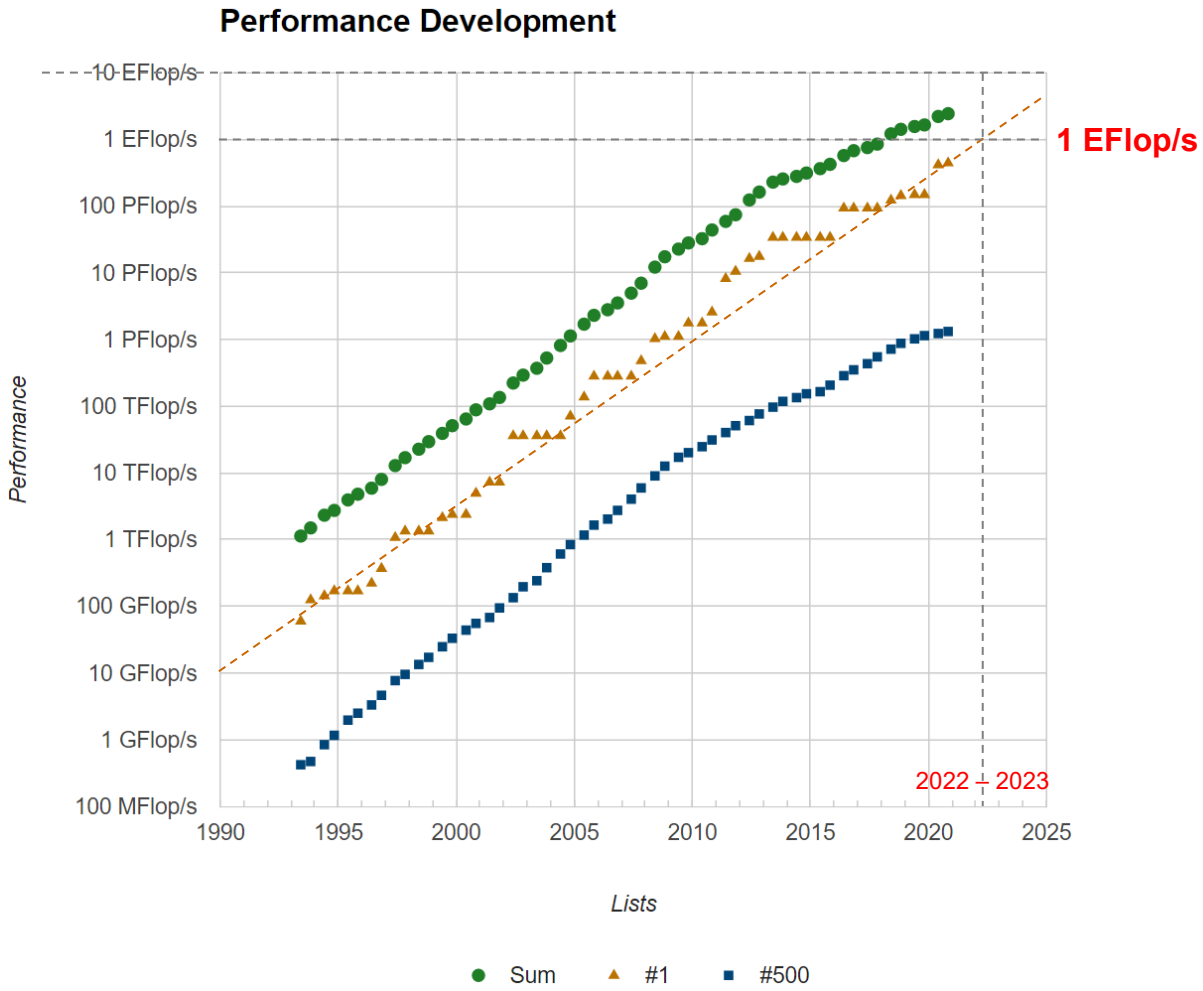
Модель параллельных вычислений для оценки масштабируемости итерационных численных алгоритмов на многопроцессорных системах с распределенной памятью

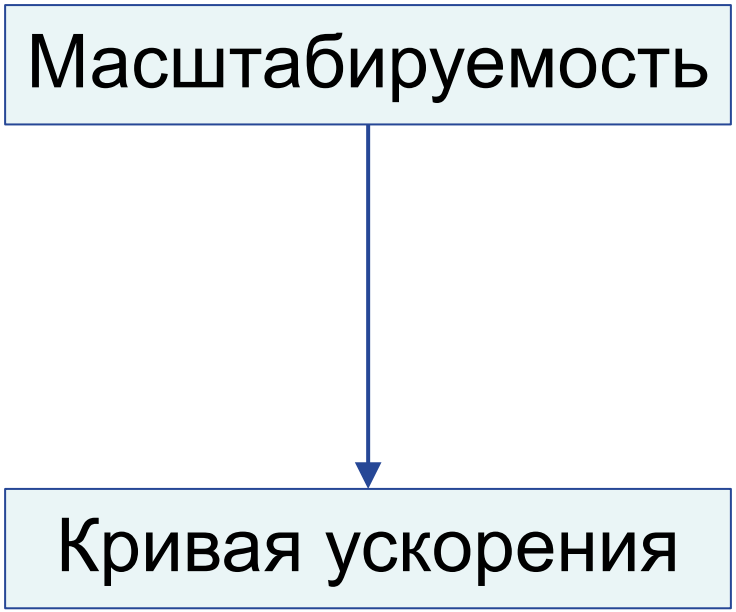
д.ф.-м.н., профессор Л.Б. Соколинский

Южно-Уральский государственный университет
(национальный исследовательский университет)
Челябинск

