,2} = 8(m - 1) + 5(m - 1) = 13(m - 1)$ операций, что гораздо меньше, чем $s'_1 = 24 * (m + 1) - 21 = 24m + 3$.

Иными словами, учитывая структуру систем уравнений, нам удалось уменьшить число арифметических операций почти в 2 раза по сравнению с трехкратным применением метода правой прогонки для решения трех систем уравнений.

В итоге число операций на всех этапах в методе Яненко (при использовании правой прогонки на втором этапе) выглядит следующим образом (из (4) число неизвестных равно $M + 1$):

$$s_1 = 13M(m - 1), \quad s_{2,1} = 5 + 10(M - 1) + 5 = 10M, \quad s_{2,2} = 8(M + 1) - 7 = 8M + 1, \quad s_3 = 4M(m - 1),$$
$$S = 13M(m - 1) + 18M + 1 + 4M(m - 1) = 17M(m - 1) + 18M + 1.$$

В случае, если число процессоров равно 1 ($M = 1, m = K$), то $S = 17(K - 1) + 18 + 1 = 17K + 2$ для системы (1). Видно, что при работе на одном процессоре метод Яненко будет проигрывать методу правой прогонки, в котором $S = 8K - 7$.

Теперь попробуем оценить арифметическую сложность алгоритмов, которые мы планируем применить на втором этапе метода Яненко (встречная прогонка, параллельно-циклическая редукция). Получаемые нами в дальнейшем оценки будут составлять слагаемое $s_{2,2}$ в вышеприведенной формуле арифметической сложности метода Яненко. Из (4) число уравнений в системе на этом этапе равно $M + 1$.

Из (6) видно, что число арифметических операций для метода встречной прогонки равно

$$2 + 6k + 2 + 6(M - k - 1) + 5 + 2k + 2(M - k) = 8M + 3,$$

что немногим больше числа операций в методе правой прогонки ($8M + 1$).

Теперь рассмотрим метод параллельно-циклической редукции. Из (7) видно, что на этапе преобразования коэффициентов матрицы для гранично-процессорных точек число операций будет выражаться величиной $S_{2,1} = 6 + (2 + 1 + 4 + 1 + 4)(M - 1) + 6 = 6 + 12M - 12 + 6 = 12M$. После таких преобразований мы получаем две независимые системы трехточечных уравнений, которые можно решать одновременно на разных процессорах (с четными и нечетными номерами, соответственно). Формулы преобразований будут теми же, но изменится индексация. Для системы, в которой на четных местах стоят ненулевые коэффициенты, диапазон изменения индекса i следующий: $i = 0, 2, \ldots, 2 * [M/2]$, а для системы, в которой на нечетных местах стоят ненулевые коэффициенты, — $i = 1, 3, \ldots, 2 * [(M - 1)/2] + 1$. В этом случае число арифметических операций будет равно

$$S_{2,2} = 12[M/2] + 12\big[(M - 1)/2\big] = 12\Big(\big(M/2 - \{M/2\}\big) + \Big((M - 1)/2 - \big\{(M - 1)/2\big\}\Big)\Big) =$$
$$= 12\Big(\big(M/2 + (M - 1)/2\big) - \Big(\{M/2\} + \big\{(M - 1)/2\big\}\Big)\Big) = 12\big((M - 0.5) - 0.5\big) = 12\,(M - 1)\,,$$

где $[M]$ — целая часть числа M, а $\{M\}$ — дробная часть числа M.

На следующем этапе мы уже будем иметь 4 системы трехточечных уравнений, которые можно решать независимо. Аналогично вышеописанному имеем

$$S_{2,3} = 12[M/4] + 12\big[(M - 1)/4\big] + 12\big[(M - 2)/4\big] + 12\big[(M - 3)/4\big] =$$
$$= 12\big(M/4 + (M - 1)/4 + (M - 2)/4 + (M - 3)/4\big) -$$
$$- 12\Big(\{M/4\} + \big\{(M - 1)/2\big\} + \big\{(M - 2)/4\big\} + \big\{(M - 3)/4\big\}\Big) =$$
$$= 12(M - 6/4) - 12(3/4 + 2/4 + 1/4 + 0) = 12(M - 3).$$

