Модифицированный следящий алгоритм для решения нестационарных задач линейного программирования на кластерных вычислительных системах с многоядерными ускорителями

И.М. Соколинская, Л.Б. Соколинский

Южно-Уральский государственный университет

Статья посвящена новой редакции параллельного следящего алгоритма *Pursuit*, предложенного авторами в предыдущих работах. Этот алгоритм использует фейеровские отображения для построения псевдопроекции на многогранник. Алгоритм отслеживает изменения в исходных данных и корректирует процесс вычислений. Предыдущая версия алгоритма предполагала использование следящей области кубической формы с количеством ячеек K по одному измерению. Общее количество ячеек в этом случае составляет K^n, где n – размерность задачи. Это приводит к высокой вычислительной сложности алгоритма. Новая версия алгоритма использует следящую область крестообразной формы с n лучами, содержащую только nK ячеек. Алгоритм ориентирован на кластерные вычислительные системы, оснащенные процессорами Intel Xeon Phi.

Ключевые слова: нестационарная задача линейного программирования, фейеровские отображения, следящий алгоритм, массовый параллелизм, кластерные вычислительные системы, архитектура MIC, Intel Xeon Phi, нативный режим, OpenMP.

1. Введение

В работах [1,2] авторами был предложен следящий алгоритм *Pursuit* решения нестационарных задач линейного программирования большой размерности, ориентированный на кластерные вычислительные системы. Нестационарные задачи линейного программирования большой размерности с быстро меняющимися входными данными достаточно часто встречаются в практике современного экономико-математического моделирования. Одним из примеров таких задач является задача управления портфелем ценных бумаг с использованием методов алгоритмической торговли [3, 4]. В подобных задачах количество переменных и неравенств в системе ограничений может составлять десятки и даже сотни тысяч, а период изменения исходных данных находиться в пределах сотых долей секунды. Разработанный авторами алгоритм *Pursuit* в своей начальной версии предполагал использование следящей области кубической формы с количеством ячеек K по одному измерению. Общее количество ячеек в этом случае составляет K^n, где n – размерность задачи. Это приводит к высокой вычислительной сложности алгоритма на задачах большой размерности. В настоящей работе описывается новая версия алгоритма *Pursuit*, в которой используется следящая область крестообразной формы с n лучами, содержащая только nK ячеек. Базовой частью алгоритма *Pursuit* является подпрограмма вычисления псевдопроекции на многогранник. Псевдопроектирование использует фейеровские отображения для замены операции проектирования на выпуклое множество [6]. Авторами выполнена реализация этого алгоритма на языке C++ с использованием технологии параллельного программирования OpenMP 4.0 [7] и векторных команд Intel C++ Compiler для Xeon Phi [8]. Эффективность реализации алгоритма для сопроцессора Xeon Phi с архитектурой MIC [9] была исследована на модельной масштабируемой задаче линейного программирования. Результаты этих экспериментов приведены в данной статье. Статья организована следующим образом. В разделе 2 приводится формальная постановка задачи линейного программирования, дают-

ся определения фейеровского процесса и операции псевдопроектирования на многогранник. В разделе 3 приводится описание новой версии алгоритма с крестообразной следящей областью. В разделе 4 дается описание головной подпрограммы и подпрограммы вычисления псевдопроекции модифицированного алгоритма с помощью диаграмм деятельности UML. Раздел 5 посвящен исследованию эффективного использования сопроцессоров Intel Xeon Phi для вычисления псевдопроекций. В заключении суммируются полученные результаты и определяются направления дальнейших исследований.

2. Постановка задачи

Задана задача линейного программирования

$$\max\left\{\langle c,x\rangle \,|Ax\le b,\ x\ge 0\right\}. \tag{1}$$

Определим фейеровское отображение $\varphi:\mathbb{R}^n\to\mathbb{R}^n$ следующим образом:

$$\varphi(x)=x-\sum_{i=1}^{m}\alpha_i\lambda_i\frac{\max\left\{\langle a_i,x\rangle-b_i,0\right\}}{\|a_i\|^2}\cdot a_i. \tag{2}$$

Пусть M – многогранник, задаваемый ограничениями задачи линейного программирования (1). Такой многогранник всегда является выпуклым. Известно [5], что φ будет однозначным непрерывным M-фейеровским отображением для любой системы положительных коэффициентов $\{\alpha_i>0\}$ $(i=1,\dots,m)$, таких, что $\sum\limits_{i=1}^{m}\alpha_i=1$, и коэффициентов релаксации $0<\lambda_i<2$. Полагая в формуле (2) $\lambda_i=\lambda$ и $\alpha_i=1/m$ $(i=1,\dots,m)$, получаем формулу

$$\varphi(x)=x-\frac{\lambda}{m}\sum_{i=1}^{m}\frac{\max\left\{\langle a_i,x\rangle-b_i,0\right\}}{\|a_i\|^2}\cdot a_i, \tag{3}$$

которая используется в алгоритме *Pursuit*.