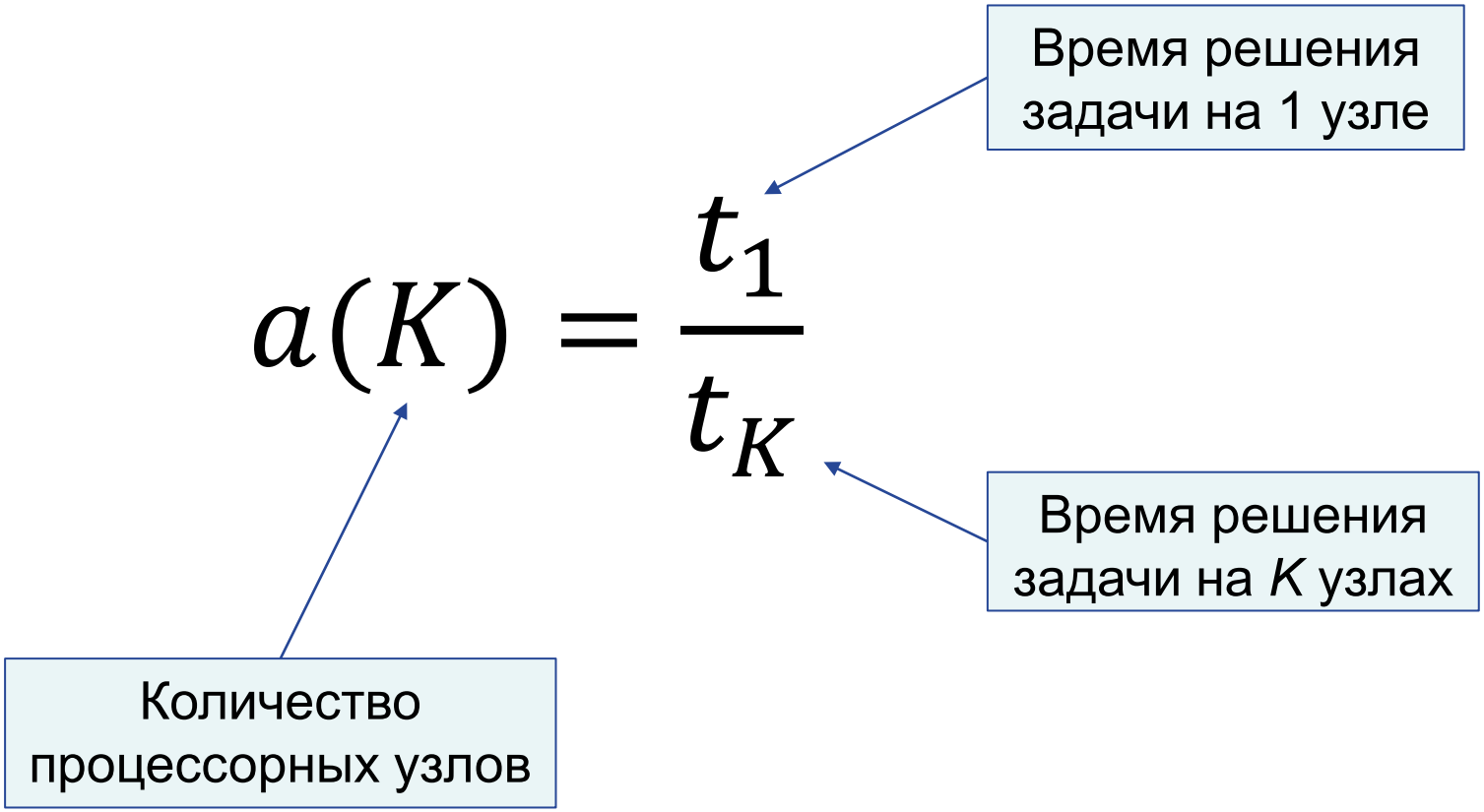
TOP500 (ноябрь 2020)



Главная характеристика современного параллельного численного алгоритма



Ускорение


Время решения задачи на 1 узле
$a(K) = \frac{t_1}{t_K}$
Время решения задачи на K узлах
Количество процессорных узлов

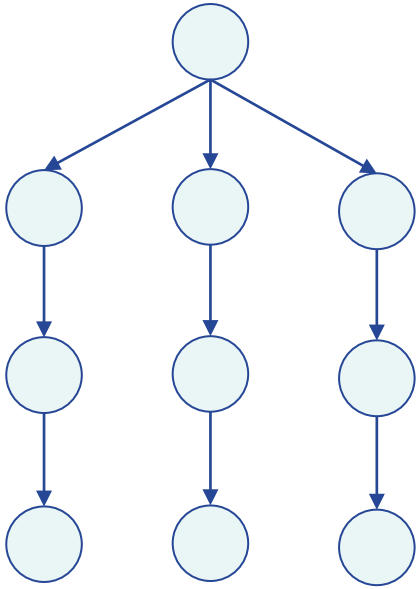
Алгоритм является хорошо масштабируемым, если:

кривая ускорения близка к линейной
с положительной производной

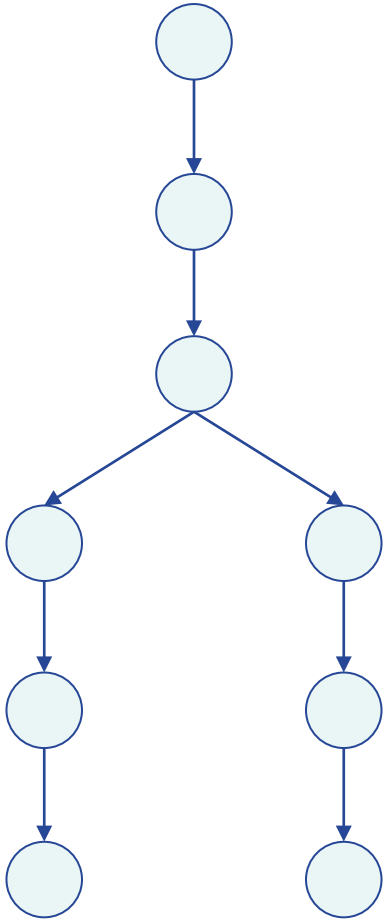
Фундаментальные факторы, влияющие на масштабируемость алгоритма

