УДК 004.272.2 doi 10.26089/NumMet.v19r437

ИССЛЕДОВАНИЕ МАСШТАБИРУЕМОСТИ ИТЕРАЦИОННЫХ АЛГОРИТМОВ ПРИ СУПЕРКОМПЬЮТЕРНОМ МОДЕЛИРОВАНИИ ФИЗИЧЕСКИХ ПРОЦЕССОВ

Н. А. Ежова[1], **Л. Б. Соколинский**[2]

Статья посвящена разработке методики исследования масштабируемости ресурсоемких итерационных алгоритмов, применяемых в моделировании сложных физических процессов на суперкомпьютерных системах. В основе предлагаемой методики лежит модель параллельных вычислений BSF (Bulk Synchronous Farm), позволяющая на ранней стадии разработки итерационного алгоритма определить границу его масштабируемости. Модель BSF предполагает представление алгоритма в виде операций над списками с использованием функций высшего порядка. При этом рассматривается два класса представлений: BSF-M (Map BSF) и BSF-MR (Map-Reduce BSF). Предлагаемая методика описывается на примере решения систем линейных алгебраических уравнений методом Якоби. Для метода Якоби строится два итерационных алгоритма: Jacobi-M на основе представления BSF-M и Jacobi-MR на основе представления BSF-MR. Для указанных алгоритмов с помощью стоимостных метрик модели BSF даются аналитические оценки для ускорения, эффективности распараллеливания и верхней границы масштабируемости для многопроцессорных вычислительных систем с распределенной памятью. Приводится информация о реализации этих алгоритмов на языке C++ с использованием программного шаблона BSF и библиотеки параллельного программирования MPI. Демонстрируются результаты масштабных вычислительных экспериментов, выполненных на кластерной вычислительной системе. На основе экспериментальных результатов дается анализ адекватности оценок, полученных аналитическим путем с помощью стоимостных метрик модели BSF.

Ключевые слова: итерационный алгоритм, модель параллельных вычислений BSF, оценка масштабируемости, ускорение, эффективность распараллеливания, метод Якоби, кластерные вычислительные системы.

1. Введение. Концепция “Индустрия 4.0” рассматривает “умное” производство в виде предприятия, управляемого на основе интеллектуального анализа данных, собираемых с сенсорных датчиков, отслеживающих состояние оборудования, технологических линий и конечной продукции. Важной составляющей такого интеллектуального управления являются цифровые двойники (digital twins), представляющие собой компьютерную реализацию физической модели соответствующего устройства или процесса [1]. Как правило, физическая модель описывается системой дифференциальных уравнений, для которых не удается найти аналитическое решение. В этом случае применяются численные методы решения систем дифференциальных уравнений, зачастую сводящие решение исходной задачи к решению системы линейных алгебраических уравнений (СЛАУ). При моделировании технических устройств и процессов получающиеся системы уравнений обычно имеют высокую вычислительную сложность. Для решения таких систем в приемлемое время необходимо применять масштабируемые параллельные алгоритмы, ориентированные на многопроцессорные вычислительные системы с распределенной памятью. При создании параллельных алгоритмов для больших многопроцессорных систем важно уже на ранней стадии разработки алгоритма (до написания программы) получить аналитические оценки его масштабируемости. Для этой цели используются различные модели параллельных вычислений [2]. В настоящее время известно большое количество различных параллельных вычислительных моделей. Наиболее известными среди них являются модели PRAM (Parallel Random Access Machine) [3], BSP (Bulk Synchronous Parallel) [4] и LogP [5]. Указанные модели подверглись обобщениям и уточнениям, породив целые семейства, насчитывающие десятки параллельных вычислительных моделей (см., например, [6–8]). Задача разработки новых параллельных вычислительных моделей не утратила актуальности и в настоящее время. Это объясняется

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

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

тем, что невозможно создать модель параллельных вычислений — “хорошую во всех отношениях”. Поэтому необходимо ограничиваться определенными многопроцессорными архитектурами и определенными классами алгоритмов. В работах [9, 10] была предложена модель параллельных вычислений BSF (Bulk Synchronous Farm), ориентированная на кластерные вычислительные системы и алгоритмы итерационного типа. Модель BSF является расширением модели BSP, и основанная на методе программирования SPMD (Single Program Multiple Data) [11, 12] и парадигме “мастер–рабочие” [13]. Модель BSF позволяет с высокой точностью оценить верхнюю границу масштабируемости параллельного итерационного алгоритма до написания программы. Пример использования модели BSF для исследования параллельного алгоритма NSLP (Non-Stationary Linear Programming, алгоритм для решения нестационарных задач линейного программирования) можно найти в работе [14].

Целью настоящей статьи является исследование масштабируемости итерационных алгоритмов, применяемых в суперкомпьютерном моделировании физических процессов на многопроцессорных системах с распределенной памятью, путем использования модели параллельных вычислений BSF. В качестве таких алгоритмов выбраны две различные реализации метода Якоби для решения СЛАУ большой размерности. Статья организована следующим образом. В разделе 1 дается краткий обзор модели параллельных вычислений BSF. Раздел 2 посвящен вопросу представления итерационных численных алгоритмов в виде операций над списками с использованием функций высшего порядка Map и Reduce. Для таких представлений рассматриваются универсальные подходы для распараллеливания вычислений. В разделе 3 приводятся стоимостные метрики модели BSF, позволяющие получать аналитические оценки для ускорения, параллельной эффективности и верхней границы масштабируемости параллельного алгоритма. В разделе 4 дается формальное описание метода Якоби для решения системы линейных алгебраических уравнений. В разделе 5 описывается алгоритм Jacobi-M, реализующий метод Якоби в виде операций над списками с использованием функции высшего порядка Map. В разделе 6 с помощью стоимостных метрик модели BSF выводятся аналитические оценки для ускорения, параллельной эффективности и верхней границы масштабируемости алгоритма Jacobi-M. В разделе 7 описывается алгоритм Jacobi-MR, реализующий метод Якоби в виде операций над списками с использованием функций высшего порядка Map и Reduce. В разделе 8 с помощью стоимостных метрик модели BSF выводятся аналитические оценки для ускорения, параллельной эффективности и верхней границы масштабируемости алгоритма Jacobi-MR. В разделе 9 дается информация о реализациях алгоритмов Jacobi-M и Jacobi-MR, выполненных на языке C++ с использованием программных каркасов BSF-M, BSF-MR и библиотеки параллельного программирования MPI. Приводится сравнение результатов, полученных аналитическим и экспериментальным путями. В заключении суммируются полученные результаты и намечаются направления дальнейших исследований.

2. Модель параллельных вычислений BSF. Идея *модели параллельных вычислений BSF* (Bulk Synchronous Farm) впервые была предложена в работе [10]. В данном разделе мы кратко опишем усовершенствованный вариант этой модели. *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 K узлов называются *узлами-рабочими* (или просто *рабочими*). В BSF-компьютере должен быть по крайней мере один узел мастер и один рабочий ($K \geqslant 1$). BSF-компьютер работает по схеме SPMD. *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 существен. 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 всем рабочим одинаковые задания. 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 этого происходит глобальная барьерная синхронизация. В ходе четвертого супершага мастер производит обработку и анализ полученных результатов. Рабочие в это время простаивают. Если после обработки результатов 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 включает в себя следующие основные стоимостные параметры в рамках одной итерации:

K: количество узлов-рабочих;

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

t_w: время выполнения задания бригадой из одного рабочего;

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

t_p: время, затрачиваемое мастером на обработку полученных результатов и проверку условия завершения;

L: латентность (время пересылки сообщения длиной в 1 байт).

