УДК 519.688; 004.272.2 doi 10.26089/NumMet.v19r448

МАСШТАБИРУЕМЫЙ АЛГОРИТМ ДЛЯ РЕШЕНИЯ НЕСТАЦИОНАРНЫХ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

И. М. Соколинская[1], **Л. Б. Соколинский**[2]

Статья посвящена исследованию алгоритма NSLP для решения нестационарных задач линейного программирования сверхбольшой размерности, ориентированного на кластерные вычислительные системы. В основе анализа лежит модель параллельных вычислений BSF, основанная на моделях BSP и SPMD. Даются краткие описания алгоритма NSLP и модели BSF. Рассматривается реализация алгоритма NSLP в виде BSF-программы. На базе стоимостной метрики модели BSF выводится верхняя граница масштабируемости алгоритма NSLP и оценивается эффективность его параллелизации. Описывается реализация алгоритма NSLP на основе программного каркаса BSF на языке Си и приводятся результаты экспериментов, исследующих масштабируемость указанной реализации на модельной задаче линейного программирования. Делается сравнение результатов, полученных аналитическим и экспериментальным путем.

Ключевые слова: нестационарная задача линейного программирования сверхбольшой размерности, алгоритм NSLP, модель параллельных вычислений BSF, оценка масштабируемости, кластерные вычислительные системы.

1. Введение. Одним из результатов развития технологий обработки и анализа больших данных стало появление математических моделей в виде задач линейного программирования (ЛП) сверхбольшой размерности [1]. Подобные задачи возникают в таких областях, как составление расписаний, логистика, реклама, ритейл, электронная торговля [2], квантовая физика [3], управление пассивами и активами [4], алгоритмическая торговля [5–8] и др. В подобных задачах количество переменных и неравенств в системе ограничений, формируемых с использованием больших данных, может составлять десятки миллионов. Во многих случаях, особенно это характерно для экономико-математического моделирования, указанные задачи ЛП имеют нестационарный характер, когда исходные данные (коэффициенты при переменных в системе ограничений, правые части неравенств, коэффициенты целевой функции) меняются в процессе решения задачи, при этом период изменения исходных данных может находиться в пределах сотых долей секунды.

До настоящего времени одним из самых распространенных способов решения задачи ЛП являлся класс алгоритмов, предложенных и разработанных Данцигом на основе симплекс-метода [9]. Симплекс-метод оказался эффективным для решения обширного класса задач ЛП. Однако в определенных случаях симплекс-методу приходится перебирать все вершины симплекса, что соответствует экспоненциальной временно́й сложности [10]. Кармаркар в работе [11] предложил алгоритм внутренних точек, который демонстрирует полиномиальное время решения задачи ЛП.

Симплекс-метод и метод внутренних точек на сегодня остаются основными методами решения задачи ЛП, однако эти методы могут оказаться неэффективными в случае сверхбольших задач ЛП с быстро меняющимися исходными данными. Для решения сверхбольших нестационарных задач линейного программирования авторами в работе [12] был предложен масштабируемый алгоритм *NSLP* (*Non-Stationary Linear Programming*), ориентированный на кластерные вычислительные системы. Алгоритм состоит из двух фаз: *Quest* (поиск) и *Targeting* (позиционирование). На фазе *Quest* происходит поиск решения системы неравенств, задающих систему ограничений задачи линейного программирования в условиях динамического изменения исходных данных. В качестве решения берется *псевдопроекция* [17] произвольной точки на n-мерный многогранник, определяемый системой ограничений задачи ЛП. Операция псевдопроектирования является обобщением операции проектирования и реализуется с помощью фейеровского (релаксационного) итерационного процесса [13–16]. Отличительной особенностью фейеровского процесса является его "самонаводящийся" характер: при изменении положения многогранника во время вычисления псевдопроекции фейеровский процесс автоматически корректирует направление своего движения.

[1] Южно-Уральский государственный университет, факультет вычислительной математики и информатики, просп. Ленина, 76, 454080, Челябинск; доцент, e-mail: irina.sokolinskaya@susu.ru

[2] Южно-Уральский государственный университет, просп. Ленина, 76, 454080, Челябинск; проректор по информатизации, e-mail: Leonid.Sokolinsky@susu.ru

Фаза *Quest* была исследована в работе [12], где была доказана теорема сходимости для случая, когда многогранник сдвигается путем параллельного переноса с постоянной скоростью. В работе [17] авторами было показано, что при вычислении псевдопроекций могут быть эффективно использованы многоядерные сопроцессоры Intel Xeon Phi.

На фазе *Targeting* формируется специальная система точек, имеющая форму n-мерного осесимметричного креста, которая передвигается в n-мерном пространстве таким образом, чтобы решение задачи линейного программирования постоянно находилось в ε-окрестности центральной точки креста. Фаза *Targeting* может быть эффективно реализована в виде параллельной программы для кластерной вычислительной системы на базе фреймворка "мастер–рабочие" [18–20]. В нашей работе выполняется аналитическое исследование масштабируемости параллельного алгоритма NSLP с использованием модели параллельных вычислений BSF, предложенной в [21].

Статья имеет следующую структуру. В разделе 2 формулируется постановка задачи и дается краткое описание алгоритма NSLP. Раздел 3 посвящен обзору модели параллельных вычислений BSF; приводятся итоговые формулы для оценки масштабируемости и эффективности распараллеливания BSF-программы. В разделе 4 описывается представление алгоритма NSLP в виде BSF-программы, вычисляется верхняя граница масштабируемости и оценивается эффективность распараллеливания в зависимости от процента исходных данных задачи, подвергаемых динамическим изменениям. В разделе 5 описывается реализация алгоритма NSLP на основе программного каркаса BSF на языке Си и приводится сравнение результатов, полученных аналитическим и экспериментальным путем. В разделе 6 суммируются полученные результаты и намечаются направления дальнейших исследований.