Этот процесс продолжается до тех пор, пока в каждой из получившихся систем не останется по одному уравнению с одним неизвестным, из которого мы и найдем решение нашей системы. Число таких итераций будет $H = \big[\log_2(M + 1)\big] + d$, где $d = 0$, если M является степенью двойки, и $d = 1$, если M не

является степенью двойки. Тогда общее число арифметических операций в этом методе будет равно

$$S_2 = S_{2,1} + S_{2,2} + \ldots + S_{2,H} = 12(M + M - 1 + M - 3 + \ldots + M - (2H - 3)) =$$
$$= 12HM - 12\big(1 + 3 + ... + (2H - 3)\big) = 12HM - 12\Big(\big((1 + 2H - 3)/2\big) * (H - 1)\Big) =$$
$$= 12HM - 12(H - 1)^2 = 12\Big(\big[\log_2(M + 1)\big] + d\Big) * M - 12\Big(\big[\log_2(M + 1)\big] + d - 1\Big)^2.$$

Окончательно $S_2 = 12M\big[\log_2(M + 1)\big] - 12\Big(\big[\log_2(M + 1)\big] + d - 1\Big)^2 + 12dM$, M из (4), $d = 0$, если M является степенью двойки, $d = 1$, если M не является степенью двойки. Очевидно, что это гораздо больше, чем в методе встречной прогонки, но здесь практически все процессоры работают независимо (за исключением тех случаев, когда на процессоре уже найдено решение системы уравнений).

Сведем полученные результаты для методов распараллеливания прогонки в один список.

1. *Метод правой прогонки.* Число арифметических операций: $8M + 1$; число независимо работающих MPI-процессов: 1.

2. *Метод встречной прогонки.* Число арифметических операций: $8M + 3$; число независимо работающих MPI-процессов: 2 (почти всегда).

3. *Метод параллельно-циклической редукции.* Число арифметических операций:

$$12M\big[\log_2(M + 1)\big] - 12\Big(\big[\log_2(M + 1)\big] + d - 1\Big)^2 + 12dM,$$

$d = 0$, если M является степенью двойки, $d = 1$, если M не является степенью двойки; число независимо работающих MPI-процессов: $M - 1$ (почти всегда).

При трехмерной процессорной декомпозиции M — число процессоров вдоль направления прогонки.

Из полученных оценок следует, что неизвестно, какой метод прогонки будет эффективнее работать на втором этапе метода Яненко: метод встречной прогонки (число операций меньше, но и число независимо работающих MPI-процессов мало) либо метод параллельно-циклической редукции (число операций существенно больше, но и число независимо работающих MPI-процессов почти максимально). При работе на графических ускорителях метод параллельно-циклической редукции на втором этапе метода Яненко оказался лучше метода встречной прогонки примерно на 20%.

5. Реализация двухуровневого распараллеливания метода прогонки. Реализация на общей памяти была выполнена с использованием стандарта OpenMP, а реализация на распределенной памяти — с использованием стандарта MPI.

Как уже было отмечено, при решении двумерных или трехмерных задач, использующих метод расщепления, возникает множество независимых систем трехточечных уравнений. При решении задач на параллельных ЭВМ как с общей, так и распределенной памятью на каждом MPI-процессе лежит часть уравнений таких систем, и чтобы найти их решения, мы применяем метод Яненко, на втором этапе которого в силу простоты реализации был использован метод встречной прогонки. При этом каждый OpenMP-поток выполняет вычисления для нескольких систем уравнений. Для повышения эффективности распараллеливания при решении множества систем трехточечных уравнений решено было использовать параллельно-конвейерный метод на втором этапе метода Яненко, т.е. множество систем (4) разбивается на порции, в каждой из которых OpenMP-потоки выполняли арифметическую работу для нескольких систем. Следует отметить, что здесь, как и в [22], был реализован автоматический подбор числа порций конвейера в зависимости от времени работы. А именно:

1) на первом шаге задаем начальное число порций точек, которые необходимо обработать за один такт конвейера, равным 1, т.е. все линии точек на процессоре будут обработаны за 1 такт конвейера;

2) на каждом последующем шаге число порций увеличивается на 1;

3) если на текущем шаге время работы метода прогонок стало больше, чем на предыдущем, то число порций уменьшается на 1;

4) на каждом последующем шаге это число будет уменьшаться до тех пор, пока время прогонок на текущем шаге снова не станет больше, чем на предыдущем; тогда это число начнет увеличиваться.