1. Информационная структура алгоритма
2. Закон Амдала
3. Размер задачи

Информационная структура алгоритма



a) Большой ресурс параллелизма



b) Средний ресурс параллелизма

c) Ресурс параллелизма отсутствует

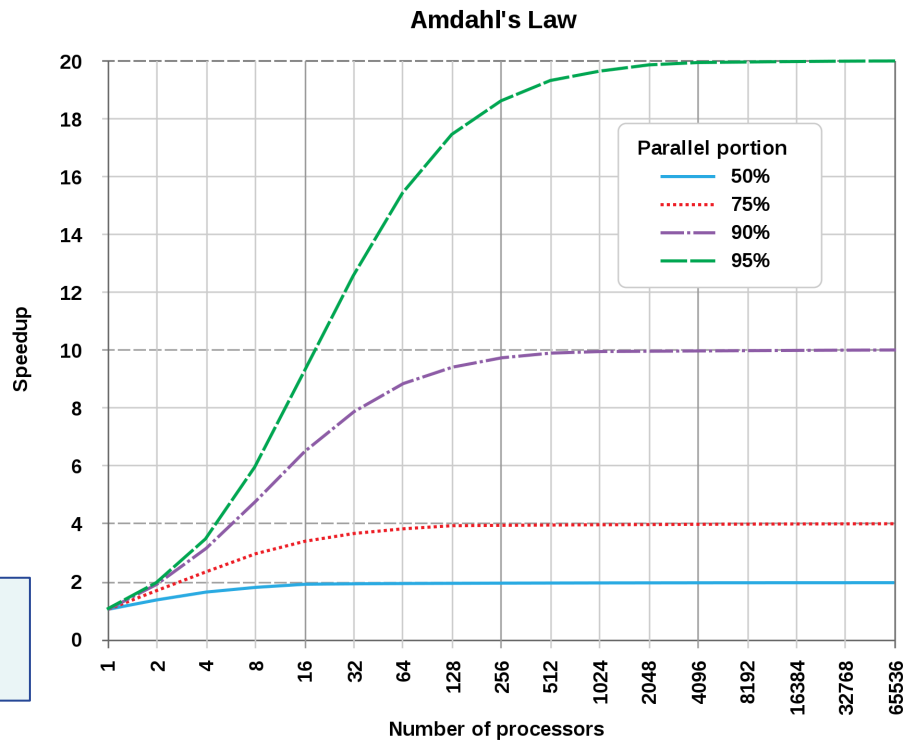
Закон Амдала

t_1- время решения задачи на 1 узле

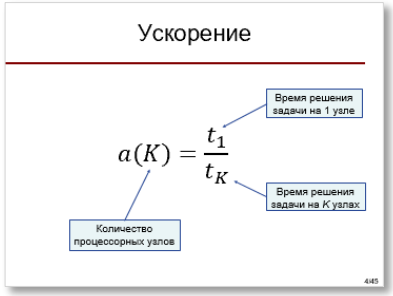
$$t_K = \delta t_1 + \frac{(1-\delta)t_1}{K}$$