2. Алгоритм NSLP. Пусть в пространстве $\mathbb{R}^n$ задана нестационарная задача ЛП

$$\max\Big\{\langle c_t, x\rangle \Big| A_t x \leqslant b_t,\ x \geqslant 0\Big\}, \tag{1}$$

где матрица A_t имеет m строк. Нестационарность задачи заключается в том, что значения элементов матрицы A_t и векторов b_t, c_t зависят от момента времени $t \in \mathbb{R}_{\geqslant 0}$. При этом мы полагаем, что значение $t = 0$ соответствует начальному моменту времени:

$$A_0 = A, \quad b_0 = b, \quad c_0 = c.$$

Пусть M_t — многогранник, задаваемый ограничениями нестационарной задачи ЛП (1). Такой многогранник всегда является выпуклым.

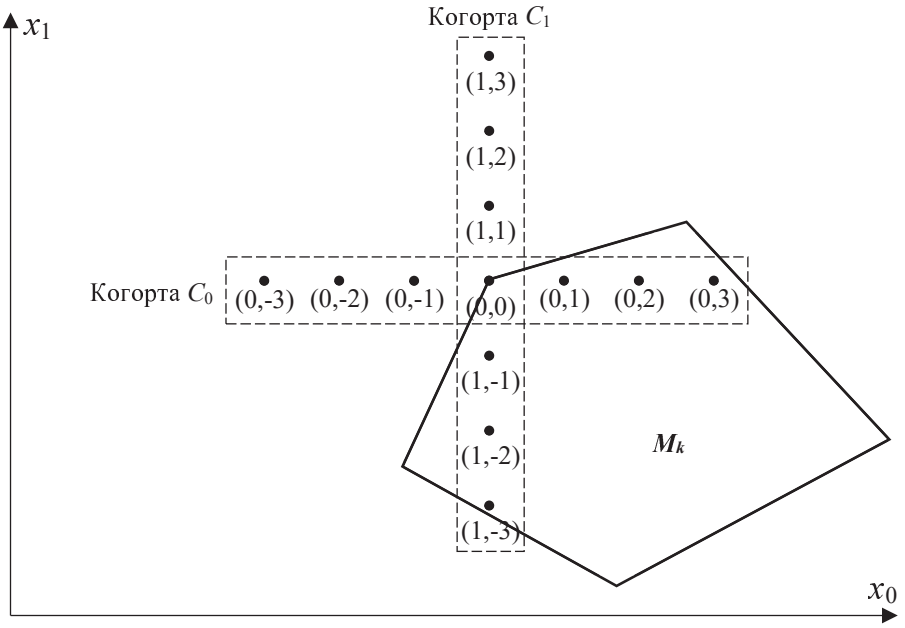

Рис. 1. Двумерный крест G: $K = 6$, $P = 12$

Фаза *Quest* вычисляет точку z, принадлежащую многограннику M_t. Этот процесс подробно описан в работе [12]. За фазой *Quest* следует фаза *Targeting*. На фазе *Targeting* формируется *осесимметричный n-мерный крест*, представляющий собой конечное множество точек

$$G = \{g_0, \ldots, g_{P-1}\} \subset \mathbb{R}^n,$$

имеющее мощность $P + 1$, где P кратно $n \geqslant 2$. Среди этих точек выделяется точка g_0, называемая *центральной*. В качестве начальных координат центральной точки выбираются координаты точки z, получаемой в результате выполнения фазы *Quest*. Множество $G\backslash\{g_0\}$ делится на n непересекающихся подмножеств C_i $(i = 0, \ldots, n-1)$, называемых *когортами*:

$$G\backslash\{g_0\} = \bigcup_{i=0}^{n-1} C_i.$$

Каждая i-я когорта ($i = 0, \ldots, n-1$) состоит из

$$K = P/n \tag{2}$$

точек, расположенных на одной прямой, параллельной i-й координатной оси и проходящей через центральную точку g_0. Сама центральная точка не входит ни в какую когорту. Расстояние между соседними точками множества $G \cup \{g_0\}$ является одинаковым и равным величине s, которая может меняться в ходе работы алгоритма. Пример двумерного креста приведен на рис. 1. Количество точек по одному измерению, не включая центральную точку, составляет K. Симметричность креста предполагает, что K принимает только четные значения, бóльшие либо равные 2. Из формулы (2) получается следующая формула, задающая общее количество точек в кресте:

$$P + 1 = nK + 1. \tag{3}$$

Поскольку K может принимать только четные значения, бóльшие либо равные 2, и $n \geqslant 2$, из формулы (3) следует, что P тоже может принимать только четные значения и $P \geqslant 4$. В примере на рис. 1: $n = 2$, $K = 6$, $P = 12$.

Каждая точка креста G однозначно идентифицируется *маркером* — парой целых чисел (χ, η), где $0 \leqslant \chi < n$ и $|\eta| \leqslant K/2$. С неформальной точки зрения χ задает номер когорты, а η задает порядковый номер точки в когорте C_χ относительно центральной точки. Соответствующая маркировка точек для двумерного случая приведена на рис. 1. Декартовы координаты точки $x_{(\chi,\eta)}$ с маркером (χ, η) восстанавливаются по формуле

$$x_{(\chi,\eta)} = g_0 + (0, \ldots, 0, \underbrace{\eta \cdot s}_{\chi}, 0, \ldots, 0). \tag{4}$$

Вектор, который прибавляется к g_0 в правой части формулы (4), имеет только одну ненулевую координату с номером χ, равную $\eta \cdot s$, где s — расстояние между соседними точками в когорте.

В самом общем виде работа алгоритма в фазе *Targeting* может быть описана следующей последовательностью действий.