На основе этих параметров модель BSF позволяет получить аналитические оценки для ускорения, параллельной эффективности и верхней границы масштабируемости алгоритма. Однако конечный вид формул зависит от выбранного класса представления алгоритма в виде операций над списками с использованием функций высшего порядка. Модель BSF предусматривает два класса представлений: BSF-M и BSF-MR. В *представлении BSF-M* для имплементации алгоритма используется функция высшего порядка Map. В *представлении BSF-MR* кроме Map используется функция высшего порядка Reduce. Указанные функции высшего порядка кратко рассматриваются в следующем разделе.

3. Операции над списками. Для получения аналитических оценок алгоритма в соответствии с метриками модели BSF он должен быть представлен в виде операций над списками с использованием функций высшего порядка Map и Reduce, определяемых формализмом Бёрда–Миртенса (Bird–Meertens formalism) [15]. Для заданных функции $F : \mathbb{A} \to \mathbb{B}$ и списка $[a_1, \ldots, a_l]$ функция высшего порядка Map формирует новый список той же длины путем применения функции F ко всем элементам списка $[a_1, \ldots, a_l]$:

$$\mathrm{Map}\big(F, [a_1, \ldots, a_l]\big) = \big[F(a_1), \ldots, F(a_l)\big]. \tag{1}$$

Для заданных бинарной ассоциативной операции $\oplus : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ и списка $[b_1, \ldots, b_l]$ функция высшего порядка Reduce редуцирует список $[b_1, \ldots, b_l]$ к одному элементу путем многократного применения операции $\oplus$ к элементам списка:

$$\mathrm{Reduce}\big(\oplus, [b_1, \ldots, b_l]\big) = b_1 \oplus \ldots \oplus b_l. \tag{2}$$

Обозначим символом ++ операцию конкатенации двух списков. Из формулы (1) следует, что

$$\mathrm{Map}\big(F, [a_1, \ldots, a_l]\big) = \mathrm{Map}\big(F, [a_1, \ldots, a_m]\big) {+}{+} \ldots {+}{+} \mathrm{Map}\big(F, [a_{1+(k-1)m}, \ldots, a_l]\big)$$

для любых $m, k, l \in \mathbb{N}$, таких, что $l = k \cdot m$. Это означает, что выполнение функции Map может быть организовано как параллельное вычисление этой функции на k подсписках длины m, составляющих список $[a_1, \ldots, a_l]$, с последующей конкатенацией результирующих списков (см. алгоритм 1). Итерации цикла **pardo** могут выполняться параллельно, поскольку не зависят друг от друга по данным.

Из формул (1) и (2) следует:

$$\mathrm{Reduce}\Big(\oplus, \mathrm{Map}\big(F, [a_1, \ldots, a_l]\big)\Big) =$$
$$= \mathrm{Reduce}\left(\oplus, \left[\mathrm{Reduce}\Big(\oplus, \mathrm{Map}\big(F, [a_1, \ldots, a_m]\big)\Big), \ldots, \mathrm{Reduce}\Big(\oplus, \mathrm{Map}\big(F, [a_{(k-1)m+1}, \ldots, a_l]\big)\Big)\right]\right)$$

для любых $m, k, l \in \mathbb{N}$, таких, что $l = k \cdot m$. Это означает, что выполнение комбинации функций Map и Reduce может быть организовано как параллельное вычисление этой функции на k подсписках, составляющих список $[a_1, \ldots, a_l]$, с последующей конкатенацией результирующих списков (см. алгоритм 2). Здесь $e \in \mathbb{B}$ обозначает нейтральный элемент по отношению к операции $\oplus$: $b \oplus e = b$ для любого элемента $b \in \mathbb{B}$.

Алгоритм 1. Параллельное выполнение функции Map $\big($in $[a_1, \ldots, a_l]$, где $l = k \cdot m$; out $[b_1, \ldots, b_l]$, где $b_i = F(a_i)\big)$

```
1: for j from 1 to k
2:     [b_{1+m(j-1)}, ..., b_{j·m}] = Map(F, [a_{1+m(j-1)}, ..., a_{j·m}])
3: end for
```

Алгоритм 2. Параллельное выполнение функции Map и Reduce $\big($in $[a_1, \ldots, a_l]$, где $l = k \cdot m$; out $B = \mathrm{Reduce}\Big(\oplus, \mathrm{Map}\big(F, [a_1, \ldots, a_l]\big)\Big)$

```
1: for j from 1 to k pardo
2:     [b_{1+m(j-1)}, ..., b_{j·m}] = Map(F, [a_{1+m(j-1)}, ..., a_{j·m}])
3:     B_j = e
4:     for i from 1 + m(j − 1) to j · m do
5:         B_j = B_j ⊕ b_i
6:     end for
7: end for
8: B = e
9: for j from 1 to k
10:    B_j = B_j ⊕ b_i
11: end for
```

4. Стоимостные метрики модели BSF. Модель BSF предоставляет следующие метрики для оценки ускорения, параллельной эффективности и верхней границы масштабируемости алгоритма. Для представления BSF-M ускорение как функция от K вычисляется по формуле

$$a_{\mathrm{BSF-M}}(K) = \frac{K(2L + t_s + t_R + t_p + t_w)}{K^2(2L + t_s) + K(t_R + t_p) + t_w}\,. \tag{3}$$

Параллельная эффективность алгоритма как функция от K может быть оценена следующим образом:

$$e_{\mathrm{BSF-M}}(K) = \frac{2L + t_s + t_R + t_p + t_w}{K^2(2L + t_s) + K(t_R + t_p) + t_w}\,. \tag{4}$$

Верхняя граница масштабируемости алгоритма оценивается по следующей формуле:

$$P_{\mathrm{BSF-M}} = \sqrt{\frac{t_w}{2L + t_s}}\,. \tag{5}$$

Здесь $P_{\mathrm{BSF-M}}$ обозначает количество узлов-рабочих, при котором достигается максимум ускорения для алгоритма в представлении BSF-M.

Для представления BSF-MR вводятся три дополнительных параметра:

t_r: время, необходимое для передачи мастеру результата, полученного одним рабочим (без учета латентности);

t_a: время, необходимое для выполнения одной операции $\oplus$, являющейся параметром оператора Reduce;

l: длина списка Reduce.