Обозначим

$$\varphi^s(x)=\underbrace{\varphi\dots\varphi}_{s}(x).$$

Под *фейеровским процессом*, порождаемым отображением φ при произвольном начальном приближении $x_0\in\mathbb{R}^n$, будем понимать последовательность $\{\varphi^s(x_0)\}_{s=0}^{+\infty}$. Известно, что указанный фейеровский процесс сходится к точке, принадлежащей множеству M:

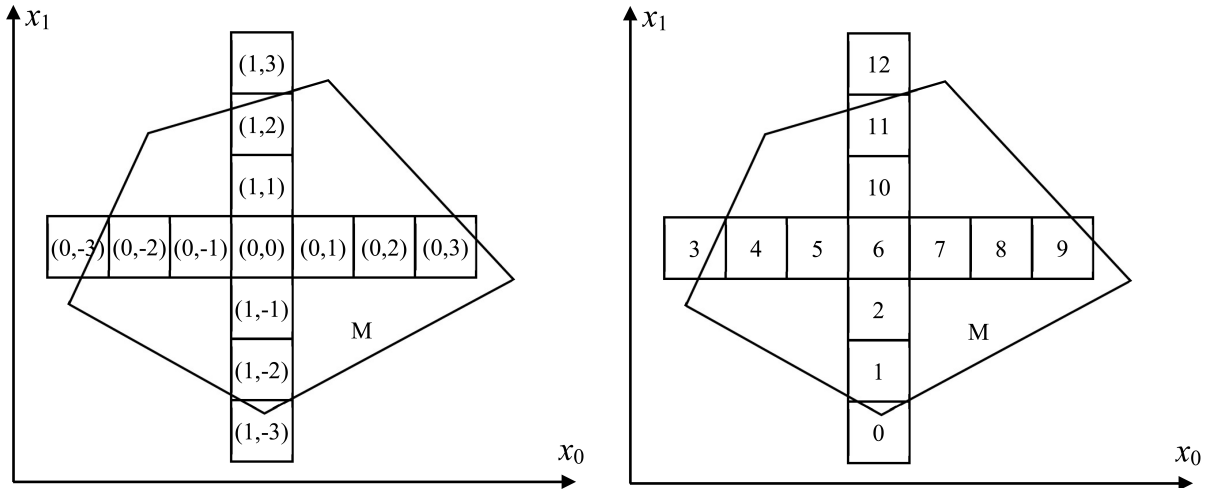
$$\{\varphi^s(x_0)\}_{s=0}^{+\infty}\to\bar{x}\in M. \tag{4}$$

Кратко обозначим это следующим образом: $\lim\limits_{s\to\infty}\varphi^s(x_0)=\bar{x}$.

Под *φ-проектированием* (*псевдопроектированием*) точки $x\in\mathbb{R}^n$ на многогранник M понимается отображение $\pi_M^{\varphi}(x)=\lim\limits_{s\to\infty}\varphi^s(x)$.

3. Описание модифицированного алгоритма

Без ограничения общности мы можем считать, что все процессы происходят в положительной области координат. Пусть – размерность пространства решений. В новой версии алгоритма используется крестообразная следящая область. Такая область состоит из n-мерных ячеек кубической формы, имеющих одинаковые размеры. Ребра всех ячеек сонаправлены с координатными осями. Среди этих ячеек выделяется ячейка, называемая *центральной*. Остальные ячейки образуют осесимметричную крестообразную фигуру вокруг центральной ячейки. Пример крестообразной следящей области для двумерного случая приведен на рис. 1. Пусть K – количество ячеек по одному измерению. В примере на



а) с маркировкой вида (χ, η) б) с линейной нумерацией

Рис. 1. Крестообразная следящая область ($n = 2$, $K = 7$).

рис. 1 $K = 7$. Симметричность следящей области предполагает, что K принимает только нечетные значения. Общее количество ячеек в крестообразной следящей области можно вычислить по следующей формуле:

$$P = n(K - 1) + 1 \tag{5}$$

Каждая ячейка в крестообразной следящей области однозначно идентифицируется *маркером* — парой целых чисел (χ, η) таких, что $0 \le \chi < n$, $|\eta| \le (K - 1)/2$. С неформальной точки зрения χ задает столбик ячеек, сонаправленный координатной оси с индексом χ, а η задает порядковый номер ячейки в столбце относительно центральной ячейки. Далее χ будем называть *измерением*, а η – *индексом* ячейки. Соответствующая маркировка ячеек для двумерного случая приведена на рис. 1 а).

Нулевой вершиной кубической ячейки будем называть вершину, находящуюся ближе всего к началу координат. Пусть $(g_0, \ldots, g_{n-1})$ – декартовы координаты нулевой вершины центральной ячейки. Обозначим через s длину ребра ячейки. Тогда декартовы координаты $(y_0, \ldots, y_{n-1})$ нулевой вершины ячейки (χ, η) определяются следующей формулой

$$y_j = \begin{cases} g_\chi + \eta s, \textit{если } j = \chi \\ g_j, \textit{если } j \ne \chi \end{cases} \tag{6}$$

для всех $j = 0, \ldots, n - 1$.

Неформально работа алгоритма с крестообразной следящей областью может быть описана следующей последовательностью действий.

1. Первоначально выбирается крестообразная следящая область с количеством ячеек по одному измерению K, длиной ребра ячейки s и нулевой вершиной центральной ячейки в точке g таким образом, что центральная ячейка имеет непустое пересечение с многогранником M.