1. Строится осесимметричный n-мерный крест G с количеством точек по одному измерению K (исключая центральную), с расстоянием между соседними точками s и центром в точке $g_0 = z_k$, где z_k — результат итерационного процесса, организованного на фазе *Quest*.
2. Вычисляется $G' = G \cap M_k$.
3. Вычисляются $C'_\chi = C_\chi \cap G'$ для $\chi = 0, \ldots, n-1$.
4. Вычисляется $Q = \bigcup\limits_{\chi=0}^{n-1} \left\{ \arg\max \left\{ \langle c_k, g \rangle | g \in C'_\chi, C'_\chi \neq \emptyset \right\} \right\}$.
5. Если $g_0 \in M_k$ и $\langle c_k, g_0 \rangle \geqslant \max\limits_{q \in Q} \langle c_k, q \rangle$, то $k := k + 1$ и перейти на шаг 2.
6. $g_0 := \dfrac{1}{|Q|} \sum\limits_{q \in Q} q$.
7. $k := k + 1$.
8. Перейти на шаг 2.

Таким образом, на фазе *Targeting* бесконечно выполняется итерационный процесс, описываемый шагами 2–7 алгоритма выше. С неформальной точки зрения на шаге 2 определяются точки креста G, принадлежащие многограннику M_k. На шаге 3 из каждой когорты отбрасываются точки, не принадлежащие многограннику M_k. На шаге 4 для оставшихся точек в каждой когорте находится точка с максимальным значением целевой функции. На шаге 5 проверяется: если значение целевой функции в центральной точке креста больше всех найденных на шаге 4 максимумов, то крест не сдвигается, счетчик времени t увеличивается на одну условную единицу и осуществляется переход к следующей итерации. На шаг 6 мы попадаем, если среди точек креста, принадлежащих M_k, найдены точки, в которых значение целевой функции превышает ее значение в центральной точке креста. В этом случае вычисляется новая центральная точка как центр масс точек, полученных на шаге 4. На шаге 7 счетчик времени t увеличивается

на одну условную единицу. На шаге 8 осуществляется переход к следующей итерации. Таким образом, центр g_0 креста G постоянно является приближенным решением нестационарной задачи (1).

3. Модель BSF. Для оценки верхних границ масштабируемости алгоритма NSLP в фазе *Targeting* мы используем модель параллельных вычислений BSF, предложенную в работе [21]. Модель BSF (*Bulk Synchronous Farm — блочно-синхронная ферма*) ориентирована на многопроцессорные системы с кластерной архитектурой и архитектурой MPP (Massively Parallel Processing). *BSF-компьютер* представляет собой множество однородных процессорных узлов с приватной памятью, соединенных сетью, позволяющей передавать данные от одного процессорного узла другому. Среди процессорных узлов выделяется один, называемый *узлом–мастером* (или кратко *мастером*). Остальные P узлов называются *узлами–рабочими* (или просто *рабочими*). В BSF-компьютере должен быть по крайней мере один узел-мастер и один рабочий ($P \geqslant 1$).

BSF-компьютер работает по схеме *SPMD* (*Single Program–Multiple Data*) [22], в соответствии с которой все процессорные узлы выполняют одну и ту же программу, но обрабатывают различные данные. Какие именно данные необходимо обрабатывать тому или иному процессорному узлу, определяется его уникальным номером, который является параметром программы.

BSF-программа состоит из последовательности *супершагов* и глобальных барьерных синхронизаций, выполняемых мастером и всеми рабочими. Каждый супершаг делится на секции двух типов: *секции мастера*, выполняемые только мастером, и *секции рабочего*, выполняемые только рабочими. Относительный порядок секций мастера и рабочего в рамках супершага не существенен. Данные, обрабатываемые конкретным узлом-рабочим, определяются его номером, являющимся параметром среды исполнения.

BSF-программа включает в себя следующие последовательные разделы:

— инициализация;

— итерационный процесс;

— завершение.

Инициализация представляет собой супершаг, в ходе которого мастер и рабочие считывают или генерируют исходные данные. Инициализация завершается барьерной синхронизацией. *Итерационный процесс* состоит в многократном повторении *тела итерационного процесса* до тех пор, пока не выполнено условие выхода, проверяемое мастером. В разделе *завершение* осуществляется вывод или сохранение результатов и завершение программы.

Тело итерационного процесса включает в себя следующие супершаги:

1) передача рабочим заданий от мастера;

2) выполнение задания (рабочими);

3) передача мастеру результатов от рабочих;

4) обработка полученных результатов мастером.

На первом супершаге мастер рассылает задания всем рабочим. На втором супершаге происходит выполнение полученного задания рабочими (мастер при этом простаивает). Все рабочие выполняют один и тот же программный код, но обрабатывают различные данные, адреса которых определяются по номеру рабочего. Это означает, что все рабочие тратят на вычисления одно и то же время. Никаких пересылок данных при выполнении задания не происходит. Это является важным свойством модели BSF. На третьем супершаге все рабочие пересылают мастеру полученные результаты, после чего происходит глобальная барьерная синхронизация. В ходе четвертого супершага мастер производит обработку и анализ полученных результатов. Рабочие в это время простаивают. Если условие выхода оказывается истинным, то происходит выход из итерационного процесса, в противном случае осуществляется переход на первый супершаг итерационного процесса. На четвертом супершаге происходит вывод или сохранение результатов и завершение работы мастера и рабочих.

Модель BSF позволяет получить аналитическую оценку масштабируемости BSF-программ и включает в себя следующие стоимостные параметры [21]:

P: количество узлов-рабочих;

L: латентность (время посылки сообщения длиной в 1 байт);

t_s: время, затрачиваемое мастером на передачу сообщения одному рабочему (без учета латентности);

t_w: время выполнения задания бригадой из одного рабочего в рамках одной итерации;

t_r: время, затрачиваемое мастером на получение результатов от всех рабочих (без учета латентности);

t_p: время, затрачиваемое мастером на обработку полученных результатов.