Ускорение в этом случае вычисляется по формуле

$$a_{\mathrm{BSF-MR}}(K) = \frac{2L + t_s + t_r + t_p + t_w + lt_a}{K(2L + t_s + t_r + t_a) + (t_w + lt_a)/K - t_a + t_p}. \tag{6}$$

Параллельная эффективность может быть оценено следующим образом:

$$e_{\mathrm{BSF-MR}}(K) = \frac{2L + t_s + t_r + t_p + t_w + lt_a}{K^2(2L + t_s + t_r + t_a) + K(t_p - t_a) + t_w + lt_a}. \tag{7}$$

Верхняя граница масштабируемости алгоритма оценивается по следующей формуле:

$$P_{\mathrm{BSF-MR}} = \sqrt{\frac{t_w + lt_a}{2L + t_s + t_r + t_a}}. \tag{8}$$

5. Метод Якоби для СЛАУ. Итерационный метод Якоби [16] решения СЛАУ был впервые описан немецким математиком Карлом Густавом Якоби в работе [17]. Практическое применение метода Якоби стало возможным в связи с появлением ЭВМ [18]. Научный интерес к методу Якоби не утрачен до сих пор (см., например, [19, 20]). Дадим краткое описание метода Якоби. Пусть в евклидовом пространстве $\mathbb{R}^n$ задана совместная система линейных неравенств в матричном виде:

$$Ax = b, \tag{9}$$

где $A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$; $x = (x_1, \ldots, x_n)$; $b = (b_1, \ldots, b_n)$.

Предполагается, что $a_{ii} \neq 0$ для всех $i = 1, \ldots, n$. Определим матрицу

$$C = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{pmatrix}$$ следующим образом: $c_{ij} = \begin{cases} -\dfrac{a_{ij}}{a_{ii}} & \forall j \neq i; \\ 0 & \forall j = i. \end{cases}$

Определим вектор $d = (d_1, \ldots, d_n)$ следующим образом: $d_i = b_i/a_{ii}$. Метод Якоби нахождения приближенного решения системы (9) состоит из следующих шагов:

1. $k := 0$; $x^{(0)} := d$.
2. $x^{(k+1)} := Cx^{(k)} + d$.
3. Если $\left\|x^{(k+1)} - x^{(k)}\right\|^2 < \varepsilon$, перейти на шаг 5.
4. $k := k + 1$; перейти на шаг 2.
5. Стоп.

В качестве начального приближения $x^{(0)}$ в методе Якоби может быть взят произвольный вектор в $\mathbb{R}^n$. На шаге 1 в качестве начального приближения $X^{(0)}$ берется вектор d. На шаге 3 в критерии завершения используется евклидова норма $\|\cdot\|$. Достаточным условием сходимости метода Якоби является наличие у матрицы A диагонального преобладания

$$|a_{ii}| \geqslant \left(\sum_{j=1}^{n} |a_{ij}|\right) - |a_{ii}|$$

для всех $i = 1, \ldots, n$, причем хотя бы одно неравенство является строгим. Матрицы с диагональным преобладанием довольно часто возникают в приложениях. В этом случае система (9) имеет единственное решение при любых правых частях.

6. Алгоритм Jacobi-M. Алгоритм Jacobi-M реализует метод Якоби в виде операций над списками с использованием представления BSF-M. Определим L_{Map} как список номеров строк матрицы C, взятых в некотором порядке: $L_{\text{Map}} = [j_1, \ldots, j_n]$, где $j_k \in \{1, \ldots, n\}$ при $k = 1, \ldots, n$. Для произвольного $x \in \mathbb{R}^n$ определим функцию $F_x : \{1, \ldots, n\} \to \mathbb{R}^n$:

$$F_x(i) = d_i + \sum_{j=1}^{n} c_{ij} x_j \tag{10}$$

для всех $i \in \{1, \ldots, n\}$. Иначе говоря, функция $F_x(i)$ вычисляет i-ю координату следующего приближения. *Алгоритм Jacobi-M* состоит из следующих шагов:

1. $k := 0$; $x^{(0)} := d$; $L_{\text{Map}} := [1, \ldots, n]$.
2. $x^{(k+1)} := Map(F_{x^{(k)}}, L_{\text{Map}})$.
3. Если $\left\|x^{(k+1)} - x^{(k)}\right\|^2 < \varepsilon$, перейти на шаг 5.
4. $k := k + 1$; перейти на шаг 2.
5. Стоп.

Модель BSF предполагает, что алгоритм выполняется вычислительной системой, состоящей из одного узла-мастера и K узлов-рабочих ($K > 0$). Для простоты мы будем везде далее предполагать, что

$$n = mK \tag{11}$$

при некотором $m \in \mathbb{N}$. Исходные данные задачи (матрица A и вектор b) копируются на все узлы. Шаг 1 алгоритма Jacobi-M выполняется и мастером, и рабочими в ходе инициализации итерационного процесса. Шаг 2 (Map) выполняется только на узлах-рабочих. При этом распараллеливание осуществляется по схеме алгоритма 1. Шаги 3 и 4 выполняются только на узле-мастере.

7. Аналитическое исследование алгоритма Jacobi-M. В модели BSF предполагается, что все арифметические операции (сложение и умножение), а также операции сравнения двух чисел с плавающей точкой занимают одинаковое время, которое мы будем обозначать как τ_{op}. Введем следующие обозначения для анализа масштабируемости алгоритма Jacobi-M:

c_s: количество вещественных чисел, передаваемых от мастера одному рабочему;

c_{Map}: количество арифметических операций, выполняемых на шаге Map (шаг 2 алгоритма);

c_r: количество вещественных чисел, передаваемых от одного рабочего мастеру;

c_p: количество арифметических операций, выполняемых мастером на шаге 3 алгоритма.

Вычислим указанные значения. В начале итерации мастер передает каждому рабочему очередное приближение x_k, являющееся вектором размерности n. Следовательно,

$$c_s = n. \tag{12}$$

Подсчитаем количество арифметических операций, выполняемых на шаге Map. Для каждого элемента списка L_{Map} вычисляется один вектор по формуле (10), что составляет $2n$ операций с плавающей точкой. Умножив это число на количество строк в матрице C, получаем

$$c_{\text{Map}} = 2n^2. \tag{13}$$

В соответствии с (11) каждый узел-рабочий будет вычислять m координат следующего приближения, которые должны быть переданы узлу-мастеру. Значит,

$$c_r = m. \tag{14}$$

Выполнение шага 3 требует $(2n+2)$ операций. Отсюда получаем следующую формулу:

$$c_p = 2n + 2. \tag{15}$$

Положим τ_{op} — время, затрачиваемое рабочим на выполнение одной арифметической операции или операции сравнения, τ_{tr} — время, затрачиваемое на пересылку по сети одного вещественного числа без учета латентности. Тогда мы получаем следующие значения для стоимостных параметров модели BSF [9] в случае алгоритма Jacobi-M:

$$t_s = \tau_{\text{tr}} n; \tag{16}$$

$$t_w = 2\tau_{\text{op}} n^2; \tag{17}$$

$$t_R = \tau_{\text{tr}} n; \tag{18}$$

$$t_p = 2\tau_{\text{op}} (n+1). \tag{19}$$

Формула (16), полученная на основе (12), дает оценку времени t_s, затрачиваемого мастером на передачу сообщения одному рабочему без учета латентности. Формула (17), полученная с использованием формулы (13), позволяет оценить величину t_w суммарных временны́х затрат узлов-рабочих на вычисления над локальными данными. Для подсчета времени t_R, затрачиваемого мастером на получение результатов от всех рабочих (без учета латентности), необходимо c_r умножить на количество рабочих: $t_R = K \cdot c_r$. С учетом (14) и (11) получаем формулу (18). Формула (19), полученная на основе (15), вычисляет время t_p, затрачиваемое мастером на проверку условия завершения.