6. Численные исследования эффективности распараллеливания. В этом разделе приводятся результаты численных экспериментов для определения следующих характеристик рассмотренных методов:

— эффективность распараллеливания (масштабируемость) методов как на ЭВМ с универсальными процессорами, так и на гибридных ЭВМ с сопроцессорами Intel Xeon Phi;

— ускорение программы на гибридных ЭВМ с сопроцессорами Intel Xeon Phi по сравнению с программой на ЭВМ с универсальными процессорами (вычисляется как отношение времени работы программы на ЭВМ с универсальными процессорами к времени работы программы на ЭВМ с сопроцессорами при использовании одного и того же числа вычислительных узлов).

Тестирование проведено на трехмерной задаче газовой динамики с неявной схемой аппроксимации дифференциальных уравнений на кубической трехмерной сетке, в каждом направлении было взято 200 точек. Таким образом, общее число точек задачи равно $200 \times 200 \times 200 = 8\ 000\ 000$ точек.

Задача была просчитана в различных сочетаниях количества OpenMP-потоков и MPI-процессов. Было получено, что физические результаты расчета не зависят от соотношения числа OpenMP-потоков и MPI-процессов.

Будем учитывать только время решения СЛАУ методом прогонки без учета времени расчета коэффициентов трехдиагональной матрицы.

Тестирование проводилось на вычислительном комплексе Центра компетенций и обучения (ЦКО) [26]. Этот комплекс занимает 39-ю строчку списка “50 наиболее мощных компьютеров СНГ” (в редакции от 28.09.2015) [27]. В каждом вычислительном узле находилось 2 CPU Intel Xeon E5-2680 (10 вычислительных ядер в каждом) и 2 Intel Xeon Phi coprocessor 7120P (61 ядро в каждом). Распараллеливание внутри устройств выполнено на общей памяти с использованием OpenMP, а между устройствами — с помощью MPI (при использовании библиотеки S-MPI [28]).

Поскольку, как было отмечено выше, эффективность распараллеливания метода Яненко напрямую зависит от метода распараллеливания решения системы на втором этапе, то сначала мы сравнили масштабируемость второго этапа метода Яненко с разными алгоритмами распараллеливания прогонки.

Масштабирование проводилось методом деления — при увеличении числа MPI-процессов в 2 раза число точек задачи не менялось (сильная масштабируемость).

На 64-х универсальных узлах (процессорная декомпозиция задачи — $8 \times 4 \times 4$) второй этап с методом встречной прогонки выполнился в 1,28 раза быстрее, чем с методом правой прогонки. А параллельно-конвейерный метод встречной прогонки выполнился в 1,46 раза быстрее, чем метод встречной прогонки без конвейерности.