Верхняя граница масштабируемости BSF-программы определяется следующей формулой [21]:

$$P_{\max} = \sqrt{\frac{t_w}{2L + t_s}}\,. \tag{5}$$

Отметим, что верхняя граница масштабируемости BSF-программы не зависит от времени, затрачиваемого на передачу и обработку результатов, получаемых мастером от рабочих.

Ускорение для BSF-программы может быть посчитано по формуле [21]

$$a = \frac{P(2L + t_s + t_r + t_p + t_w)}{P^2(2L + t_s) + P(t_r + t_p) + t_w}\,. \tag{6}$$

Еще одной важной характеристикой параллельной программы является эффективность распараллеливания. Для BSF-программы эффективность распараллеливания может быть подсчитана по следующей приближенной формуле [21]:

$$e \approx \frac{1}{1 + \big(P^2(2L + t_s) + P(t_r + t_p)\big)\Big/t_w}\,. \tag{7}$$

4. Представление алгоритма NSLP в виде BSF-программы. В этом разделе мы покажем, как алгоритм, описанный в разделе 2, может быть представлен в виде BSF-программы. На основе такого представления мы оценим временну́ю стоимость итерации и дадим аналитическую оценку верхней границы масштабируемости алгоритма NSLP. При этом для расчетов мы будем использовать модельную масштабируемую задачу линейного программирования *Model-n* размерности n, приведенную в работе [17]. Матрица A в этой задаче имеет размеры $n \times 2(n+1)$, при этом мы предполагаем, что $n > 10^4$.

Алгоритм NSLP представляется в виде BSF-программы следующим образом. На супершаге "инициализация" мастер и все рабочие считывают (генерируют) и сохраняют в локальной памяти все исходные данные нестационарной задачи ЛП (1); мастер выполняет фазу *Quest* и находит точку z, принадлежащую многограннику M_t. Далее начинается *итерационный процесс*. В каждой итерации выполняются следующие шаги:

1) передача рабочим задания от мастера;

2) выполнение задания (рабочими);

3) передача мастеру результатов от рабочих;

4) обработка полученных результатов мастером.

Задание включает в себя информацию, приведенную в таблице. Предположим, что доля измененных элементов матрицы A, столбца b и коэффициентов c целевой функции равна $\delta(n)$, где для всех значений n выполнены неравенства $0 \leqslant \delta(n) \leqslant 1$. Тогда время t_s, необходимое для передачи задания для рабочего, в соответствии с данными из таблицы ниже может быть оценено следующим образом (без учета латентности):

$$\begin{aligned} t_s = t_\theta + t_\alpha + t_\beta + t_\gamma &= O(n) + O\big(\delta(n)\cdot n(n+1)\big) + O\big(\delta(n)(n+1)\big) + O\big(\delta(n)n\big) < \\ &< O(n+1) + O\big(\delta(n)\cdot(n+1)^2\big) + O\big(\delta(n)(n+1)\big) + O\big(\delta(n)(n+1)\big) = \\ &= O(n+1) + O\big(\delta(n)\cdot(n+1)^2\big) + O\big(\delta(n)(n+1)\big) < O\big(\delta(n)\cdot(n+1)^2\big) + O(n+1). \end{aligned}$$

Таким образом,

$$t_s < O\big(\delta(n)\cdot(n+1)^2\big) + O(n+1). \tag{8}$$

Наименьшей единицей распараллеливания в BSF-реализации алгоритма NSLP является когорта. Количество когорт совпадает с размерностью пространства n. Таким образом, количество рабочих узлов P не должно превышать размерность пространства n. Мы будем предполагать, что $n \gg P$. Рабочий последовательно обрабатывает все назначенные ему когорты. Для каждой точки x текущей когорты с помощью

формулы (4) вычисляются ее координаты. Трудоемкость этой операции составляет $O(n)$. Затем проверяется ее принадлежность многограннику M_t. Для этого рабочему достаточно проверить истинность условия $A_t x = b_t$. Так как A_t имеет размер $n \times 2(n+1)$, трудоемкость этой операции составляет $O\big(n^2 + n)\big)$. Количество точек в когорте, не считая центральной, равно константе K. В соответствии с формулой (3) общее количество точек в кресте равно nK. Следовательно, трудоемкость вычислений, выполняемых для всех точек креста на шагах 2–3, может быть оценена как $O\big(n^3 + n^2\big)$. После этого рабочие частично (для своих когорт) выполняют шаг 4 фазы *Targeting* (см. раздел 2). Общая трудоемкость этих действий составляет $O\big(n^2\big)$. Таким образом, суммарная трудоемкость всех вычислений, выполняемых рабочими, имеет следующую оценку:

$$t_w = O\big(n^3 + n^2\big) + O\big(n^2\big) + O(n) \leqslant O\big(n^3 + n^2 + n\big). \tag{9}$$

Структура сообщения “задание для рабочих”

№	Атрибут	Семантика	Затраты
1	θ	Координаты нового центра n-мерного креста	t_θ
2	α	Новые значения элементов матрицы A	t_α
3	β	Новые значения элементов столбца b	t_β
4	γ	Новые значения коэффициентов c целевой функции	t_γ