На основании формул (3)–(5) и (16)–(19) мы получаем следующие формулы для ускорения, параллельной эффективности и верхней границы масштабируемости алгоритма Jacobi-M:

$$a_{\text{Jacobi}-\text{M}}(K) = \frac{K\big(2L + \tau_{\text{tr}} n + \tau_{\text{tr}} n + \tau_{\text{op}} \cdot (2n+2) + \tau_{\text{op}} 2n^2\big)}{K^2(2L + \tau_{\text{tr}} n) + K\big(\tau_{\text{tr}} n + \tau_{\text{op}} \cdot (2n+2)\big) + \tau_{\text{op}} 2n^2}, \tag{20}$$

$$e_{\text{Jacobi}-\text{M}}(K) = \frac{2L + \tau_{\text{tr}} n + \tau_{\text{tr}} n + \tau_{\text{op}} \cdot (2n+2) + \tau_{\text{op}} 2n^2}{K^2(2L + \tau_{\text{tr}} n) + K\big(\tau_{\text{tr}} n + \tau_{\text{op}} \cdot (2n+2)\big) + \tau_{\text{op}} 2n^2}, \tag{21}$$

$$P_{\text{Jacobi}-\text{M}} = \sqrt{\frac{\tau_{\text{op}} \cdot 2n^2}{2L + \tau_{\text{tr}} n}}. \tag{22}$$

Упростим формулу (22). При $n \to \infty$ имеем

$$2L + \tau_{\text{tr}} n = O(n). \tag{23}$$

Подставив правую часть уравнения (23) в (22), получим

$$P_{\text{Jacobi}-\text{M}} = \sqrt{\frac{O\big(n^2\big)}{O(n)}},$$

что равносильно $P_{\text{Jacobi}-\text{M}} = \sqrt{O(n)}$.

Таким образом, верхняя граница $P_{\text{Jacobi}-\text{M}}$ масштабируемости алгоритма Jacobi-M растет пропорционально корню квадратному из размерности задачи.

8. Алгоритм Jacobi-MR. Алгоритм Jacobi-MR реализует метод Якоби в виде операций над списками с использованием функций высшего порядка Map и Reduce. Определим L_{Map} как список номеров столбцов матрицы C, взятых в некотором порядке:

$$L_{\text{Map}} = [i_1, \ldots, i_n],$$

где $i_k \in \{1, \ldots, n\}$ при $k = 1, \ldots, n$. Для произвольного $x \in \mathbb{R}^n$ определим функцию $F_x : \{1, \ldots, n\} \to \mathbb{R}^n$:

$$F_x(j) = (x_j c_{1j}, \ldots, x_j c_{nj})$$

для всех $j \in \{1, \ldots, n\}$. С неформальной точки зрения функция $F_x(j)$ умножает j-й столбец матрицы C на j-ю координату вектора x. Для произвольного $x \in \mathbb{R}^n$ определим список $L^{(x)}_{\text{Reduce}} \subset \mathbb{R}^n$ следующим образом:

$$L^{(x)}_{\text{Reduce}} = \big[F_x(j_1), \ldots, F_x(j_n)\big].$$

Список $L^{(x)}_{\text{Reduce}}$ включает в себя столбцы матрицы C, умноженные на соответствующую координату вектора x, взятые в порядке, определяемом списком L_{Map}. Таким образом, список $L^{(x)}_{\text{Reduce}}$ получается из списка L_{Map} путем применения к нему функции высшего порядка Map, использующей в качестве параметра функцию F_x:

$$L^{(x)}_{\text{Reduce}} = \text{Map}(F_x, L_{\text{Map}}).$$

Определим бинарную ассоциативную операцию $\oplus : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ следующим образом:

$$x \oplus y = x + y$$

для любых $x, y \in \mathbb{R}^n$. В данном случае операция $\oplus$ выполняет обычное сложение векторов. Тогда для текущего приближения $x^{(k)}$ следующее приближение $x^{(k+1)}$ может быть получено путем применения к списку $L^{(x^{(k)})}_{\text{Reduce}}$ функции высшего порядка Reduce, использующей в качестве параметра операцию сложения векторов $\oplus$, с последующим сложением с вектором d:

$$x^{(k+1)} = \text{Reduce}\left(\oplus, L^{(x^{(k)})}_{\text{Reduce}}\right) + d.$$

Алгоритм Jacobi-MR состоит из следующих шагов:

1. $k := 0$; $x^{(0)} := d$; $L_{\text{Map}} := [1, \ldots, n]$.
2. $L^{(x^{(k)})}_{\text{Reduce}} := \text{Map}(F_{x^{(k)}}, L_{\text{Map}})$.
3. $x^{(k+1)} := \text{Reduce}\left(\oplus, L^{(x^{(k)})}_{\text{Reduce}}\right)$.
4. $x^{(k+1)} := x^{(k+1)} + d$.
5. Если $\left\|x^{k+1} - x^k\right\|^2 < \varepsilon$, то перейти на шаг 7.
6. $k := k + 1$; перейти на шаг 2.
7. Стоп.

Исходные данные задачи (матрица A и вектор b) копируются на все узлы. Шаг 1 алгоритма Jacobi-MR выполняется и мастером, и рабочими в ходе инициализации итерационного процесса. Шаг 2 (Map) выполняется только на узлах-рабочих. При этом распараллеливание осуществляется по схеме алгоритма 1. Шаг 3 (Reduce) выполняется на узлах-рабочих и частично на узле-мастере. При этом распараллеливание осуществляется по схеме алгоритма 2. Шаги 4–6 выполняются только на узле-мастере.

9. Аналитическое исследование алгоритма Jacobi-MR. В рамках одной итерации введем следующие обозначения для анализа масштабируемости алгоритма Jacobi-MR:

c_s: количество вещественных чисел, передаваемых от мастера одному рабочему;

c_{Map}: количество арифметических операций, выполняемых на шаге Map (шаг 2 алгоритма);

c_r: количество вещественных чисел, передаваемых от одного рабочего мастеру;

c_p: количество арифметических операций, выполняемых мастером на шагах 4 и 5 алгоритма.

Вычислим указанные значения. В начале каждой итерации мастер передает каждому рабочему очередное приближение c_r, являющееся вектором размерности n. Следовательно, получаем

$$c_r \tag{24}$$

На шаге Map происходит умножение столбцов матрицы C с номерами из списка L_{Map} на соответствующую координату вектора $x^{(k)}$. Для этого требуется n^2 арифметических операций. Следовательно,

$$c_{\text{Map}} = n^2. \tag{25}$$

Для сложения двух векторов размерности n требуется n арифметических операций. Следовательно,

$$c_a = n. \tag{26}$$

Вектор, полученный каждым рабочим на шаге Reduce, пересылается мастеру. Значит,

$$c_r = n. \tag{27}$$

Выполнение шага 4 требует n арифметических операций. Выполнение шага 5 требует $3n-1$ арифметических операций и одну операцию сравнения. Отсюда получаем следующую формулу:

$$c_p = 3n. \tag{28}$$

Мы по-прежнему полагаем, что τ_{op} обозначает время, затрачиваемое рабочим на выполнение одной арифметической операции, а τ_{tr} — время, затрачиваемое на пересылку по сети одного вещественного числа без учета латентности. Тогда мы получаем следующие значения для стоимостных параметров алгоритма Jacobi-MR:

$$t_s = \tau_{\text{tr}} n; \tag{29}$$

$$t_w = c_{\text{Map}} + (m-1)c_a = \tau_{\text{op}}\big(n^2 + n(n-K)\big); \tag{30}$$

$$t_r = \tau_{\text{tr}} n; \tag{31}$$

$$t_a = \tau_{\text{op}} n; \tag{32}$$

$$t_p = 3\tau_{\text{op}} n. \tag{33}$$

Формула (29), полученная на основе (24), дает оценку времени t_s, затрачиваемого мастером на передачу сообщения одному рабочему без учета латентности. Формула (30) получена с использованием формул (11), (25) и (26). Действительно, в соответствии со стоимостной метрикой модели BSF t_w обозначает суммарное время рабочих, затрачиваемые на вычисления над локальными данными. В результате распараллеливания шага 3 (Reduce) алгоритма Jacobi-MR между K рабочими в соответствии с методикой, описанной в разделе 2 (см. алгоритм 2), на каждого рабочего приходится подсписок длиной m. В соответствии с этим каждый рабочий должен произвести $(m-1)$ сложений векторов размерности n, что требует $c_a(m-1)$ арифметических операций. Следовательно, суммарное количество операций, выполняемых K рабочими на шаге Reduce, с учетом формулы (11) составит $n(n-K)$. Добавив сюда правую часть формулы (25) и умножив все на τ_{op}, получаем итоговую формулу (30) для t_w. Формула (31), полученная на основе (27), дает оценку времени t_r, затрачиваемого мастером на получение сообщения от одного рабочего без учета латентности. Формула (32), полученная на основе (26), вычисляет время t_a, затрачиваемое мастером на выполнение своей части шага Reduce. И наконец, формула (33), полученная на основе (28), вычисляет время t_p, затрачиваемое мастером на вычисление очередного приближения и проверку условия завершения.

Подставляя в формулу (6) значения правых частей из формул (29)–(33) получаем следующую формулу для оценки ускорения алгоритма Jacobi-MR:

$$a_{\text{Jacobi-MR}}(K) = \frac{2(L+\tau_{\text{tr}} n) + \tau_{\text{op}} n(3n-K+3)}{K\big(2(L+\tau_{\text{tr}} n) + \tau_{\text{op}} n\big) + 3\tau_{\text{op}} n(n/K+1)}. \tag{34}$$

Формула (7) дает оценку параллельной эффективности алгоритма Jacobi-MR:

$$e_{\text{Jacobi-MR}}(K) = \frac{2(L+\tau_{\text{tr}} n) + \tau_{\text{op}} n(3n-K+3)}{K^2\big(2(L+\tau_{\text{tr}} n) + \tau_{\text{op}} n\big) + 3\tau_{\text{op}} n(n+K)}. \tag{35}$$

Верхняя граница масштабируемости алгоритма Jacobi-MR получается путем подстановки значений правых частей из формул (29)–(33) в формулу (8):

$$P_{\text{Jacobi-MR}} = \sqrt{\frac{\tau_{\text{op}} n(3n-K)}{2(L+\tau_{\text{tr}} n) + \tau_{\text{op}} n}}. \tag{36}$$

Упростим формулу (36). При $n \to \infty$ имеем

$$2(L+\tau_{\text{tr}} n) + \tau_{\text{op}} n = O(n). \tag{37}$$

Из (11) следует, что $K \leqslant n$. Тогда для $n \to \infty$ имеем

$$\tau_{\text{op}} n(3n-K) = O\big(n^2\big). \tag{38}$$

Подставив правые части уравнений (37) и (38) в (36), получим

$$P_{\text{Jacobi}-\text{MR}} = \sqrt{\frac{O(n^2)}{O(n)}},$$

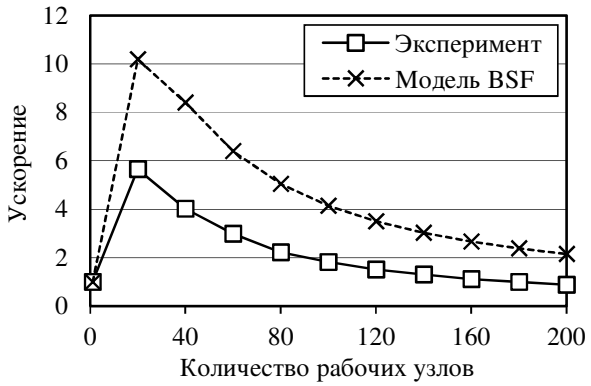
что равносильно $P_{\text{Jacobi}-\text{MR}} = \sqrt{O(n)}$. Таким образом, верхняя граница масштабируемости алгоритма Jacobi-MR над списками тоже растет пропорционально корню квадратному из размерности задачи n.

10. Вычислительные эксперименты. Для верификации результатов, полученных аналитическим путем, мы выполнили реализацию алгоритмов Jacobi-M и Jacobi-MR на языке C++ с использованием программного каркаса BSF и библиотеки параллельного программирования MPI. Эти реализации свободно доступны в сети Интернет по адресам

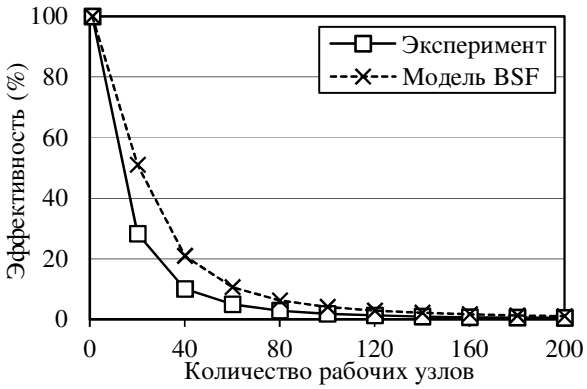
https://github.com/nadezhda-ezhova/Jacobi-M и https://github.com/nadezhda-ezhova/Jacobi-MR.

Для проведения экспериментов была использована масштабируемая система линейных уравнений (9), имеющая следующие матрицу коэффициентов A и столбец свободных членов b:

$$A = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & n \end{pmatrix}, \quad b = \begin{pmatrix} n \\ n+1 \\ \vdots \\ 2n-1 \end{pmatrix}$$