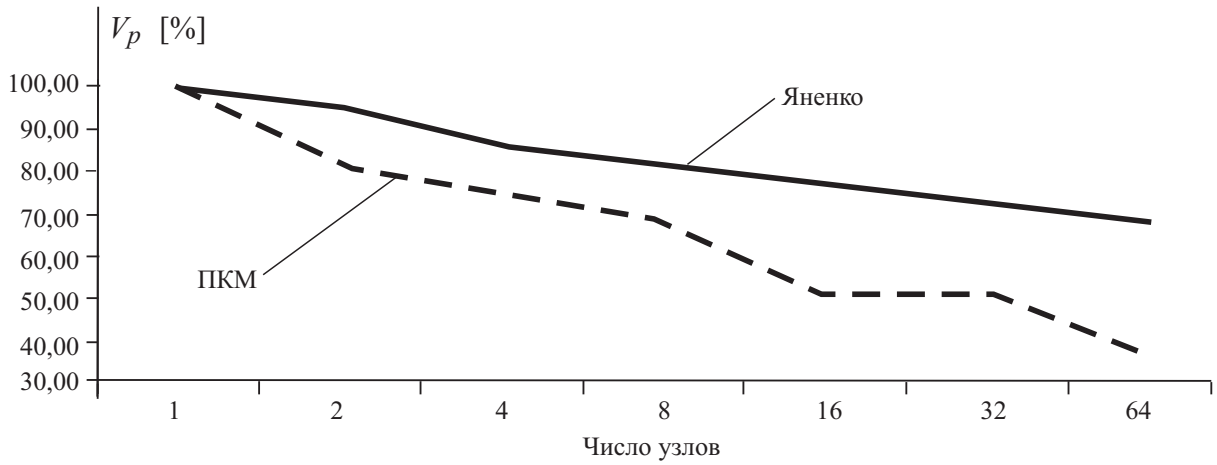

Рис. 2. Эффективность распараллеливания методов прогонки в зависимости от числа универсальных узлов (сильная масштабируемость)

Сравним масштабируемость метода Яненко с параллельно-конвейерным методом встречной прогонки на втором этапе и параллельно-конвейерного метода на универсальной архитектуре. Введем следующие обозначения:

ПКМ — время работы параллельно-конвейерного метода [сек.];

Яненко — время работы метода Яненко [сек.];

$V_p = \big(t_1/(pt_p)\big) \cdot 100\%$ — эффективность распараллеливания при сильной масштабируемости [%], где

t_1 — время работы программы на одном вычислительном узле;

t_p — время работы программы на p вычислительных узлах.

На рис. 2 приведена эффективность распараллеливания методов прогонки с помощью различных

методов в зависимости от числа универсальных узлов.

Теперь будем масштабировать задачу методом умножения — при увеличении числа MPI-процессов в 2 раза число точек задачи также увеличивается в 2 раза (слабая масштабируемость).

На рис. 3 приведена эффективность распараллеливания методов прогонки с помощью различных методов в зависимости от числа универсальных узлов.

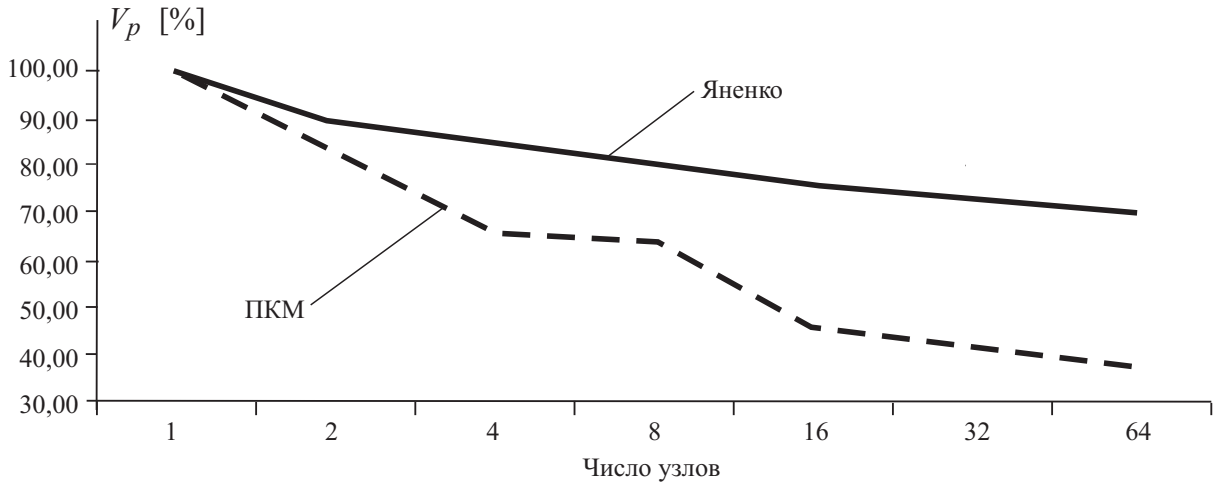

Рис. 3. Эффективность распараллеливания методов прогонки в зависимости от числа универсальных узлов (слабая масштабируемость)

Введем следующие обозначения:

$V_p = \left(t_1/t_p\right) \cdot 100\%$ — эффективность распараллеливания при слабой масштабируемости [%], где

t_1 — время работы программы на одном вычислительном узле;

t_p — время работы программы на p вычислительных узлах.