В качестве результата каждый рабочий возвращает мастеру суммарный вектор точек, принадлежащих многограннику и имеющих максимальное значение целевой функции в соответствующей когорте. Таким образом, общая трудоемкость передачи мастеру результатов от рабочих составляет

$$t_r = O(PKn) \approx O(n).$$

Получив результаты от рабочих, мастер выполняет их суммирование, завершая шаг 4 алгоритма. Трудоемкость этих вычислений будет иметь следующую оценку:

$$t_{\text{step 4}} \approx O\big(n^2\big).$$

В силу нестационарности задачи ЛП мы предполагаем, что условие, проверяемое на шаге 5, будет выполняться редко и после шага 4 будет, как правило, выполняться шаг 6. Так как количество когорт совпадает с n, общая трудоемкость шага 6 фазы *Targeting* составляет

$$t_{\text{step 6}} = O\big(n^2\big).$$

Таким образом, общая трудоемкость обработки мастером результатов, полученных от рабочих, составляет

$$t_p \approx t_{\text{step 4}} + t_{\text{step 6}} = O\big(n^2\big).$$

Подставляя значения из (9) и (8) в формулу (5), получаем следующую оценку верхней границы масштабируемости алгоритма NSLP:

$$P_{\text{maxNSLP}} = \sqrt{\frac{O\big(n^3 + n^2 + n\big)}{2L + O\big(\delta(n) \cdot (n+1)^2\big) + O(n+1)}}\,. \tag{10}$$

Предположим, что на каждой итерации меняются все исходные данные задачи, т.е. $\delta(n) = 1$. В этом случае формула (10) преобразуется к виду

$$P_{\text{maxNSLP}} = \sqrt{\frac{O\big(n^3 + n^2 + n\big)}{2L + O\big((n+1)^2\big) + O(n+1)}} \approx O\left(\sqrt{n}\right).$$

Это означает, что верхняя граница масштабируемости программы будет расти как корень квадратный от размерности задачи. Следовательно, реализация алгоритма NSLP в виде BSF-программы в этом случае является ограниченно масштабируемой.

Предположим теперь, что на каждой итерации доля измененных исходных данных задачи составляет

$$\delta(n) = \frac{1}{2(n+1)}\,. \tag{11}$$

Это соответствует ситуации, когда в матрице A меняется одна строка, в столбце b — один элемент, а в целевой функции — не более одного коэффициента. Подставляя значение $\delta(n)$ из формулы (11) в формулу (10), в этом случае получаем

$$P_{\text{maxNSLP}} = \sqrt{\frac{O\big(n^3+n^2+n\big)}{2L+O(n+1)+O(n+1)}} \approx O\left(\sqrt{n^2}\right) = O(n). \tag{12}$$

Это означает, что верхняя граница масштабируемости программы будет расти пропорционально размерности задачи. Следовательно, реализация алгоритма NSLP в виде BSF-программы в этом случае будет хорошо масштабируемой.

Используя формулу (6), мы можем оценить ускорение BSF-реализации алгоритма NSLP:

$$a = \frac{P\Big(2L+t_s+O(n)+O\big(n^2\big)+O\big(n^3+n^2+n\big)\Big)}{P^2(2L+t_s)+P\Big(O(n)+O\big(n^2\big)\Big)+O\big(n^3+n^2+n\big)}.$$

При $\delta(n) = \dfrac{1}{2(n+1)}$, $n \to \infty$ и $P \to \infty$, используя (8), отсюда получаем следующую оценку

$$a \approx \frac{O\big(n^3\big)P}{O(n)P^2+O\big(n^2\big)P+O\big(n^3\big)}.$$

Используя формулу (7), мы также можем оценить эффективность распараллеливания BSF реализации алгоритма NSLP:

$$e = \frac{1}{1+\Big(P^2\Big(2L+O\big(\delta(n)(n+1)^2\big)+O(n+1)\Big)+P\Big(O(n)+O\big(n^2\big)\Big)\Big)\Big/O\big(n^3+n^2\big)}. \tag{13}$$

При $\delta(n) = 1$, $n \to \infty$ и $P \to \infty$ отсюда получаем следующую оценку

$$e \approx \frac{1}{1+P^2\Big/O(n)}.$$

Это означает, что для высокой эффективности распараллеливания в случае $\delta(n) = 1$ необходимо, чтобы $n \gg P^2$. При $\delta(n) = \dfrac{1}{2(n+1)}$ из (13) получаем

$$e \approx \frac{1}{1+P^2\Big/O\big(n^2\big)+P\Big/O(n)}.$$

Это означает, что для высокой эффективности распараллеливания в случае $\delta(n) = \dfrac{1}{2(n+1)}$ необходимо, чтобы $n \gg P$.

5. Вычислительные эксперименты. Реализация фазы *Qwest* была описана и исследована нами в статье [17]. В рамках настоящей работы мы выполнили реализацию фазы *Targeting* на языке Си с использованием программного каркаса BSF. Данная реализация свободно доступна в сети Интернет по адресу https://github.com/leonid-sokolinsky/BSF-NSLP. С использованием модельной масштабируемой задачи линейного программирования *Model-n* размерности n, приведенной в работе [17], мы исследовали ускорение и эффективность распараллеливания фазы *Targeting* на суперкомпьютере "Торнадо ЮУрГУ" [23]. Тестирование проводилось для размерностей 400, 800 и 1080. Одновременно мы построили для этих же размерностей графики ускорения и эффективности с использованием формул (6) и (7). При этом мы предполагали, что $\delta(n) = 1\big/2(n+1)$.

Результаты приведены на рис. 2–4. Во всех случаях аналитические оценки оказались очень близки к экспериментальным. Кроме этого, проведенные эксперименты показывают, что верхняя граница масштабируемости программы растет пропорционально размерности задачи, что было предсказано аналитически с помощью формулы (12).