2. В качестве начального приближения выбирается точка $z = g$.

3. В условиях динамического изменения входных данных (A, b, c) для всех ячеек, принадлежащих крестообразной следящей области, вычисляется псевдопроекция из точки z на пересечение ячейки с многогранником M. Если пересечение пусто, то такие ячейки отбрасываются.

4. Если получено пустое множество псевдопроекций, то мы увеличиваем размер ячейки следящей области в w раз и переходим на шаг 3.

5. Если получено непустое множество псевдопроекций, то для каждого измерения находим ячейку, на которой в точке псевдопроекции достигается максимум целевой функции. Остальные ячейки отбрасываются. Для оставшихся ячеек находим их центр масс. В качестве следующего приближения выбираем точку z, совпадающую с найденным центром масс.

6. Если расстояние от центра масс до центральной ячейки следящей области меньше $\frac{1}{4}s$, то длина s ребра ячейки уменьшается в 2 раза.

7. Если расстояние от точки максимума до центра следящей области больше $\frac{3}{4}s$, то длина s ребра ячейки увеличивается в 1.5 раза.

8. Выполняется параллельный перенос крестообразной следящей области таким образом, чтобы ее центральная ячейка оказалась в центре масс, найденном на шаге 5.

9. Переходим на шаг 3.

Псевдопроекции на шаге 3 для различных ячеек следящей области могут вычисляться параллельно без обменов данными между MPI-процессами. При этом будет задействовано P MPI-процессов, где P определяется по формуле (5). Для распределения ячеек по MPI-процессам мы используем линейную нумерацию ячеек. Каждой ячейке крестообразной следящей области присваивается уникальный номер $\alpha \in \{0, \ldots, P-1\}$. Порядковый номер α может быть однозначно преобразован в маркер (χ, η) по следующим формулам[1]:

$$\chi = ||\alpha - K + 1| - 1| \div ((K-1)/2)\,; \tag{7}$$

$$\eta = \operatorname{sign}(\alpha - K + 1) \cdot (((|\alpha - K + 1| - 1) \bmod ((K-1)/2)) + 1)\,. \tag{8}$$

Обратный переход от (χ, η) к α выполняется с помощью формулы

$$\alpha = \eta + \operatorname{sign}(\eta)\chi(K-1)/2 + K - 1. \tag{9}$$

На рис. 1 a) изображена линейная нумерация ячеек, соответствующая маркировке на рис. 1 b).

4. Реализация новой версии следящего алгоритма

В данном разделе описываются изменения в реализации новой версии следящего алгоритма относительно описания, приведенного в работе [2].

4.1. Схема головной подпрограммы

Общая схема головной подпрограммы следящего алгоритма приведена на рис. 2. В цикле *until* с меткой 1 выполняется постоянная корректировка приближенного решения задачи линейного программирования (1), которое вычисляется в векторе $z = (z_0, \ldots, z_{n-1})$ в соответствии с описанием идеи алгоритма в разделе 3. В качестве начального значения z может быть взята произвольная точка.

Головная подпрограмма следящего алгоритма оформляется в виде независимого процесса, который выполняется до тех пор, пока переменная *stop* не примет значение *true* (истина). Начальную установку переменной *stop* в значение *false* (ложь) осуществляет головной процесс, соответствующий основной программе. Он же присваивает переменной *stop* значение *true*, когда вычислительный процесс нужно остановить.

[1] С помощью символа ÷ здесь обозначается целочисленное деление.