δ - доля последовательных вычислений

K – количество процессорных узлов

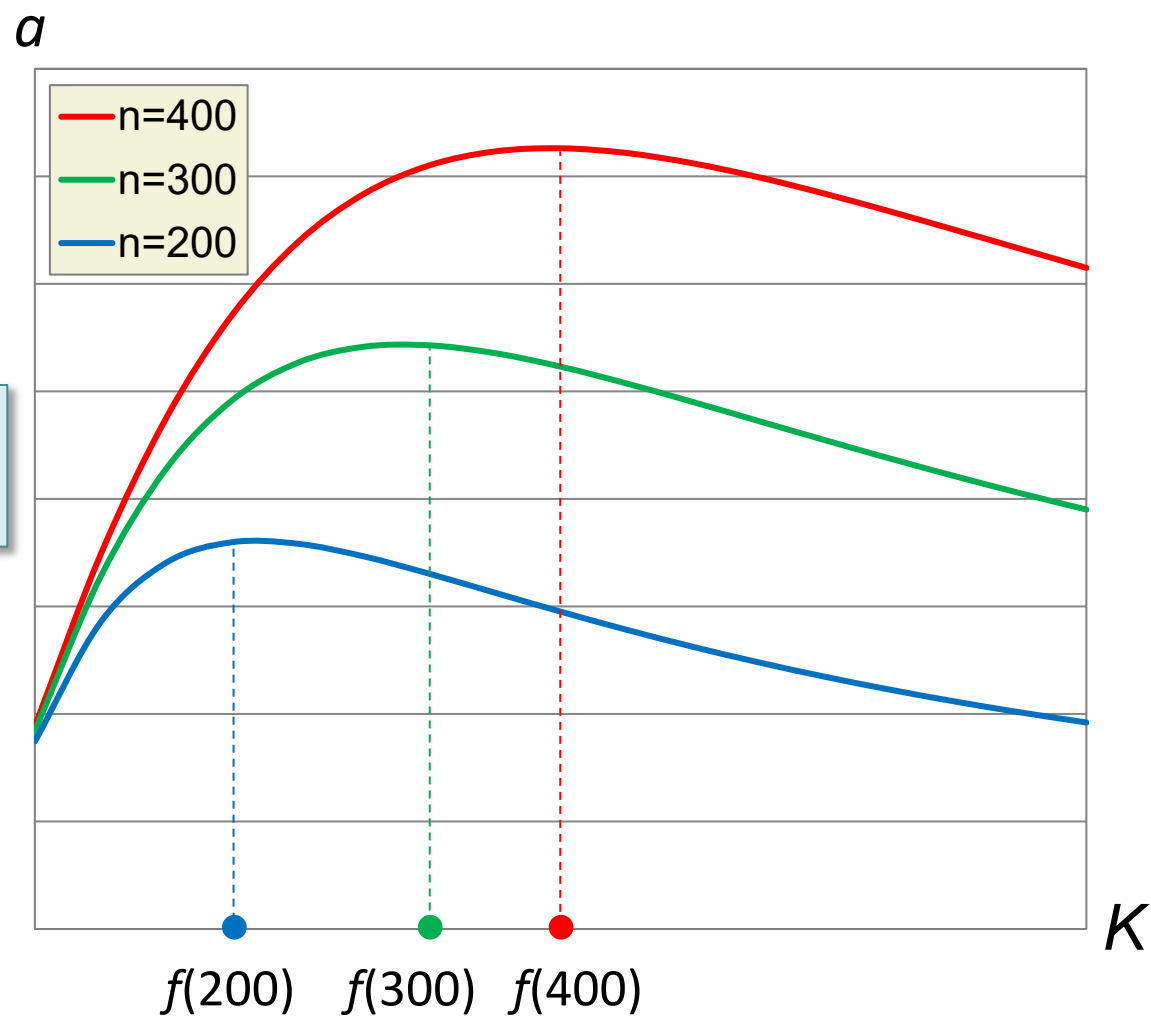


$$a(K) = \frac{1}{\delta + \frac{(1-\delta)}{K}}$$



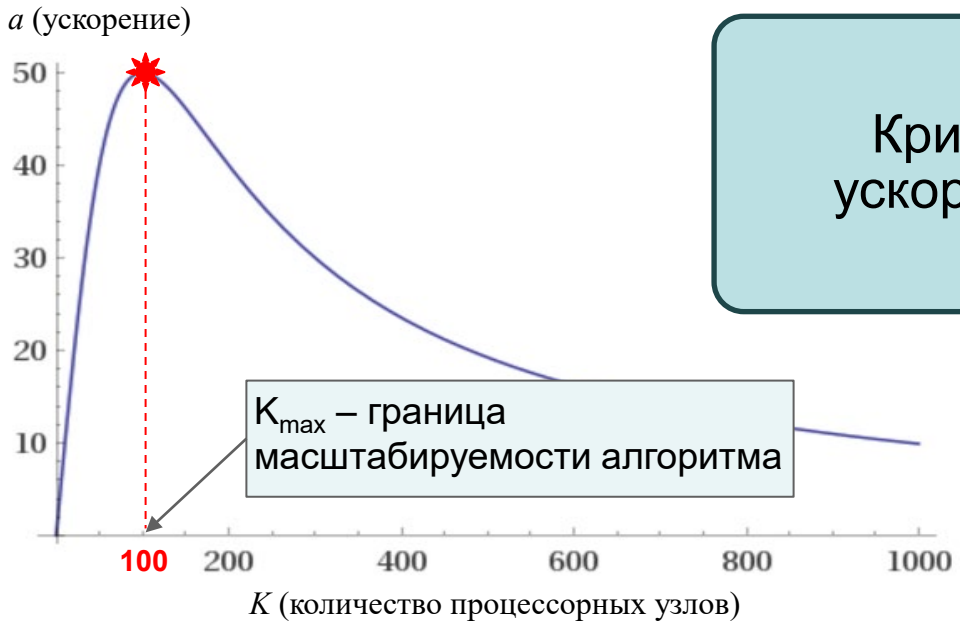
Масштабируемость зависит от размера задачи *n*

$$K_{max} = f(n)$$



Цикл исследования масштабируемости алгоритма

Разработка алгоритма → Кодирование → Тестирование → Вычислительные эксперименты → Кривая ускорения



Желание

Оценить границу масштабируемости алгоритма **ДО** написания программы **БЕЗ** вычислительных экспериментов

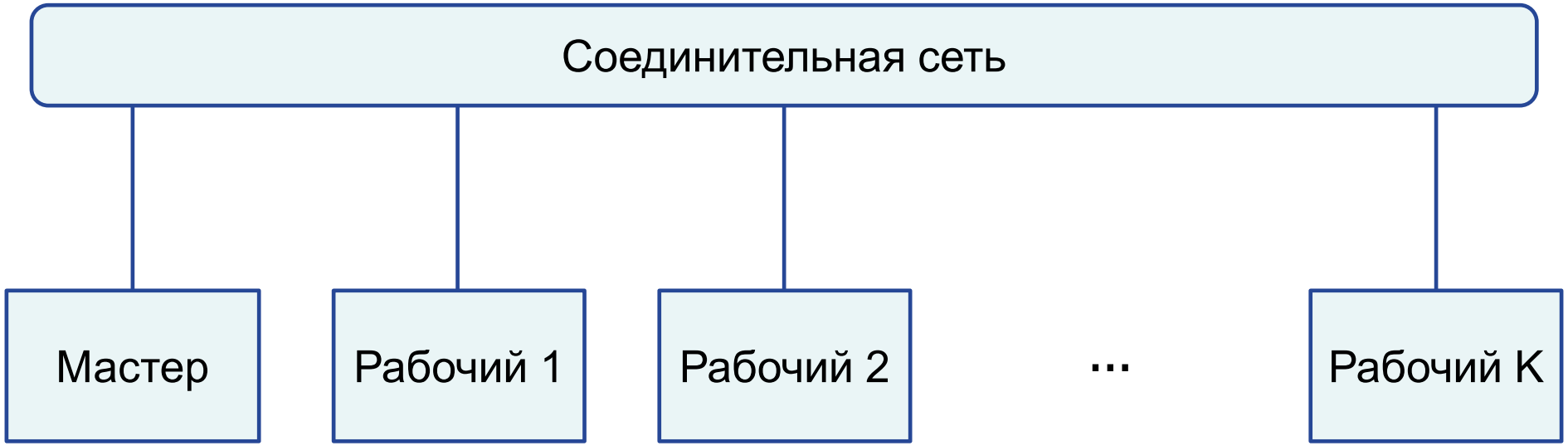
Модель параллельных вычислений BSF (Bulk-Synchronous Farm)

- **Область применения:**
  - Многопроцессорные системы с распределенной памятью
  - Параллельные итерационные алгоритмы с высокой вычислительной сложностью
- **Позволяет предсказать:**
  - ускорение параллельного алгоритма
  - **границу масштабируемости параллельного алгоритма** (уникальное качество модели BSF)