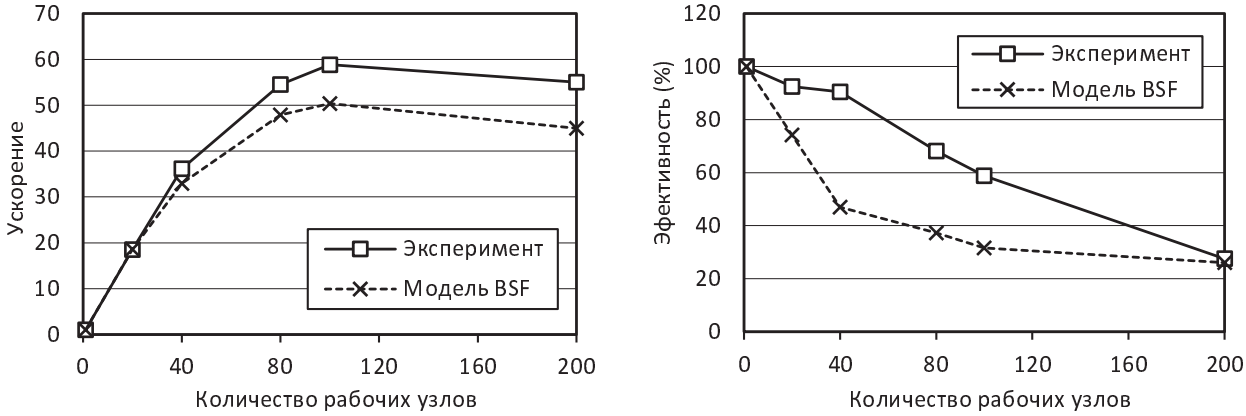

Рис. 2. Эксперименты при $n = 400$

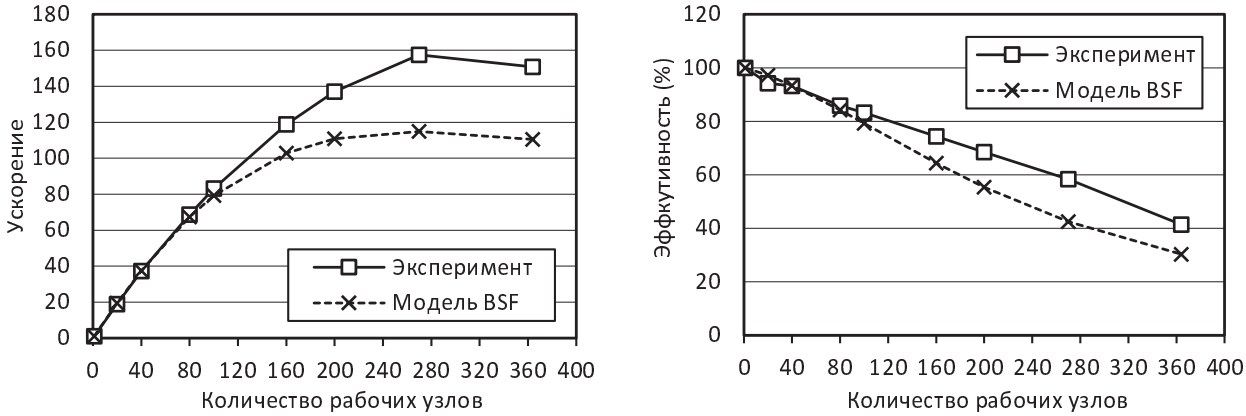

Рис. 3. Эксперименты при $n = 800$

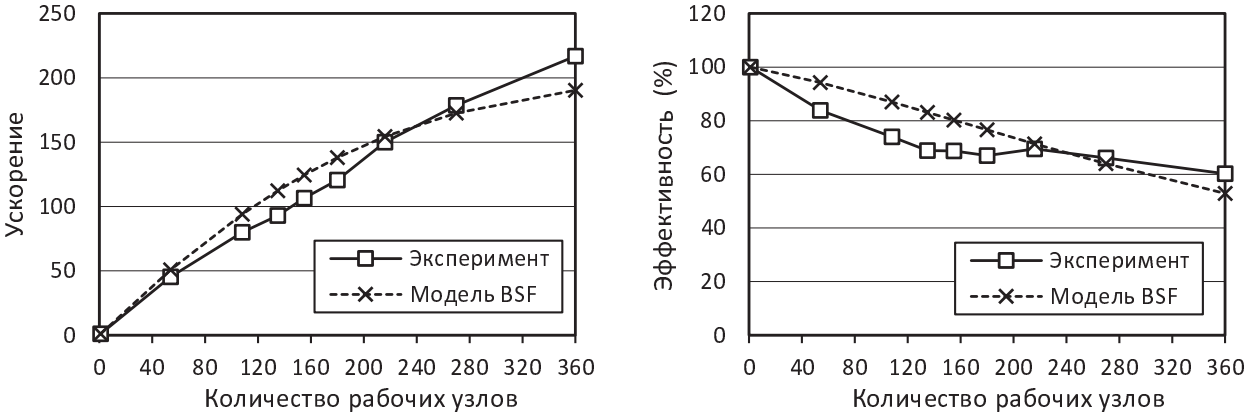

Рис. 4. Эксперименты при $n = 1080$

6. Заключение. В настоящей статье выполнено аналитическое исследование масштабируемости и эффективности распараллеливания алгоритма NSLP, применяемого для решения нестационарных задач линейного программирования большой размерности на кластерных вычислительных системах. Для этого использована модель параллельных вычислений BSF, основанная на парадигме “мастер–рабочие”. Описана BSF-реализация алгоритма NSLP. Получена оценка верхней границы масштабируемости BSF-реализации алгоритма NSLP. Эта оценка показывает, что если в ходе итерации динамически меняются все исходные данные задачи линейного программирования, то верхняя граница масштабируемости программы будет расти как корень квадратный от размерности задачи, т.е. реализация алгоритма NSLP в виде BSF-программы в этом случае будет ограниченно масштабируемой. Если же в каждом неравенстве системы ограничений меняется не более одного параметра, то верхняя граница масштабируемости про-

граммы будет расти пропорционально размерности задачи. Следовательно, реализация алгоритма NSLP в виде BSF-программы в этом случае будет хорошо масштабируемой. Кроме того, получены формулы для оценки эффективности распараллеливания BSF-реализации алгоритма NSLP. Указанные формулы позволяют сделать следующий вывод. Если в ходе итерации динамически меняются все исходные данные задачи, то для высокой эффективности распараллеливания необходимо, чтобы размерность задачи была намного больше, чем квадрат от количества рабочих: $n \gg P^2$. Если же в каждом неравенстве системы ограничений меняется не более одного параметра, то для высокой эффективности распараллеливания необходимо, чтобы размерность задачи была намного больше количества рабочих: $n \gg P$. Эксперименты, проведенные на модельной задаче, показали, что модель BSF с высокой точностью предсказывает верхнюю границу масштабируемости программы, реализующей фазу *Targeting* на основе программного каркаса BSF.