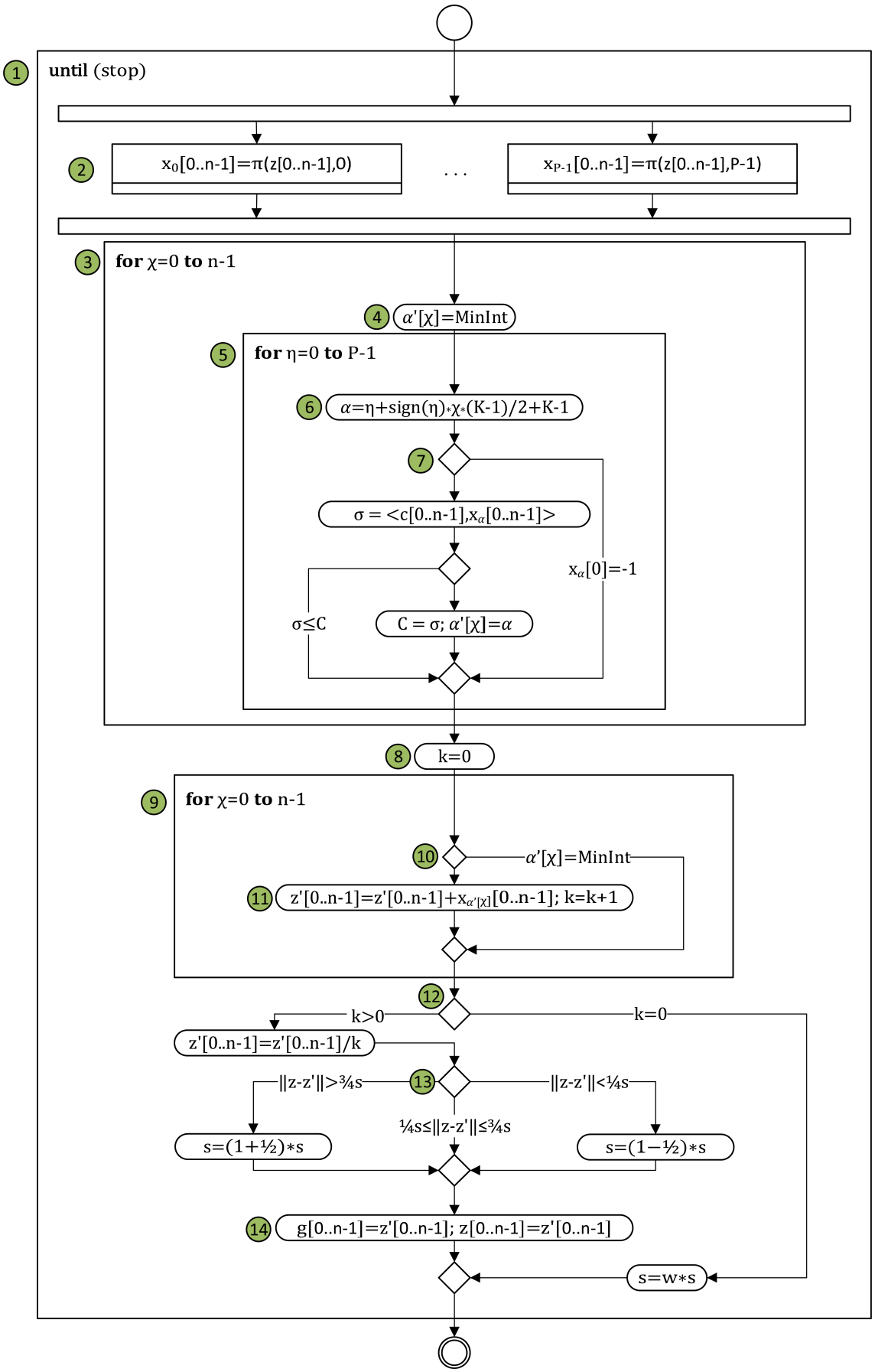


Рис. 2. Головная подпрограмма алгоритма *Pursuit*.

В теле цикла *until* выполняются следующие действия. На шаге 2 организуется K параллельных потоков управления (нитей), которые независимо друг от друга вычисляют псевдопроекции из точки z на пересечение i-той ячейки с многогранником M ($i = 0, \ldots, P-1$). Напомним, что P равно количеству MPI-процессов, и в соответствии с формулой (5) равно количеству ячеек в крестообразной следящей области. Схема подпрограммы π вычисления псевдопроекции детально описана в разделе 4.2.

В цикле *for* с меткой 3 для каждого измерения $\chi = 0, \ldots, n-1$ вычисляется порядковый номер α'_χ ячейки данного измерения, на которой достигается максимум C целевой функции. Это делается во вложенном цикле с меткой 5. Для корректной работы цикла 5 переменной α'_χ на шаге 4 присваивается начальное значение *MinInt*, соответствующее минимальному машинному значению целого типа. На шаге 6 по значениям χ и η вычисляется порядковый номер α ячейки с маркером (χ, η) по формуле (9).

Подпрограмма π, вычисляющая точку $x_\alpha = (x_0, \ldots, x_{n-1})$ псевдопроекции точки z на пересечение многогранника M с ячейкой с номером α, присваивает координате x_0 значение -1 в том случае, когда точка x_α псевдопроекции не принадлежит многограннику M. Эта ситуация возникает в случае, когда пересечение многогранника M с ячейкой с номером α является пустым. Если же x_α принадлежит многограннику, то в силу предположения о том, что все процессы находятся в положительной области координат (см. раздел 3), значение x_0 не может быть отрицательным. Указанное условие проверяется на шаге 7. Случаи $x_0 = -1$ из рассмотрения исключаются. Если при выполнении цикла 5 оказывается, что ни одна из ячеек текущего измерения следящей области не имеет непустого пересечения с многогранником M, то в переменной α'_χ сохраняется значение *MinInt*. Этот факт проверяется на шаге 10.

Далее в цикле 9 вычисляется новое приближенное решение z' задачи (1). В переменной k будет вычисляться количество измерений, по которым следящая область имеет непустые пересечения с многогранником M. Для этого на шаге 8 ей присваивается нулевое значение. На шаге 10 из рассмотрения исключаются измерения, не имеющие пересечения с многогранником M. На шаге 11 значение z' вычисляется как сумма тех точек псевдопроекций, на которых был достигнут максимум целевой функции для соответствующего измерения.

Если на шаге 12 выясняется, что $k = 0$, это означает, что следящая область не имеет непустого пересечения с многогранником M. В этом случае длина s ребра ячейки следящей области увеличиваются в w раз, где w – положительная константа, являющаяся параметром алгоритма, и осуществляется переход к новой итерации цикла с меткой 1. Если же на шаге 12 оказывается, что $k > 0$, то новое приближение z' вычисляется как центр масс точек, отобранных в цикле с меткой 9.