Ограничения на алгоритм

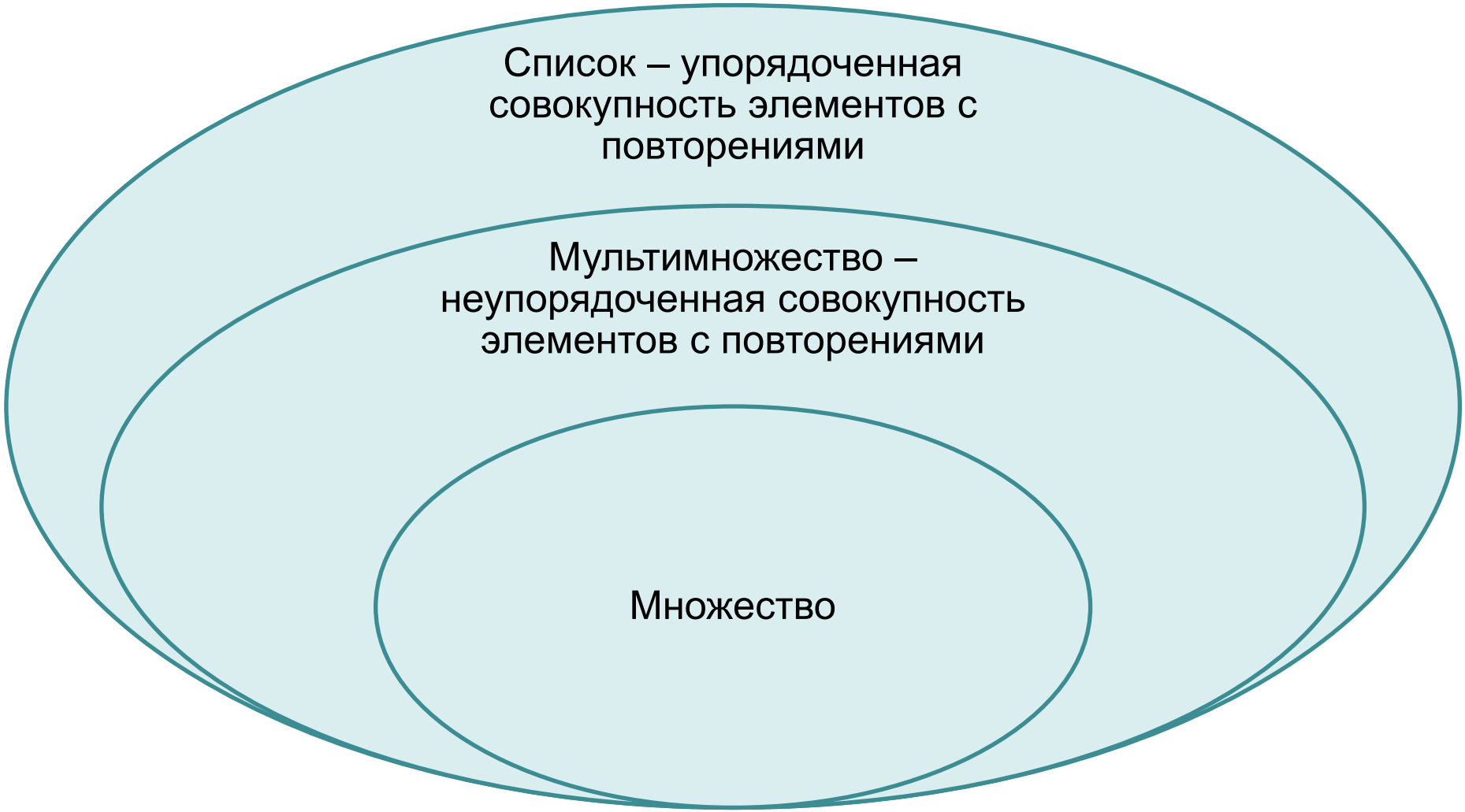
1. Распараллеливание по схеме Мастер/рабочие
2. Алгоритм должен быть представлен в виде операций над списками с использованием функций высшего порядка Map/Reduce