В рамках дальнейших исследований мы планируем решить следующие задачи:

1) параллельная реализация фазы *Qwest* на языке C++ с использованием BSF-каркаса и технологии параллельного программирования MPI;

2) проведение вычислительных экспериментов на кластерной вычислительной системе с использованием искусственных и реальных данных;

3) сравнение границ масштабируемости фазы *Qwest* и эффективности ее параллелизации, полученных экспериментально и аналитически.

Исследование выполнено при финансовой поддержке РФФИ (код проекта 17–07–00352а), Правительства РФ в соответствии с Постановлением № 211 от 16.03.2013 г. (соглашение № 02.A03.21.0011) и Министерства науки и высшего образования РФ (государственное задание 2.7905.2017/8.9).

СПИСОК ЛИТЕРАТУРЫ

1. *Chung W.* Applying large-scale linear programming in business analytics // Proceedings of the 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). New York: IEEE Press, 2015. 1860–1864.
2. *Tipi H.* Solving super-size problems with optimization // Presentation at the meeting of the 2010 INFORMS Conference on O.R. Practice. Orlando, Florida. April 2010. http://nymetro.chapter.informs.org/prac_cor_pubs/06-10%20Horia%20Tipi%20SolvingLargeScaleXpress.pdf (accessed 07.05.2017).
3. *Gondzio J. et al.* Solving large-scale optimization problems related to Bell's theorem // Journal of Computational and Applied Mathematics. 2014. **263**. 392–404.
4. *Sodhi M.S.* LP modeling for asset-liability management: a survey of choices and simplifications // Operations Research. 2005. **53**, N 2. 181–196.
5. *Brogaard J., Hendershott T., Riordan R.* High-frequency trading and price discovery // Review of Financial Studies. 2014. **27**, N 8. 2267–2306.
6. *Budish E., Cramton P., Shim J.* The high-frequency trading arms race: frequent batch auctions as a market design response // The Quarterly Journal of Economics. 2015. **130**, N 4. 1547–1621.
7. *Goldstein M.A., Kwan A., Philip R.* High-frequency trading strategies. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2973019.
8. *Hendershott T., Jones C.M., Menkveld A.J.* Does algorithmic trading improve liquidity? // The Journal of Finance. 2011. **66**, N 1. 1–33.
9. *Dantzig G.B.* Linear programming and extensions. Princeton: Princeton University Press, 1998.
10. *Klee V., Minty G.J.* How good is the simplex algorithm? // Inequalities. Vol. 3. New-York: Academic Press, 1972. 159–175.
11. *Karmarkar N.* A new polynomial-time algorithm for linear programming // Combinatorica. 1984. **4**, N 4. 373–395.
12. *Соколинская И.М., Соколинский Л.Б.* О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии — XI международная конференция, ПаВТ-2017, г. Казань, 3–7 апреля 2017 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2017. 471–484. http://omega.sp.susu.ru/pavt2017/short/014.pdf.
13. *Agmon S.* The relaxation method for linear inequalities // Canadian Journal of Mathematics. 1954. **6**. 382–392.
14. *Motzkin T.S., Schoenberg I.J.* The relaxation method for linear inequalities // Canadian Journal of Mathematics. 1954. **6**. 393–404.
15. *Еремин И.И.* Фейеровские методы для задач выпуклой и линейной оптимизации. Челябинск: Изд. центр ЮУрГУ, 2009.

16. *González-Gutiérrez E., Rebollar L.H., Todorov M.I.* Relaxation methods for solving linear inequality systems: converging results // TOP. 2012. **20**, N 2. 426–436.
17. *Соколинская И.М., Соколинский Л.Б.* Модифицированный следящий алгоритм для решения нестационарных задач линейного программирования на кластерных вычислительных системах с многоядерными ускорителями // Суперкомпьютерные дни в России: труды международной конференции (26–27 сентября 2016 г., г. Москва). М.: Изд-во МГУ, 2016. 294–306. http://2016.russianscdays.org/files/pdf16/294.pdf.
18. *Sahni S., Vairaktarakis G.* The master-slave paradigm in parallel computer and industrial settings // Journal of Global Optimization. 1996. **9**, N 3–4. 357–377.
19. *Silva L.M., Buyya R.* Parallel programming models and paradigms // High Performance Cluster Computing: Architectures and Systems. Vol. 2. Upper Saddle River: Prentice Hall, 1999. 4–27.
20. *Leung J.Y.-T., Zhao H.* Scheduling problems in master-slave model // Annals of Operations Research. 2008. **159**, N 1. 215–231.
21. *Ежова Н.А., Соколинский Л.Б.* BSF: модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Параллельные вычислительные технологии — XII международная конференция, ПаВТ-2018, г. Ростов-на-Дону, 2–6 апреля 2018 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2018. 253–265. http://omega.sp.susu.ru/pavt2018/short/001.pdf.
22. *Darema F., George D.A., Norton V.A., Pfister G.F.* A single-program–multiple-data computational model for EPEX/FORTRAN // Parallel Computing. 1988. **7**, N 1. 11–24.
23. *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. 2016. **1576**. 561–573.