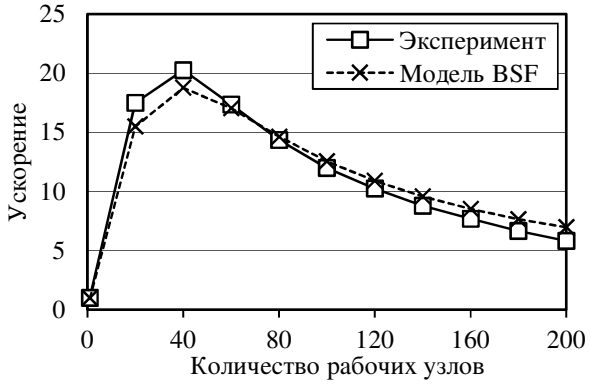


а) Ускорение

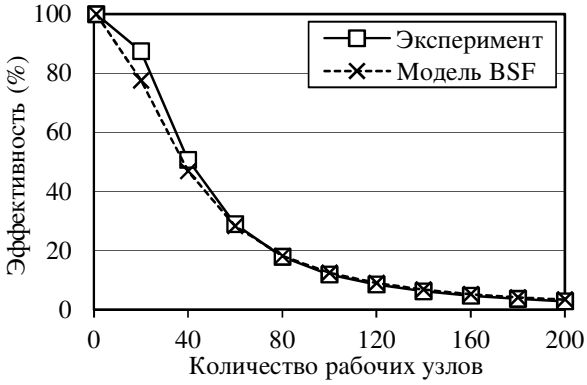


б) Параллельная эффективность

Рис. 1. Jacobi-M для $n = 1500$

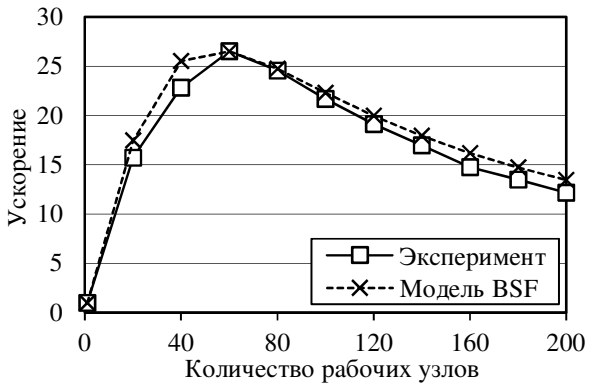


а) Ускорение

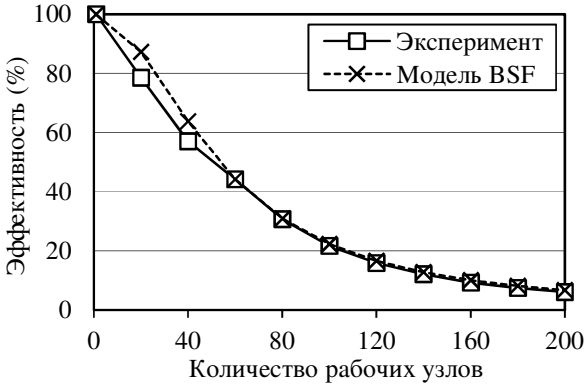


б) Параллельная эффективность

Рис. 2. Jacobi-M для $n = 5000$

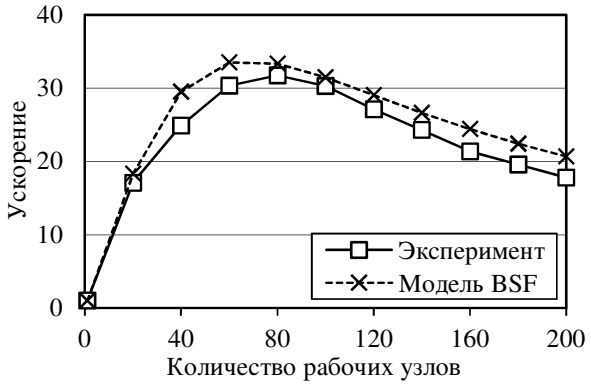


а) Ускорение

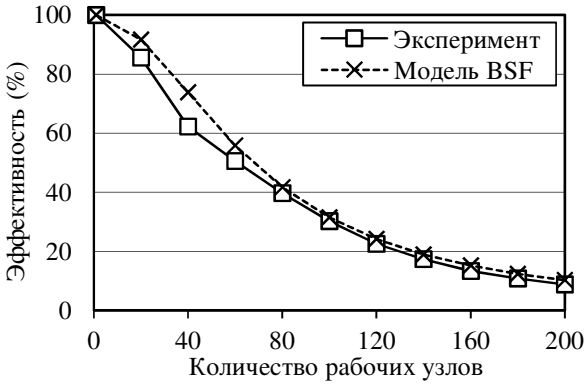


б) Параллельная эффективность

Рис. 3. Jacobi-M для $n = 10000$

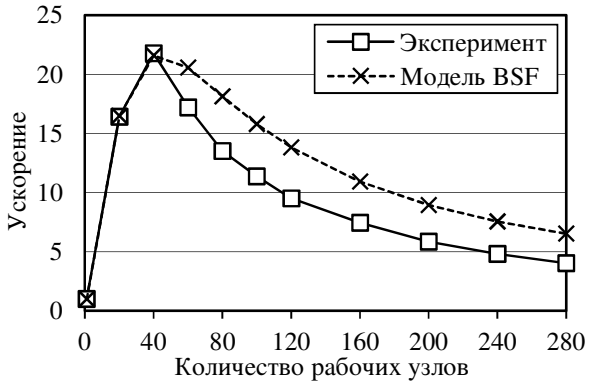


а) Ускорение

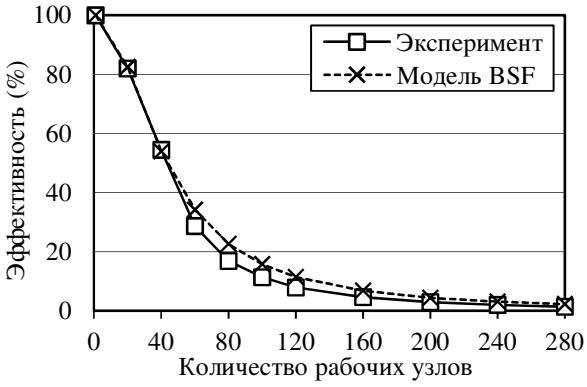


б) Параллельная эффективность

Рис. 4. Jacobi-M для $n = 16000$



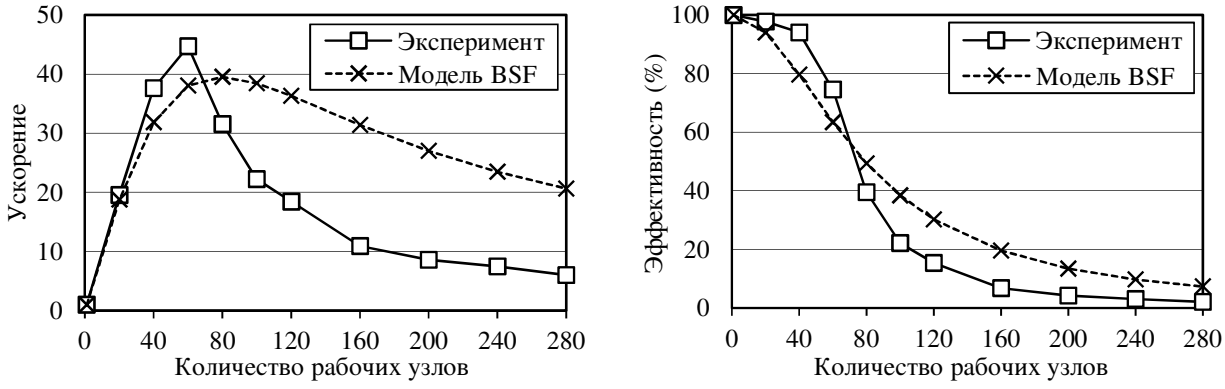
а) Ускорение



б) Параллельная эффективность

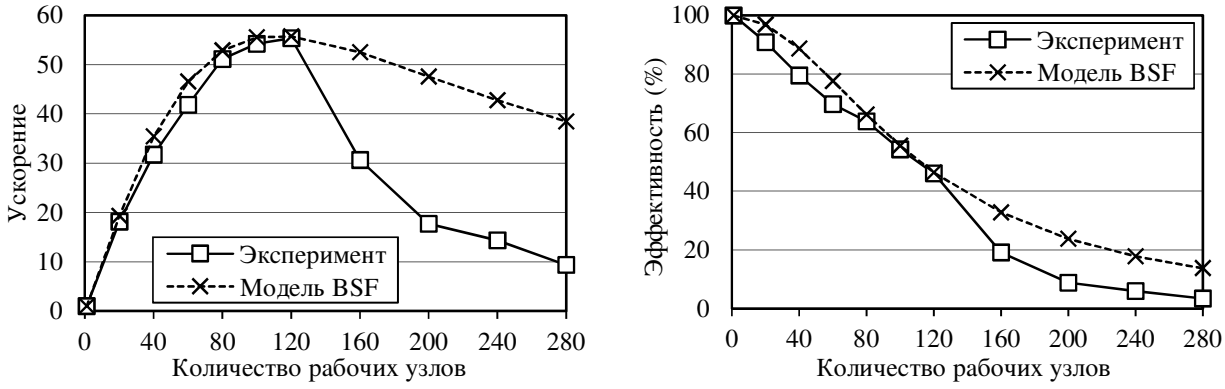
Рис. 5. Jacobi-MR для $n = 1500$

Мы исследовали ускорение и эффективность распараллеливания алгоритмов Jacobi-M и Jacobi-MR на суперкомпьютере “Торнадо ЮУрГУ” [21]. Тестирование проводилось для размерностей 1500, 5000, 10000 и 16000. Одновременно мы построили для этих же размерностей графики ускорения и эффективности обоих алгоритмов с использованием формул (20) и (21), (34) и (35). Для этого экспериментальным путем



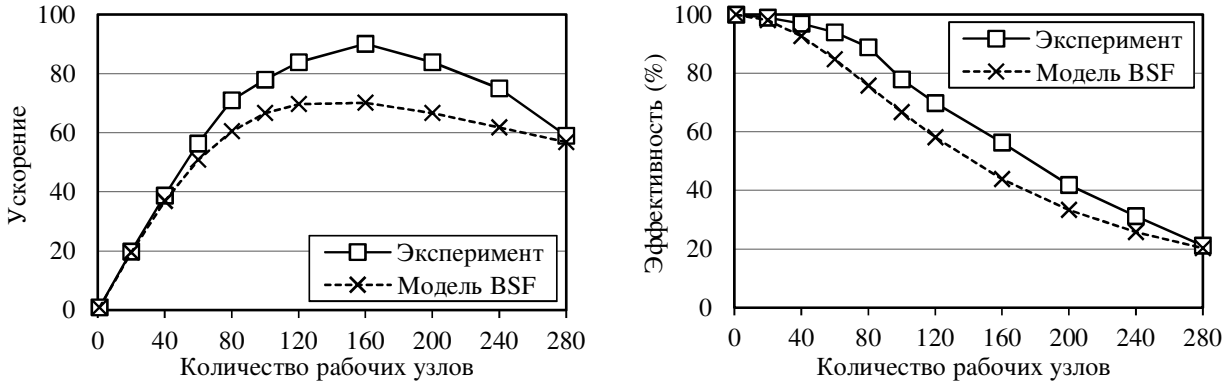
а) Ускорение б) Параллельная эффективность

Рис. 6. Jacobi-MR для $n = 5000$



а) Ускорение б) Параллельная эффективность

Рис. 7. Jacobi-MR для $n = 10000$



а) Ускорение б) Параллельная эффективность

Рис. 8. Jacobi-MR для $n = 16000$

были определены величины в секундах: $L = 1.5 \times 10^{-5}$, $\tau_{\text{op}} = 2.9 \times 10^{-8}$ и $\tau_{\text{tr}} = 1.9 \times 10^{-7}$.