На рис. 2 и 3 можно заметить, что на универсальной архитектуре метод Яненко работает быстрее, чем ПКМ (более чем в 1,8 раза на 64-х узлах), т.е. метод Яненко на данной задаче показал себя лучше, чем параллельно-конвейерный метод. При этом эффективность распараллеливания метода Яненко составила не ниже 70% на 64-х универсальных узлах. Теперь посмотрим, как ведет себя реализованный метод Яненко на ЭВМ с использованием сопроцессоров. В табл. 1 приведена эффективность распараллеливания между сопроцессорами в native-режиме (слабая масштабируемость). Из этой таблицы следует, что эффективность распараллеливания на 4-х сопроцессорах (92,83) выше, чем на 4-х универсальных узлах на рис. 2 (ниже 70%).

Таблица 1
Эффективность распараллеливания на сопроцессорах в native-режиме

Число сопроцессоров	V_p, [%]
1	100,00
2	96,32
4	92,83

Таблица 2
Эффективность распараллеливания и ускорение от использования сопроцессоров в симметричном режиме

Число гибридных узлов	V_p] [%]	Ускорение от использования сопроцессоров
1	100,00	2,20
2	95,28	2,16

В табл. 2 приведены значения эффективности распараллеливания (V_p) и ускорения (слабая масштабируемость) от использования сопроцессоров в симметричном режиме для реализованного метода.

Из табл. 2 видно, что реализованный метод имеет достаточно высокую эффективность распараллеливания (95% на двух гибридных узлах по сравнению с 80% на двух универсальных узлах) и ускорение от использования сопроцессоров на двух узлах составило 2.16 раза.

7. Заключение. В данной работе выполнена реализация метода распараллеливания прогонки для решения множества систем трехточечных уравнений, возникающих при моделировании двумерных и трехмерных физических процессов, на гибридных ЭВМ с сопроцессорами Intel Xeon Phi, имеющих как общую,

так и распределенную память. Метод заключается в том, что для распараллеливания на распределенной памяти применяется метод Яненко со встречной прогонкой на втором этапе. Для распараллеливания множества встречных прогонок реализован параллельно-конвейерный метод с автоматическим подбором количества порций. При распараллеливании на общей памяти каждый OpenMP-поток решает несколько систем трехточечных уравнений. На тестовой задаче показана независимость решения от числа MPI-процессов и OpenMP-потоков. С точки зрения эффективности распараллеливания метод Яненко выигрывает у параллельно-конвейерного метода на универсальной архитектуре, при этом эффективность распараллеливания на универсальной ЭВМ составила около 70% на 64-х узлах. Получено ускорение счета в 2.16 раза от использования четырех сопроцессоров на двух вычислительных узлах.

В дальнейшем планируется попробовать применить метод параллельно-циклической редукции на втором этапе метода Яненко, чтобы исследовать возможность повышения эффективности распараллеливания, и использовать векторизацию, чтобы повысить ускорение от использования сопроцессоров.

Статья рекомендована к публикации Программным комитетом Международной научной конференции “Параллельные вычислительные технологии” (ПаВТ–2016; http://agora.guru.ru/pavt2016).