На шаге 13 анализируется, насколько новое приближение z' далеко отстоит от предыдущего приближения z. Если расстояние между новым и предыдущим приближениями превышает $\frac{3}{4}s$, то длина s ребра ячейки увеличивается в 1.5 раза. Если расстояние между новым и предыдущим приближениями меньше $\frac{1}{4}s$, то длина s ребра ячейки уменьшается в 2 раза. Если же отклонение лежит в пределах от $\frac{1}{4}s$ до $\frac{3}{4}s$, то корректировка значения s не производится. Величины 1/4 и 3/4 являются параметрами алгоритма.

На шаге 14 следящая область сдвигается по вектору $(z' - z)$, и в качестве текущего приближения z берется точка z', после чего вычисления продолжаются.

4.2. Схема подпрограммы вычисления псевдопроекции

На рис. 3 приведена схема подпрограммы вычисления псевдопроекции $x = \pi(z, \alpha)$ из точки z на пересечение многогранника M с ячейкой крестообразной следящей области, имеющей порядковый номер α, где α вычисляется по формуле (9). Псевдопроекция вычисляется путем организации фейеровского процесса (4) (см. раздел 2). На шаге 1 выполняется инициализация переменных, необходимых для организации итерационного процесса. В качестве начального значения x берется точка z; с помощью подпрограммы zero (см. рис. 4)

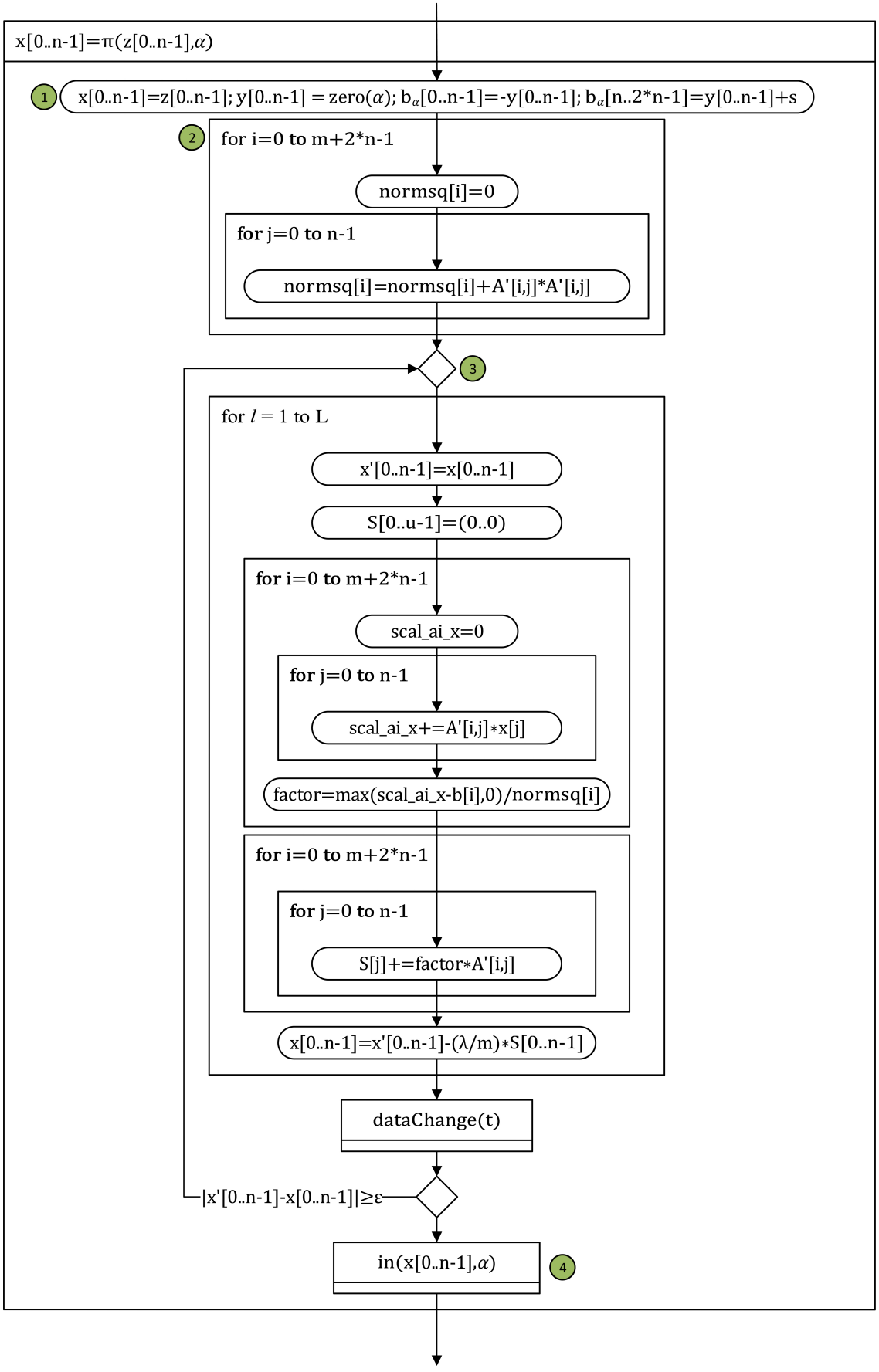


Рис. 3. Схема подпрограммы π вычисления псевдопроекции.