Результаты приведены на рис. 1–8. Во всех случаях аналитические оценки оказались очень близки к экспериментальным. Кроме того, проведенные эксперименты показывают, что верхняя граница масштабируемости алгоритмов растет пропорционально корню квадратному из размерности задачи, что было предсказано аналитически с помощью формул (22) и (36).

11. Заключение. В настоящей статье выполнено аналитическое исследование масштабируемости и эффективности распараллеливания алгоритмов Jacobi-M и Jacobi-MR для решения больших систем линейных уравнений на многопроцессорных системах с распределенной памятью. Для этого использована модель параллельных вычислений BSF (Bulk Synchronous Farm), основанная на парадигме "мастер–рабочие". Описана реализация алгоритмов Jacobi-M и Jacobi-MR в виде операций над списками с использованием функций высшего порядка Map и Reduce, определяемых формализмом Бёрда-Миртенса. Получена оценка верхней границы масштабируемости алгоритмов Jacobi-M и Jacobi-MR над списками. Кроме того, получены формулы для оценки ускорения и эффективности распараллеливания алгоритмов Jacobi-M и Jacobi-MR. Выполнена реализация на языке C++ с использованием программного каркаса BSF и библиотеки параллельного программирования MPI. На кластерной вычислительной системе проведены масштабные эксперименты для определения фактических кривых ускорения и параллельной эффективности для задач с количеством уравнений 1500, 5000, 10000, 16000. Результаты экспериментов показали, что модель BSF с высокой точностью предсказывает верхнюю границу масштабируемости алгоритмов Jacobi-M и Jacobi-MR над списками. В рамках дальнейших исследований планируется разработать и реализовать диалоговую среду для автоматизированного создания и верификации BSF-программ.