Поступила в редакцию
31.10.2018

A Scalable Algorithm for Solving Non-Stationary Linear Programming Problems

I. M. Sokolinskaya [1] **and L. B. Sokolinsky** [2]

[1] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Ph.D., Associate Professor, e-mail: irina.sokolinskaya@susu.ru*

[2] *South Ural State University; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Dr. Sci., Professor, Vice-Rector for Informatization, e-mail: Leonid.Sokolinsky@susu.ru*

Received October 31, 2018

Abstract: This paper is devoted to the scalability study of an NSLP algorithm for solving non-stationary high-dimension linear programming problems on cluster computing systems. The analysis is based on the BSF model of parallel computations. The BSF model is a new parallel computation model designed on the basis of BSP and SPMD models. The brief descriptions of the NSLP algorithm and the BSF model are given. The NSLP algorithm implementation in the form of a BSF program is considered. On the basis of the BSF cost metric, the upper bound of the NSLP algorithm scalability is derived and its parallel efficiency is estimated. The NSLP algorithm implementation using BSF skeleton is described. The scalability estimates obtained analytically and experimentally are compared.

Keywords: non-stationary high-dimension linear programming problem, NSLP algorithm, BSF parallel computation model, scalability estimation, cluster computing systems.

References

1. W. Chung, "Applying Large-Scale Linear Programming in Business Analytics," in *Proc. IEEE Int. Conf. on Industrial Engineering and Engineering Management (IEEM), Singapore, December 6–9, 2015* (IEEE Press, New York, 2016), pp. 1860–1864.

2. H. Tipi, *Solving Super-Size Problems with Optimization*, Presentation at the Meeting of the 2010 INFORMS Conference on O.R. Practice. Orlando, Florida. April 2010. http://nymetro.chapter.informs.org/prac_cor_pubs/06-10%20Horia%20Tipi%20SolvingLargeScaleXpress.pdf. Cited December 10, 2018.

3. J. Gondzio, J. A. Gruca, J. A. J. Hall, et al., "Solving Large-Scale Optimization Problems Related to Bell's Theorem," J. Comput. Appl. Math. **263**, 392–404 (2014).

4. M. S. Sodhi, "LP Modeling for Asset-Liability Management: A Survey of Choices and Simplifications," Oper. Res. **53** (2), 181–196 (2005).

5. J. Brogaard, T. Hendershott, and R. Riordan, "High-Frequency Trading and Price Discovery," Rev. Financ. Stud. **27** (8), 2267–2306 (2014).

6. E. Budish, P. Cramton, and J. Shim, "The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response," Quart. J. Econ. **130** (4), 1547–1621 (2015).

7. M. A. Goldstein, A. Kwan, and R. Philip, "High-Frequency Trading Strategies," https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2973019. Cited December 10, 2018.

8. T. Hendershott, C. M. Jones, and A. J. Menkveld, "Does Algorithmic Trading Improve Liquidity?," J. Finance **66** (1), 1–33 (2011).

9. G. B. Dantzig, *Linear Programming and Extensions* (Princeton Univ. Press, Princeton, 1998).

10. V. Klee and G. J. Minty, "How Good is the Simplex Algorithm?," in *Inequalities* (Academic, New York, 1972), Vol. 3, pp. 159–175.

11. N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," Combinatorica **4** (4), 373–395 (1984).

12. I. M. Sokolinskaya and L. B. Sokolinsky, "On the Solution of Linear Programming Problem in the Era of Big Data," in *Proc. Int. Conf. on Parallel Computational Technologies, Kazan, Russia, April 3–7, 2017* (South Ural State Univ., Chelyabinsk, 2017), pp. 471–484.

13. S. Agmon, "The Relaxation Method for Linear Inequalities," Canad. J. Math. **6**, 382–392 (1954).

14. T. S. Motzkin and I. J. Schoenberg, "The Relaxation Method for Linear Inequalities," Canad. J. Math. **6**, 393–404 (1954).

15. I. I. Eremin, *Fejer Methods for Problems of Convex and Linear Optimization* (South Ural State Univ., Chelyabinsk, 2009) [in Russian].

16. E. González-Gutiérrez, L. H. Rebollar, and M. I. Todorov, "Relaxation Methods for Solving Linear Inequality Systems: Converging Results," TOP **20** (2), 426–436 (2012).

17. I. M. Sokolinskaya and L. B. Sokolinsky, "Revised Pursuit Algorithm for Solving Unstable Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators," in *Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia, September 26–27, 2016* (Mosk. Gos. Univ., Moscow, 2016), pp. 294–306.

18. S. Sahni and G. Vairaktarakis, "The Master–Slave Paradigm in Parallel Computer and Industrial Settings," J. Glob. Optim. **9** (3–4), 357–377 (1996).

19. L. M. Silva and R. Buyya, "Parallel Programming Models and Paradigms," in *High Performance Cluster Computing: Architectures and Systems* (Prentice Hall, Upper Saddle River, 1999), Vol. 2, pp. 4–27.

20. J. Y.-T. Leung and H. Zhao, "Scheduling Problems in Master–Slave Model," Ann. Oper. Res. **159** (1), 215–231 (2008).

21. N. A. Ezhova and L. B. Sokolinsky, "BSF: A model of Parallel Computation for Multiprocessor Systems with Distributed Memory," in *Proc. Int. Conf. on Parallel Computational Technologies, Rostov-on-Don, Russia, April 2–6, 2018* (South Ural State Univ., Chelyabinsk, 2018), pp. 253–265.

22. F. Darema, D. A. George, V. A. Norton, and G. F. Pfister, "A Single-Program–Multiple-Data Computational Model for EPEX/FORTRAN," Parallel Comput. **7** (1), 11–24 (1988).

23. P. S. Kostenetskiy and A. Y. Safonov, "SUSU Supercomputer Resources," in *Proc. 10th Annual Int. Scientific Conf. on Parallel Computing Technologies (PCT 2016), Arkhangelsk, Russia, March 29–31, 2016* CEUR Workshop Proc. **1576**, 561–573 (2016).