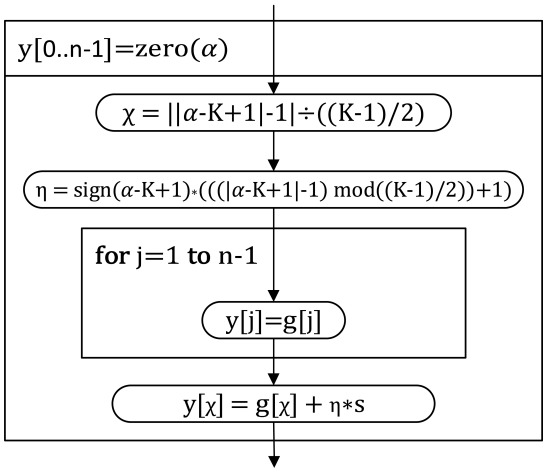


Рис. 4. Подпрограмма вычисления нулевой вершины ячейки с номером α.

$$\begin{cases} x_0 \le 200 \\ x_1 \le 200 \\ \ddots \quad \dots \\ x_{n-1} \le 200 \\ x_0 + x_1 + \dots + x_{n-1} \le 200(n-1) + 100 \\ x_0 + x_1 + \dots + x_{n-1} \ge 100 \\ x_0 \ge 0 \\ x_1 \ge 0 \\ \ddots \quad \dots \\ x_{n-1} \ge 0 \end{cases}$$
$$Q_{\max}(x) = 2x_0 + 2x_1 + \dots + 2x_{n-2} + x_{n-1}$$

Рис. 5. Задача *Model-n*.

вычисляется нулевая вершина y ячейки с номером α; определяется вариативная часть b_α расширенного столбца b' системы ограничений, получаемой при пересечении многогранника M с ячейкой с номером α (см. [2]). В цикле 2 вычисляется $normsq$ – вектор квадратов норм строк расширенной матрицы A': $normsq_i = \|a'_i\|^2$ (см. [2]).

На шаге 3 организуется итерационный процесс вычисления псевдопроекции с использованием формулы (3). Подпрограмма *dataChange* вносит изменения в исходные данные с периодом в t секунд (t – положительное число, которое может принимать значения меньше единицы).

Итерационный процесс заканчивается, когда расстояние между двумя последними приближениями x и x' будет меньше ε. На четвертом шаге подпрограмма *in* (см. [2]) проверяет принадлежность найденной точки псевдопроекции x ячейке с номером α. Если x не принадлежит ячейке с номером α, то $x[0]$ присваивается значение (-1). Константа ε задает малое положительное число, позволяющее корректно обрабатывать приближенные значения.

Схема подпрограммы вычисления нулевой вершины ячейки с порядковым номером α приведена на рис. 4. Вычисления осуществляются с использованием формул (8), (9) и (7).

5. Вычисление псевдопроекций на сопроцессоре Intel Xeon Phi

Наиболее трудоемкой операцией алгоритма *Pursuit* является вычисление песевдопроекций с помощью подпрограммы, описанной в разделе 4.2. С целью достижения максимально высокого быстродействия нами было проведено исследование возможности эффективного использования сопроцессоров Intel Xeon Phi для вычисления псевдопроекций.

При проведении вычислительных экспериментов использовалась сконструированная нами модельная задача линейного программирования *Model-n*, представленная на рис. 5. Для такого типа задач можно легко аналитически вычислить точное решение. Поэтому они хорошо подходят для проверки корректности алгоритма и исследования его масштабируемости.

При написании программы на языке C++ была использована технология программирования OpenMP. Запуск задачи на Xeon Phi осуществлялся в нативном режиме [9]. Для вычислительных экспериментов мы использовали вычислительный комплекс с кластерной архитектурой «Торнадо ЮУрГУ» [10], включающий в себя 384 процессорных узла, соединенных сетями InfiniBand QDR и Gigabit Ethernet. В состав процессорного узла входят два шестиядерных ЦПУ Intel Xeon X5680, ОЗУ 24 Гб и сопроцессор Intel Xeon Phi SE10X (61

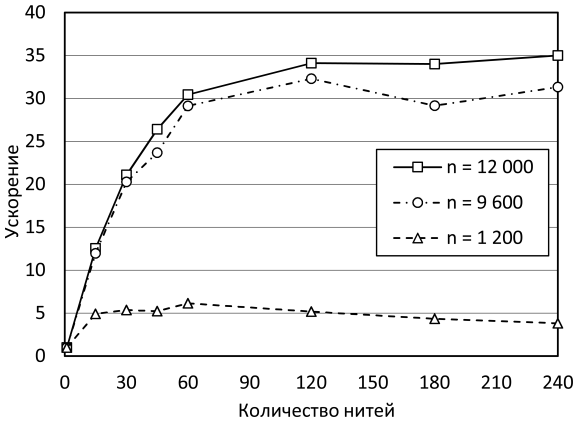


Рис. 6. Ускорение вычисления псевдопроекции на Xeon Phi.