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

СПИСОК ЛИТЕРАТУРЫ

1. *Borodulin K. et al.* Towards digital twins cloud platform // Proceedings of the 10th International Conference on Utility and Cloud Computing. New York: ACM Press, 2017. 209–210.
2. *Bilardi G., Pietracaprina A.* Models of computation, theoretical // Encyclopedia of Parallel Computing. Boston: Springer, 2011. 1150–1158.
3. *JaJa J.F.* PRAM (Parallel Random Access Machines) // Encyclopedia of Parallel Computing. Boston: Springer, 2011. 1608–1615.
4. *Valiant L.G.* A bridging model for parallel computation // Communications of the ACM. 1990. **33**, № 8. 103–111.
5. *Culler D. et al.* LogP: towards a realistic model of parallel computation // Proc. of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 1993. DOI: 10.1145/155332.155333.
6. *Forsell M., Leppänen V.* An extended PRAM-NUMA model of computation for TCF programming // Proc. of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops. Washington, DC: IEEE Press, 2012. 786–793. DOI: 10.1109/IPDPSW.2012.97.
7. *Gerbessiotis A.V.* Extending the BSP model for multi-core and out-of-core computing: MBSP // Parallel Computing. 2015. **41**. 90–102. DOI: 10.1016/j.parco.2014.12.002.
8. *Lu F., Song J., Pang Y.* HLognGP: A parallel computation model for GPU clusters // Concurrency and Computation: Practice and Experience. 2015. **27**, N 17. 4880–4896. DOI: 10.1002/cpe.3475.
9. *Ежова Н.А., Соколинский Л.Б.* Модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. **7**, № 2. 32–49.
10. *Sokolinsky L.B.* Analytical estimation of the scalability of iterative numerical algorithms on distributed memory multiprocessors // Lobachevskii Journal of Mathematics. 2018. **39**, № 4. 571–575.
11. *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.
12. *Darema F.* SPMD computational model // Encyclopedia of Parallel Computing. Boston: Springer, 2011. 1933–1943.
13. *Sahni S., Vairaktarakis G.* The master-slave paradigm in parallel computer and industrial settings // Journal of Global Optimization. 1996. **9**, № 3–4. 357–377.
14. *Sokolinskaya I., Sokolinsky L.B.* Scalability evaluation of the NSLP algorithm for solving non-stationary linear programming problems on cluster computing systems // Тр. Международной научной конференции "Суперкомпьютерные дни в России". М.: Изд-во МГУ, 2017. 319–332. http://russianscdays.org/files/pdf17/319.pdf.
15. *Cole M.I.* Parallel programming with list homomorphisms // Parallel Processing Letters. 1995. **5**, N 2. 191–203. DOI: 10.1142/S0129626495000175.
16. *Rutishauser H.* The Jacobi method for real symmetric matrices // Handbook for Automatic Computation. Vol 2. Linear Algebra Heidelberg: Springer, 1971.
17. *Jacobi C.G.J.* Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkommenden lineären Gleichungen // Astronomische Nachrichten. 1845. **22**, N 20. 297–306.
18. *Goldstine H.H., Murray F.J., von Neumann J.* The Jacobi method for real symmetric matrices // Journal of the ACM. 1959. **6**, N 1. 59–96.

19. *Yang X.I.A., Mittal R.* Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation // Journal of Computational Physics. 2014. **274**. 695–708.
20. *Adsuara J.E. et al.* Scheduled relaxation Jacobi method: improvements and applications // Journal of Computational Physics. 2016. **321**. 369–413.
21. *Kostenetskiy P.S., Safonov A.Y.* SUSU supercomputer resources// Proc. of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEURWorkshop Proceedings. Vol. 1576. 2016. 561–573.

Поступила в редакцию
19.09.2018

Scalability Evaluation of Iterative Algorithms for Supercomputer Simulation of Physical Processes

N. A. Ezhova [1] **and L. B. Sokolinsky** [2]

[1] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Lecturer, e-mail: EzhovaNA@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 September 19, 2018

Abstract: This paper is devoted to the development of a methodology for evaluating the scalability of compute-intensive iterative algorithms used for simulating complex physical processes on supercomputer systems. The proposed methodology is based on the BSF (Bulk Synchronous Farm) parallel computation model, which makes it possible to predict the upper scalability bound of an iterative algorithm in early stages of its design. The BSF model assumes the representation of the algorithm in the form of operations on lists using high-order functions. Two classes of representations are considered: BSF-M (Map BSF) and BSF-MR (Map-Reduce BSF). The proposed methodology is described by the example of solving a system of linear equations by the Jacobi method. For the Jacobi method, two iterative algorithms are constructed: Jacobi-M based on the BSF-M representation and Jacobi-MR based on the BSF-MR representation. Analytical estimations of the speedup, parallel efficiency and upper scalability bound are obtained for these algorithms using the BSF cost metrics on multi-processor computing systems with distributed memory. These algorithms are implemented on C++ language using the BSF program skeleton and MPI parallel programming library. The results of large-scale computational experiments performed on a cluster computing system are discussed. Based on the experimental results, an analysis of the adequacy of estimations obtained analytically using the BSF cost metric is made.

Keywords: iterative algorithm, BSF parallel computation model, scalability estimation, speedup, parallel efficiency, Jacobi method, cluster computing systems.

References

1. K. Borodulin et al., "Towards Digital Twins Cloud Platform," in *Proc. 10th Int. Conf. on Utility and Cloud Computing* (ACM Press, New York, 2017), pp. 209–210.

2. G. Bilardi and A. Pietracaprina, "Models of Computation, Theoretical," in *Encyclopedia of Parallel Computing* (Springer, Boston, 2011), pp. 1150–1158.

3. J. F. JaJa, "PRAM (Parallel Random AccessMachines)," in *Encyclopedia of Parallel Computing* (Sptinger, Boston, 2011), pp. 1608–1615.

4. L. G. Valiant, "A Bridging Model for Parallel Computation," Commun. ACM **33** (8), 103–111.

5. D. Culler, R. Karp, D. Patterson, et al., "LogP: Towards a Realistic Model of Parallel Computation," in *Proc. 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, San Diego, USA, May 19-22, 1993* (ACM Press, New York, 1993), doi 10.1145/155332.155333.

6. M. Forsell and V. Leppänen, "An Extended PRAM-NUMA Model of Computation for TCF Programming," in *Proc. 2012 IEEE 26th International Parallel and Distributed Processing Symp. Workshops, Shanghai, China, May 21–25, 2012* (IEEE Press, Washington, DC, 2012), pp. 786–793.

7. A. V. Gerbessiotis, "Extending the BSP Model for Multi-Core and Out-of-Core Computing: MBSP," Parallel Comput. **41**, 90–102 (2015).

8. F. Lu , J. Song, and Y. Pang, "HLognGP: A Parallel Computation Model for GPU Clusters," Concurr. Comp-Pract. E. **27** (17), 4880–4896 (2015).

9. N. A. Ezhova and L. B. Sokolinsky, "Parallel Computational Model for Multiprocessor Systems with Distributed Memory," Vestn. South Ural State Univ. Ser. Vychisl. Mat. Inf. **7** (2), 32–49 (2018).

10. L. B. Sokolinsky, "Analytical Estimation of the Scalability of Iterative Numerical Algorithms on Distributed Memory Multiprocessors," Lobachevskii J. Math. **39** 4), 571–575 (2018).

11. 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.

12. F. Darema, "SPMD Computational Model," in *Encyclopedia of Parallel Computing* (Springer, Boston, 2011), pp. 1933–1943.

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

14. I. M. Sokolinskaya and L. B. Sokolinsky, "Scalability Evaluation of the NSLP Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems," in *Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia, September 25-26, 2017* (Mosk. Gos. Univ., Moscow, 2017), pp. 319–332.

15. M. I. Cole, "Parallel Programming with List Homomorphisms," Parallel Proc. Lett. 5 (2), 191-203 (1995).

16. H. Rutishauser, "The Jacobi Method for Real Symmetric Matrices," in *Handbook for Automatic Computation. Vol 2. Linear Algebra* (Springer, Heidelberg, 1971), pp. 202–211.

17. C. G. J. Jacobi, "Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkommenden lineären Gleichungen," Astronomische Nachrichten **22** (20), 297–306 (1845).

18. H. H. Goldstine, F. J. Murray, and J. von Neumann, "The Jacobi Method for Real Symmetric Matrices," J. ACM **6** (1), 59–96 (1959).

19. X. I. A. Yang and R. Mittal, "Acceleration of the Jacobi Iterative Method by Factors Exceeding 100 Using Scheduled Relaxation," J. Comput. Phys. **274**, 695–708 (2014).

20. J. E. Adsuara et al., "Scheduled Relaxation Jacobi Method: Improvements and Applications," J. Comput. Phys. **321**, 369–413 (2016).

21. 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).