СПИСОК ЛИТЕРАТУРЫ

1. *Быков А.Н., Веселов В.А., Воронин Б.Л., Ерофеев А.М.* Методика РАМЗЕС-КП для расчета пространственных движений многокомпонентных теплопроводных сред в эйлерово-лагранжевых координатах // Труды РФЯЦ-ВНИИЭФ. 2008. Вып. 13. 50–57.
2. *Jeffers J., Reinders J.* Intel Xeon Phi coprocessor high-performance programming. San Francisco: Morgan Kaufmann, 2013.
3. NVIDIA. Графические ускорители Tesla для серверов (http://www.nvidia.ru/object/tesla-server-gpus-ru.html).
4. *Davidson A., Zhang Y., Owens J.D.* An auto-tuned method for solving large tridiagonal systems on the GPU // IPDPS'11 Proc. 2011 IEEE Int. Parallel & Distributed Processing Symp. Washington, DC: IEEE Press, 2011. 956–965 (http://idav.ucdavis.edu/func/return_pdf?pub_id=1052).
5. *Kim H.-S., Wu S., Chang L.-W., Hwu W.-M.* A scalable tridiagonal solver for GPUs // ICPP '11 Proc. 2011 Int. Conf. on Parallel Processing. Washington, DC: IEEE Press, 2011. 444–453.
6. *Hockney R.W., Jesshope C.R.* Parallel computers: architecture, programming and algorithm. Hilger: Bristol, 1988.
7. *Самарский А.А.* Введение в численные методы. М: Наука, 1982.
8. NVIDIA Corporation. NVIDIA CUDA Sparse Matrix Library (cuSPARSE) (https://developer.nvidia.com/cusparse).
9. *Sengupta S., Harris M., Zhang Y., Owens J.D.* Scan primitives for GPU computing (http://www.idav.ucdavis.edu/func/return_pdf?pub_id=915).
10. *Goddeke D., Strzodka R.* Cyclic reduction tridiagonal solvers on GPUs applied to mixed-precision multigrid // IEEE Transactions on Parallel and Distributed Systems. 2011. **22**, N 1. 22–32.
11. *Davidson A., Owens J.D.* Register packing for cyclic reduction: a case study (http://idav.ucdavis.edu/func/return_pdf?pub_id=1053).
12. *Egloff D.* High performance finite difference PDE solvers on GPUs (http://www.gpucomputing.net/sites/default/files/papers/1380/fdm_gpu.pdf).
13. *Sakharnykh N.* Efficient tridiagonal solvers for ADI methods and fluid simulation (http://on-demand.gputechconf.com/gtc/2010/presentations/S12015-Tridiagonal-Solvers-ADI-Methods-Fluid-Simulation.pdf).
14. *Zhang Y., Cohen J., Owens J.D.* Fast tridiagonal solvers on the GPU // ACM SIGPLAN Notices. 2010. **45**, N 5. 127–136.
15. *Stone H.S.* An efficient parallel algorithm for the solution of a tridiagonal linear system of equations // Journal of the ACM. 1973. **20**, N 1. 27–38.
16. *Chang L.-W., Stratton J.A., Kim H.-S., Hwu W.-M.W.* A scalable, numerically stable, high-performance tridiagonal solver using GPUs // SC'12 Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis. Los Alamitos: IEEE Press, 2012. Article No. 27.
17. *Polizzi E., Sameh A.H.* A parallel hybrid banded system solver: the SPIKE algorithm // Parallel Computing. 2006. **32**, N 2. 177–194.
18. *Polizzi E., Sameh A.* SPIKE: a parallel environment for solving banded linear systems // Computers and Fluids. 2007. **36**, N 1. 113–120.
19. *Venetis I., Sobczyk A., Kouris A., et al.* A general tridiagonal solver for coprocessors: adapting g-SPIKE for the Intel Xeon Phi (http://www.researchgate.net/publication/282286515).
20. *Яненко Н.Н., Коновалов А.Н., Бугров А.Н., Шустов Г.В.* Об организации параллельных вычислений и “распараллеливании” прогонки // Численные методы механики сплошной среды. 1978. **9**, № 7. 139–146.
21. *Povitsky A.* Parallelization of the pipelined Thomas algorithm. ICASE Report N 98-48. Hampton: NASA Langley Research Center, 1998.

22. *Сапронов И.С., Быков А.Н.* Параллельно-конвейерный алгоритм // Атом. 2009. № 44. 24–25.
23. *Быков А.Н., Ерофеев А.М., Сизов Е.А., Федоров А.А.* Метод распараллеливания прогонки на гибридных ЭВМ // Вычислительные методы и программирование. 2013. **14**. 43–47.
24. *Ильин С.А., Старченко А.В.* Распараллеливание схемы покомпонентного расщепления для численного решения уравнения теплопроводности // Параллельные вычислительные технологии (ПаВТ-2015): Труды международной научной конференции (Екатеринбург, 31 марта–2 апреля 2015 г.). Челябинск: Издательский центр ЮУрГУ, 2015. 399–402.
25. *Вержбицкий В.М.* Вычислительная линейная алгебра. М: Директ-Медиа, 2013. 296–299.
26. Центр компетенций и обучения. Предоставление вычислительных ресурсов ЦКО (http://compcenter.org/predostavlenie-vychislitelnyh-resur).
27. Топ-50. 23-я редакция от 28.09.2015 (http://top50.supercomputers.ru/?page=archive&rating=23).
28. *Воронов Г.И., Трущин В.Д., Шумилин В.В., Ежов Д.В.* Программный комплекс S-MPI для обеспечения разработки, оптимизации и выполнения высокопараллельных приложений на суперкомпьютерных кластерных системах // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2013. Вып. 3. 55–60.