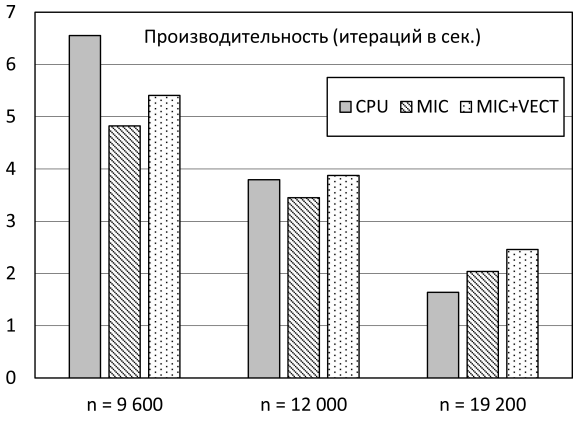


Рис. 7. Сравнение производительности 2×ЦПУ (CPU) и Xeon Phi (MIC).

ядро по 1.1 ГГц), соединенные шиной PCI Express.

В первой серии экспериментов мы исследовали эффективность распараллеливания вычисления псевдопроекции на различном количестве нитей Xeon Phi. Результаты приведены на рис. 6. Псевдопроекции вычислялись на пересечение ячейки с длиной ребра $s = 20$ и координатами нулевой вершины $(100, \ldots, 100)$ с многогранником, задаваемым ограничениями задачи линейного программирования *Model-n*. Задача решалась для размерностей $n = 1200, 9600, 12000$. Графики показывают, что эффективность распараллеливания сильно зависит от размерности задачи. Так, для размерности $n = 1200$ кривая ускорения фактически перестает расти уже после 15 нитей. Это означает, что на задаче такой размерности невозможно загрузить все ядра Xeon Phi. При использовании задачи размерности $n = 9600$ и выше картина меняется. Ускорение становится близким к линейному вплоть до 60 нитей, что совпадает с количеством процессорных ядер Xeon Phi. Затем эффективность распараллеливания снижается, а для размерности $n = 9600$ даже наблюдается деградация производительности. Особенно это заметно для точки, соответствующей использованию 180 нитей. Этот провал объясняется тем, что размерность 9600 не делится нацело на 180, то есть компилятор не может равномерно распределить итерации цикла parallel for между нитями. Та же ситуация имеет место и для точки «45 нитей» при решении задач размерности 1200 и 9600. Однако при увеличении размерности до 12000 эта проблема нивелируется.

Во второй серии вычислительных экспериментов, результаты которых приведены на рис. 7, сравнивалась производительность двух ЦПУ Intel Xeon (CPU) и сопроцессора Intel Xeon Phi. Эксперименты проводились для размерностей $n = 9600, 12000, 19200$. Для сопроцессора Intel Xeon Phi делалось две сборки: без векторизации (MIC) и с векторизацией, включая выравнивание данных (MIC+VECT). Во всех случаях при запуске на центральных процессорах задействовались 12 нитей, а при запуске на сопроцессоре Xeon Phi задействовались 240 нитей. Результаты эксперимента показали, что при размерности задачи $n = 9600$ центральные процессоры по производительности обгоняют сопроцессор Xeon Phi, при размерности $n = 12000$ ЦПУ и сопроцессор демонстрируют примерно равную производительность, а при $n = 19200$ Xeon Phi уже заметно превосходит центральные процессоры. При этом векторизация с выравниванием данных дает прирост производительности в 12%, 12.3% и 20% для размерностей 9600, 12000 и 19200 соответственно. Таким образом, можно сделать вывод, что эффективность использования сопроцессора Xeon Phi растет с увеличением размерности задачи, при этом одновременно растет значимость выравнивания данных и векторизации.

6. Заключение

В работе описана новая версия следящего алгоритма *Pursuit*, предназначенного для решения нестационарных задач линейного программирования большой размерности на современных кластерных вычислительных системах. Отличительной особенностью новой версии является то, что в ней используется следящая область крестообразной формы, состоящая из nK ячеек, где n – размерность задачи, K – количество ячеек по одному измерению. Предыдущая версия алгоритма предусматривала использование следящей области, состоящей из K^n ячеек, что порождало очень высокую вычислительную сложность алгоритма. В работе приведены результаты вычислительных экспериментов на модельной масштабируемой задаче линейного программирования по исследованию эффективности использования сопроцессоров Intel Xeon Phi для вычисления ресурсоемких операций псевдопроектирования. Проведенные исследования показали, что применение сопроцессоров Intel Xeon Phi оказывается эффективным на задачах большой размерности (более 10 000). В плане дальнейших исследований – исследовать эффективность предложенного алгоритма на кластерных вычислительных системах с использованием технологии MPI.

Литература

1. *Соколинская И.М., Соколинский Л.Б.* Решение нестационарных задач линейного программирования большой размерности на кластерных вычислительных системах // Суперкомпьютерные дни в России (Москва, 28–29 сентября 2015). Труды международной конференции. МГУ, 2015. С. 420–427.

2. *Соколинская И.М., Соколинский Л.Б.* Параллельная реализация следящего алгоритма для решения нестационарных задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2016. Т. 5, № 2. С. 15–29. DOI: 10.14529/cmse160202

3. *Дышаев М.М., Соколинская И.М.* Представление торговых сигналов на основе адаптивной скользящей средней Кауфмана в виде системы линейных неравенств // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2013. Т. 2. № 4. С. 103–108.