BSF-компьютер

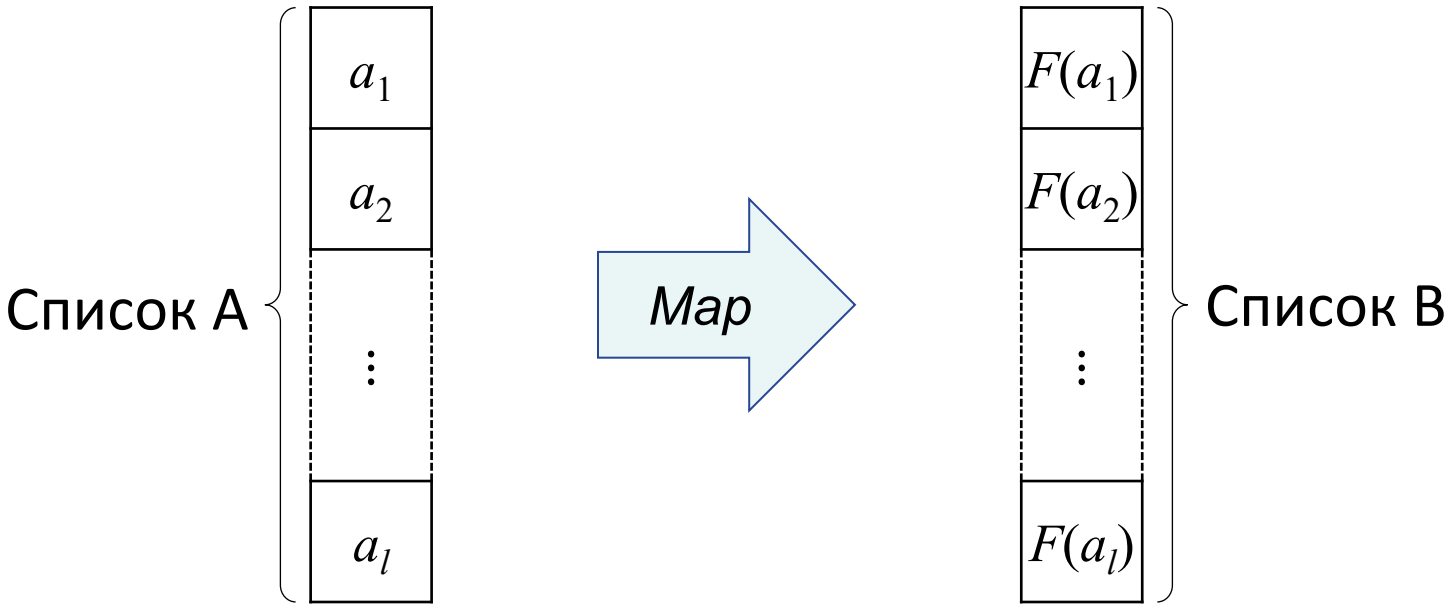


Процессорные узлы

Представление алгоритма в виде операций над списками



Функция высшего порядка *Map*



$$Map(F, A) = B$$

Ограничение применимости для множеств

BSF-модель НЕ применима для множеств в случае, когда функция *Map* порождает дубликаты

Функция высшего порядка *Reduce*

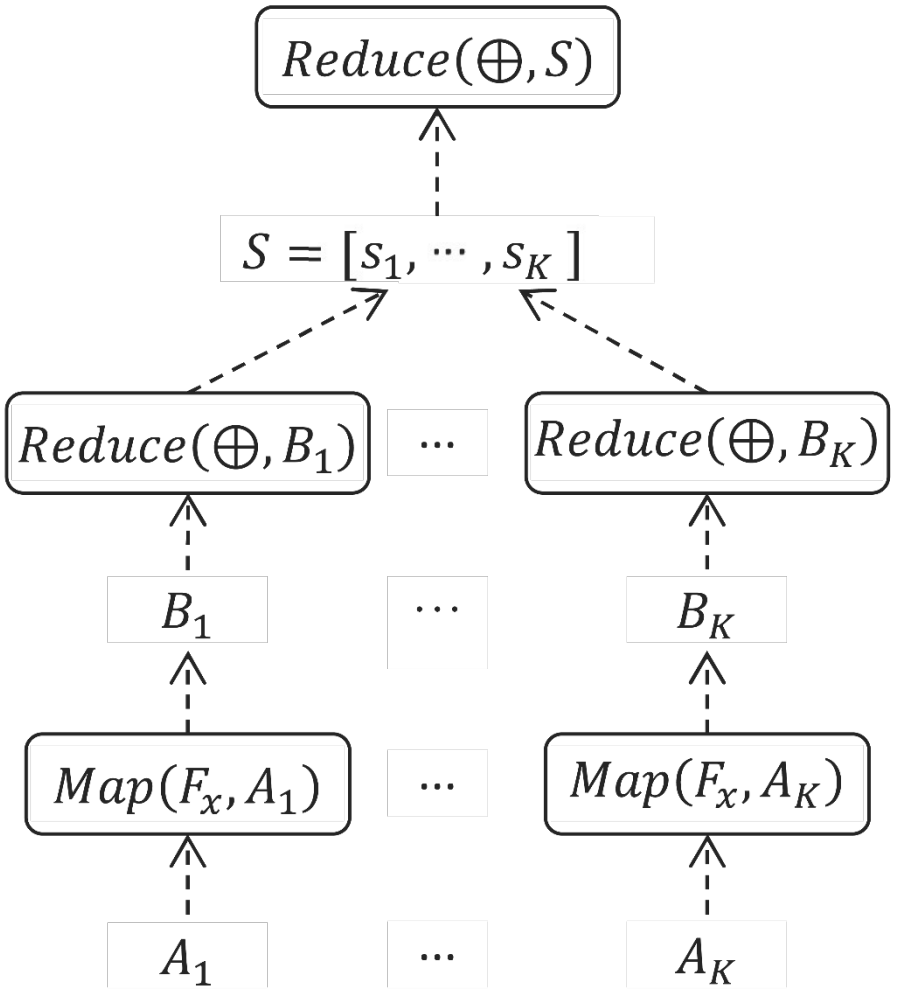
Шаблон последовательного BSF-алгоритма

i	- счетчик итераций
$A \in [\mathcal{A}]$	- список исходных элементов данных
$x^{(0)}$	- начальное приближение
$F_x: \mathcal{A} \to \mathcal{B}$	- параметризованная функция
$B \in [\mathcal{B}]$	- список результирующих элементов
$\oplus$	- ассоциативная операция
$x^{(i)}$	- *i*-тое приближение

1. $Input\left(A, x^{(0)}\right)$
2. $i := 0$
3. $B := Map\left(F_{x^{(i)}}, A\right)$
4. $s := Reduce(\oplus, B)$
5. $x^{(i+1)} := Compute\left(x^{(i)}, s\right)$;
6. $i := i + 1$
7. **if** $StopCond\left(x^{(i)}, x^{(i-1)}\right)$ **go to** 9
8. **go to** 3
9. $Output\left(x^{(i)}\right)$;
10. **stop**

Схема распараллеливания BSF-алгоритма

$$A = A_1 \mathbin{\#} \cdots \mathbin{\#} A_K$$



Шаблон параллельного алгоритма в модели BSF

Шаг	Мастер	Рабочий j (j=1,…,K)
1.	$i := 0; Input(x^{(0)})$	$Input(A_j)$
2.	$SendToAllWorkers(x^{(i)})$	$RecvFromMaster(x^{(i)})$
3.		$B_j := Map(F_{x^{(i)}}, A_j)$
4.		$s_j := Reduce(\oplus, B_j)$
5.	$RecvFromAllWorkers(s_1, \dots, s_K)$	$SendToMaster(s_j)$
6.	$s := Reduce(\oplus, [s_1, \dots, s_K])$	
7.	$x^{(i+1)} := Compute(x^{(i)}, s)$	
8.	$i := i + 1$	
9.	$exit := StopCond(x^{(i+1)}, x^{(i)})$	
10.	$SendToAllWorkers(exit)$	$RecvFromMaster(exit)$
11.	**if not** $exit$ **goto** 2	**if not** $exit$ **goto** 2
12.	$Output(x^{(i)})$	
13.	**stop**	**stop**