Поступила в редакцию
05.06.2016

A Method of Two-Level Parallelization of the Thomas Algorithm for Solving Tridiagonal Linear Systems on Hybrid Computers with Multicore Coprocessors

A. A. Fedorov [1] **and A. N. Bykov** [2]

[1] *Federal Nuclear Center — All-Russian Scientific Research Institute of Experimental Physics, prospekt Mira 37, Sarov, 607188, Russia; Junior Scientist, e-mail: android.f1@yandex.com*

[2] *Federal Nuclear Center — All-Russian Scientific Research Institute of Experimental Physics, prospekt Mira 37, Sarov, 607188, Russia; Ph.D., Head of Division, e-mail: ban3101@mail.ru*

Received June 5, 2016

Abstract: A method of two-level parallelization of the Thomas algorithm for solving tridiagonal linear systems (the thread-level parallelism using OpenMP and the process-level parallelism using MPI) arising when modeling two-dimensional and three-dimensional physical processes is described. The features of its implementation for parallel multiprocessor systems and for hybrid multiprocessor systems with multicore coprocessors Intel Xeon Phi are analyzed. The arithmetic complexity of this method is estimated. Some numerical results obtained when studying its scalability are discussed.

Keywords: systems of linear algebraic equations, tridiagonal matrices, Thomas algorithm, parallelization of Thomas algorithm, parallel-pipeline method, Yanenko's method, parallel computers, Intel Xeon Phi.

References

1. A. N. Bykov, V. A. Veselov, B. L. Voronin, and A. M. Erofeev, "RAMZES-KP Technique for Computation of Multicomponent Heat-Conducting Space Motions in Euler–Lagrange Coordinates," in *Proc. Russian Federal Nuclear Center* (Russian Federal Nuclear Center, Sarov, 2008), Issue 13, pp. 50–57.

2. J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High-Performance Programming* (Morgan Kaufmann, San Francisco, 2013).

3. NVIDIA. Tesla GPU Accelerators for Servers. http://www.nvidia.ru/object/tesla-server-gpus-ru.html. Cited June 17, 2016.

4. A. Davidson, Y. Zhang, and J. D. Owens, "An Auto-Tuned Method for Solving Large Tridiagonal Systems on the GPU," in *IPDPS'11 Proc. 2011 IEEE Int. Parallel & Distributed Processing Symp., Anchorage, USA, May 16–20, 2011* (IEEE Press, Washington, DC, 2011), pp. 956–965.
http://idav.ucdavis.edu/func/return_pdf?pub_id=1052. Cited June 17, 2016.

5. H.-S. Kim, S. Wu, L.-W. Chang, and W.-M. Hwu, "A Scalable Tridiagonal Solver for GPUs," in *ICPP '11 Proc. 2011 Int. Conf. on Parallel Processing, Taipei, Taiwan, September 13–16, 2011* (IEEE Press, Washington, DC, 2011), pp. 444–453.

6. R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming, and Algorithm* (Hilger, Bristol, 1988).

7. A. A. Samarskii, *Introduction to Numerical Methods* (Nauka, Moscow, 1982) [in Russian].

8. NVIDIA Corporation. NVIDIA CUDA Sparse Matrix Library (cuSPARSE). https://developer.nvidia.com/cusparse. Cited June 17, 2016.

9. S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens, "Scan Primitives for GPU Computing," http://www.idav.ucdavis.edu/func/return_pdf?pub_id=915. Cited June 17, 2016.