4. *Ананченко И.В., Мусаев А.А.* Торговые роботы и управление в хаотических средах: обзор и критический анализ // Труды СПИИРАН. 2014. № 3(34). С. 178–203.

5. *Еремин И.И.* Фейеровские методы для задач выпуклой и линейной оптимизации. Челябинск: Изд-во ЮУрГУ, 2009. 200 с.

6. *Ершова А.В., Соколинская И.М.* О сходимости масштабируемого алгоритма построения псевдопроекции на выпуклое замкнутое множество // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. 2011. № 37 (254), вып. 10. С. 12–21.

7. OpenMP Application Program Interface. Version 4.0 – July 2013. URL: `http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf` (дата обращения: 28.06.2016).

8. *Supalov A., Semin A., Klemm M., Dahnken Ch.* Optimizing HPC Applications with Intel Cluster Tools. Apress, 2014. 269 p. DOI: 10.1007/978-1-4302-6497-2

9. *Thiagarajan S.U., Congdon C., Naik S., Nguyen L.Q.* Intel Xeon Phi coprocessor developer's quick start guide. White Paper. Intel, 2013. URL: `https://software.intel.com/sites/default/files/managed/ee/4e/`

intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf (дата обращения: 28.06.2016).

10. *Kostenetskiy P.S., Safonov A.Y.* SUSU Supercomputer Resources // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576, CEURWS 2015. P. 561—573.

Revised Pursuit Algorithm for Solving Unstable Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators

I.M. Sokolinskaya, L.B. Sokolinsky

South Ural State University

The paper is devoted to new edition of the parallel *Pursuit* algorithm proposed the authors in previous works. *Pursuit* algorithm uses Fejer's mappings for building pseudo-projection on polyhedron. The algorithm tracks changes in input data and corrects the calculation process. The previous edition of the algorithm assumes using a pursuit region of cube shape with the number of K cells in one dimension. The total number of the cells is K^n where n is the problem dimension. This results in high computational complexity of the algorithm. The new edition uses a pursuit region of asterix shape with one beam per dimension. Such region consists of only nK cells. The new algorithm is implemented on cluster computing system with Xeon Phi processors.

Keywords: unstable linear programming problem, Fejer's mappings, pursuit algorithm, massive parallelism, cluster computing systems, MIC architecture, Intel Xeon Phi, native mode, OpenMP.

References

1. *Sokolinskaya I., Sokolinsky L.* Solving unstable linear programming problems of high dimension on cluster computing systems // Proceedings of the 1st Russian Conference on Supercomputing – Supercomputing Days (RuSCDays 2015). Moscow, Russian Federation, September 28–29, 2015. CEUR Workshop Proceedings. 2015. Vol. 1482. P. 420–427.

2. *Sokolinskaya I., Sokolinsky L.* Implementation of Parallel Pursuit Algorithm for Solving Unstable Linear Programming Problems // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). Arkhangelsk, Russia, March 29–31, 2016. CEUR Workshop Proceedings. 2016. Vol. 1576. P. 685–698.

3. *Dyshaev M.M., Sokolinskaya I.M.* Predstavlenie torgovykh signalov na osnove adaptivnoy skol'zyashchey sredney Kaufmana v vide sistemy lineynykh neravenstv [Representation of trading signals based on Kaufman's adaptive moving average as a system of linear inequalities] // Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya: Vychislitel'naya matematika i informatika [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2013. Vol. 2, No. 4. P. 103–108.

4. *Ananchenko I.V., Musaev A.A.* Torgovye roboty i upravlenie v khaoticheskikh sredakh: obzor i kriticheskiy analiz [Trading robots and management in chaotic environments: an overview and critical analysis] // Trudy SPIIRAN [SPIIRAS Proceedings]. 2014. No. 3(34). P. 178–203.

5. *Eremin I.I.* Fejerovskie metody dlya zadach linejnoj i vypukloj optimizatsii [Fejer's Methods for Problems of Convex and Linear Optimization]. Chelyabinsk, Publishing of the South Ural State University, 2009. 200 p.

6. *Ershova A.V., Sokolinskaya I.M.* O skhodimosti masshtabiruemogo algoritma postroeniya psevdoproektsii na vypukloe zamknutoe mnozhestvo [About convergence of scalable algorithm of constructing pseudo-projection on convex closed set] // Vestnik YuUrGU. Seriya "Matematicheskoe modelirovanie i programmirovanie"[Bulletin of South Ural State

University. Series: Mathematical simulation and programming]. 2011. No. 37(254), Issue 10. P. 12–21.

7. OpenMP Application Program Interface. Version 4.0 – July 2013. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf (Accessed: 28.06.2016).

8. *Supalov A., Semin A., Klemm M., Dahnken Ch.* Optimizing HPC Applications with Intel Cluster Tools. Apress, 2014. 269 p. DOI: 10.1007/978-1-4302-6497-2

9. *Thiagarajan S.U., Congdon C., Naik S., Nguyen L.Q.* Intel Xeon Phi coprocessor developer's quick start guide. White Paper. Intel, 2013. URL: https://software.intel.com/sites/default/files/managed/ee/4e/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf (Accessed: 28.06.2016).

10. *Kostenetskiy P.S., Safonov A.Y.* SUSU Supercomputer Resources // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576, CEURWS 2015. P. 561—573.