Стоимостная метрика модели BSF

K – количество рабочих процессорных узлов

l – длина списков A и B

L – латентность (время посылки сообщения длиной в 1 байт)

t_c – время, которое тратит мастер на пересылку текущего приближения рабочему и на получение от него Reduce-свертки с учетом латентности

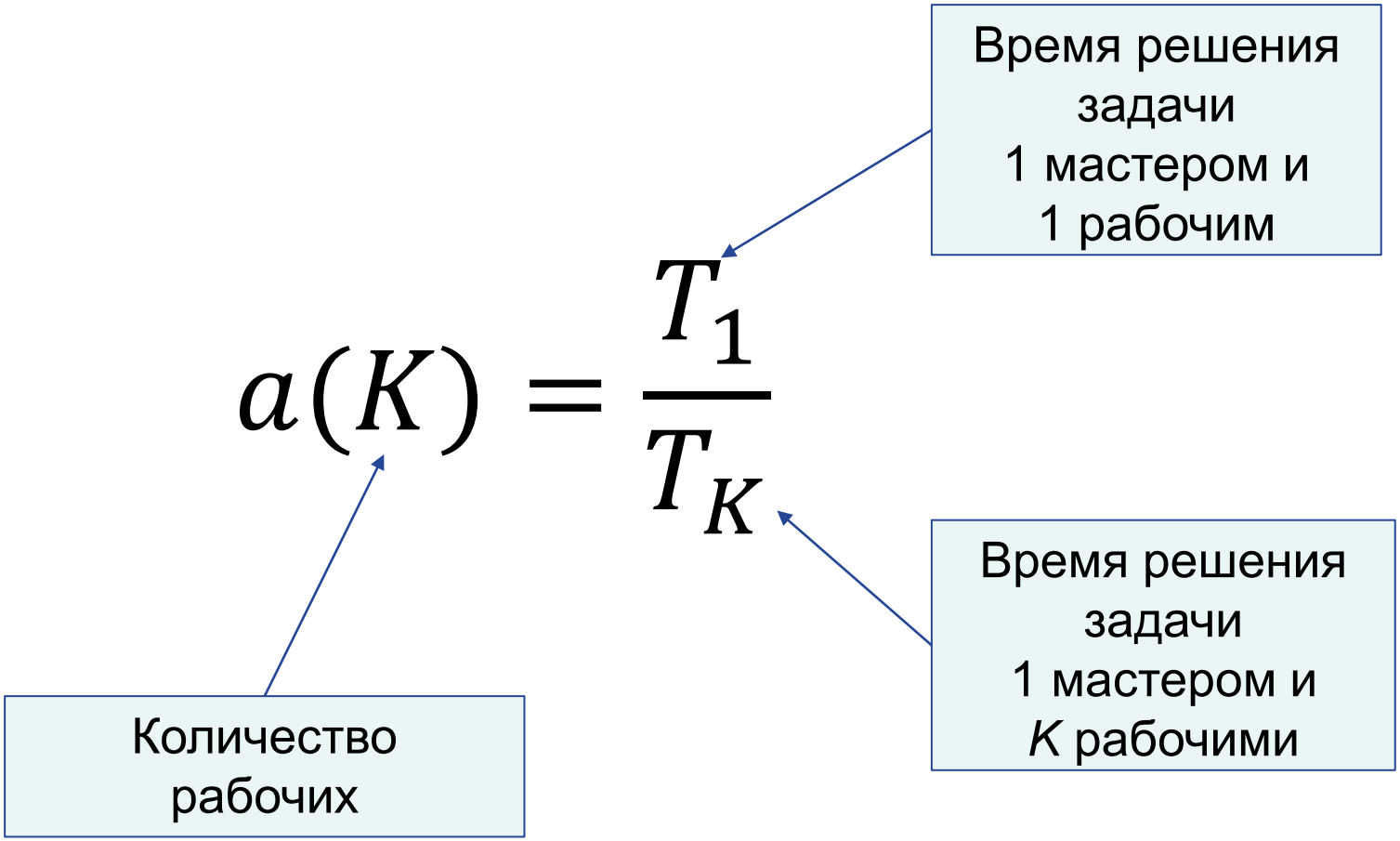
t_{Map} – время выполнения функции *Map* для всего списка A

t_{Rdc} – время выполнения функции *Reduce* для всего списка B

t_p – время на обработку результатов итерации и вычисление следующего приближения