10. D. Goddeke and R. Strzodka, "Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed-Precision Multigrid," IEEE Trans. Parallel Distrib. Syst. **22** (1), 22–32 (2011).

11. A. Davidson and J. D. Owens, "Register Packing for Cyclic Reduction: A Case Study," http://idav.ucdavis.edu/func/return_pdf?pub_id=1053. Cited June 17, 2016.

12. D. Egloff, "High Performance Finite Difference PDE Solvers on GPUs," http://www.gpucomputing.net/sites/default/files/papers/1380/fdm_gpu.pdf. Cited June 17, 2016.

13. N. Sakharnykh, "Efficient Tridiagonal Solvers for ADI Methods and Fluid Simulation," http://on-demand.gputechconf.com/gtc/2010/presentations/S12015-Tridiagonal-Solvers-ADI-Methods-Fluid-Simulation.pdf. Cited June 17, 2016.

14. Y. Zhang, J. Cohen, and J. D. Owens, "Fast Tridiagonal Solvers on the GPU," ACM SIGPLAN Notices **45** (5), 127–136 (2010).

15. H. S. Stone, "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations," J. ACM **20** (1), 27–38 (1973).

16. L.-W. Chang, J. A. Stratton, H.-S. Kim, and W.-M.W. Hwu, "A Scalable, Numerically Stable, High-Performance Tridiagonal Solver Using GPUs," in *SC'12 Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, USA, November 11–15, 2012* (IEEE Press, Los Alamitos, 2012), Article No. 27.

17. E. Polizzi and A. H. Sameh, "A Parallel Hybrid Banded System Solver: The SPIKE Algorithm," Parallel Comput. **32** (2), 177–194 (2006).

18. E. Polizzi and A. Sameh, "SPIKE: A Parallel Environment for Solving Banded Linear Systems," Comput. Fluids **36** (1), 113–120 (2007).

19. I. E. Venetis, A. Sobczyk, A. Kouris, et al., "A General Tridiagonal Solver for Coprocessors: Adapting g-SPIKE for the Intel Xeon Phi," http://www.researchgate.net/publication/282286515. Cited June 17, 2016.

20. N. N. Yanenko, A. N. Konovalov, A. N. Bugrov, and G. V. Shustov, "Organization of Parallel Computing and the Thomas Algorithm Parallelization," in *Numerical Methods in Continuum Mechanics* (Comput. Center Sib. Branch of USSR Acad. Sci., Novosibirsk, 1978), Vol. 9, Issue 7, pp. 139–146.

21. A. Povitsky, *Parallelization of the Pipelined Thomas Algorithm*, ICASE Report No. 98-48 (NASA Langley Research Center, Hampton, 1998).

22. I. S. Sapronov and A. N. Bykov, "A Parallel-Pipelined Algorithm," Atom, No. 4, 24–25 (2009).

23. A. N. Bykov, A. M. Erofeev, E. A. Sizov, and A. A. Fedorov, "A parallel Sweep Method for Hybrid Supercomputers," Vychisl. Metody Programm. **14**, 43–47 (2013).

24. S. A. Il'in and A. V. Starchenko, "Parallelization of a Componentwise Splitting for the Numerical Solution of the Heat Conduction Equation," in *Proc. Int. Conf. on Parallel Computational Technologies, Ekaterinburg, Russia, March 30–April 3, 2015* (Inst. of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences, Ekaterinburg, 2015), pp. 399–402.

25. V. M. Verzhbitskii, *Numerical Linear Algebra* (Direkt-Media, Moscow, 2013) [in Russian].

26. Education Center. http://compcenter.org/predostavlenie-vychislitelnyh-resur. Cited June 17, 2016.

27. Top-50. http://top50.supercomputers.ru/?page=archive&rating=23. Cited June 17, 2016.

28. G. I. Voronov, V. D. Trushchin, V. V. Shumilin, and D. V. Yezhov, "S-MPI Software System for the Development, Optimization and Execution of Parallel Applications on Supercomputer Clusters," Voprosy Atomnoi Nauki i Tekhniki, Ser.: Mat. Mod. Fiz. Proc., No. 3, 55–60 (2013).