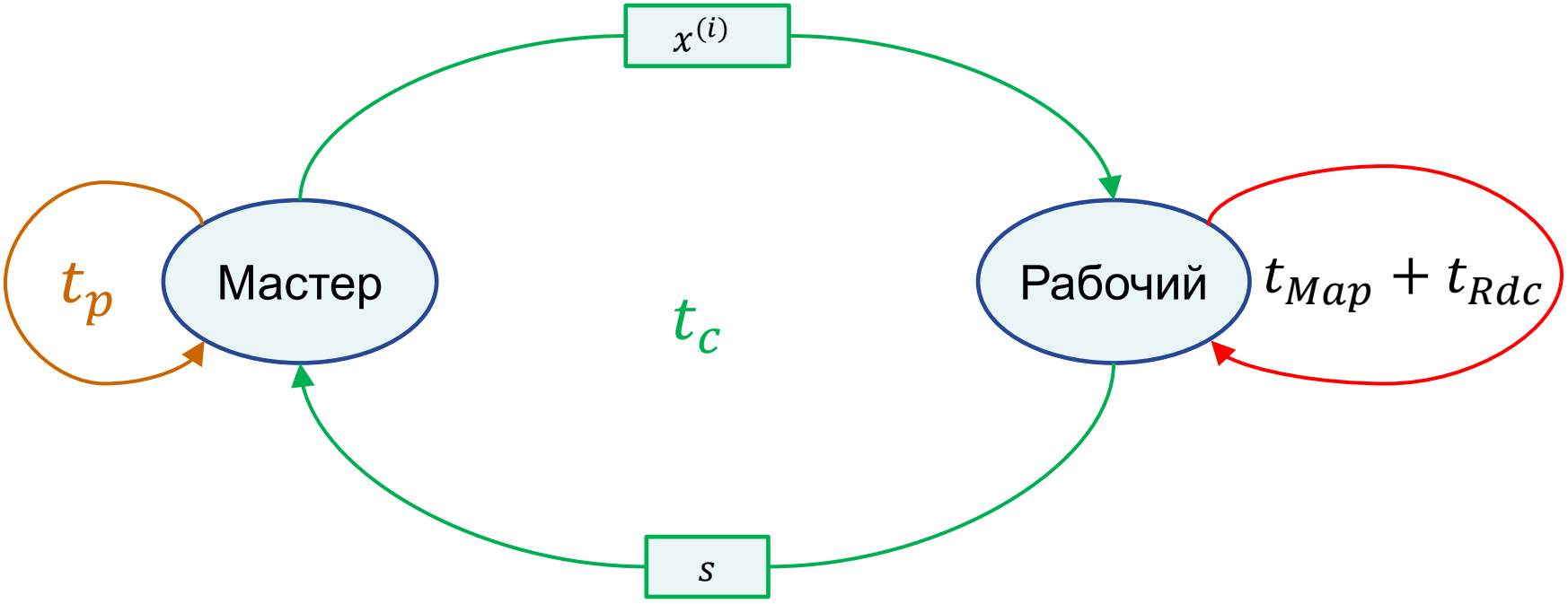
t_a – время выполнения операции $\oplus$: $t_a = t_{Rdc}/(l-1)$

Ускорение в модели BSF



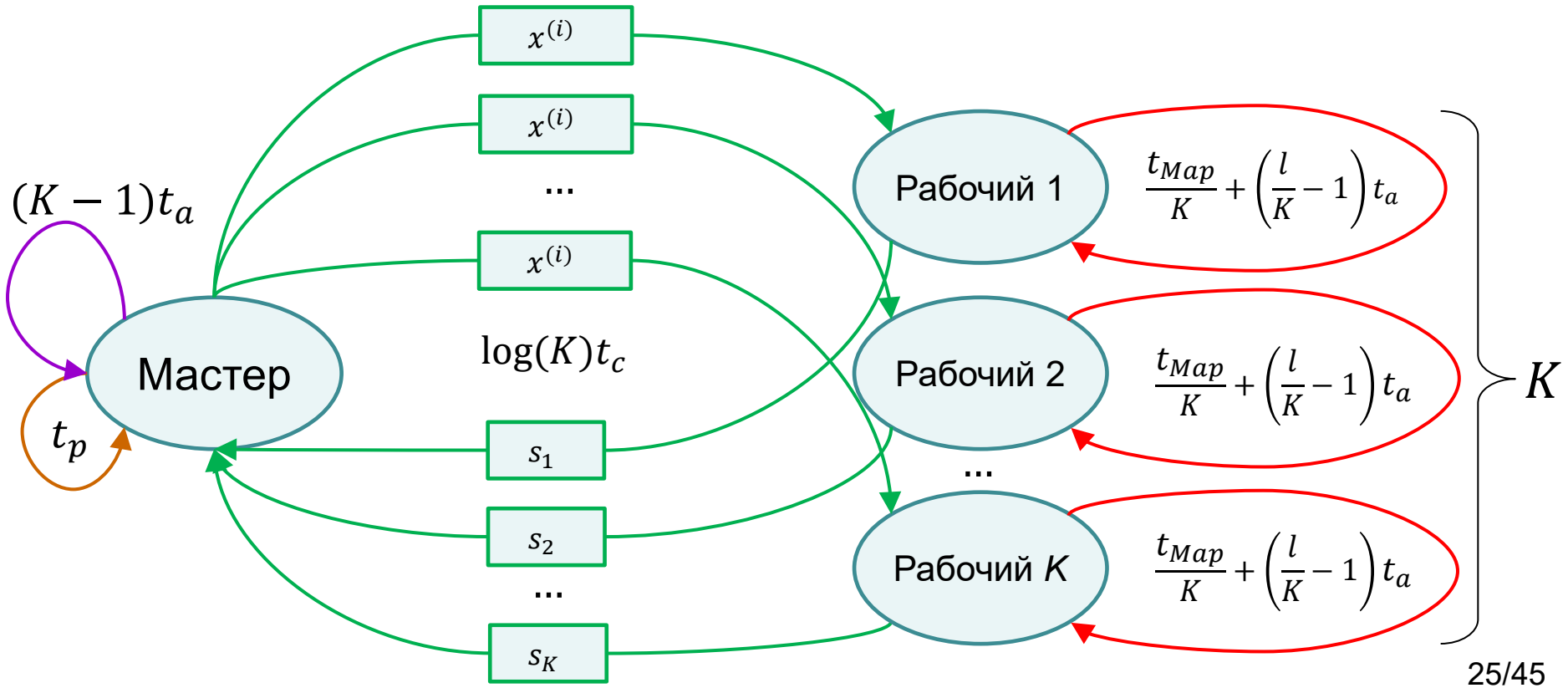
Оценка времени T_1

$$T_1 = t_p + t_c + t_{Map} + t_{Rdc}$$



Оценка времени T_K

$$T_K = (K-1)t_a + t_p + (\log_2(K)+1)t_c + \frac{t_{Map}+(l-K)t_a}{K}$$



Ускорение для модели BSF

$$a_{BSF}(K) = \frac{t_p + t_c + t_{Map} + t_{Rdc}}{(K-1)t_a + t_p + (\log_2(K)+1)t_c + \frac{t_{Map} + (l-K)t_a}{K}}$$

Свойства:

$a_{BSF}(1) = 1$

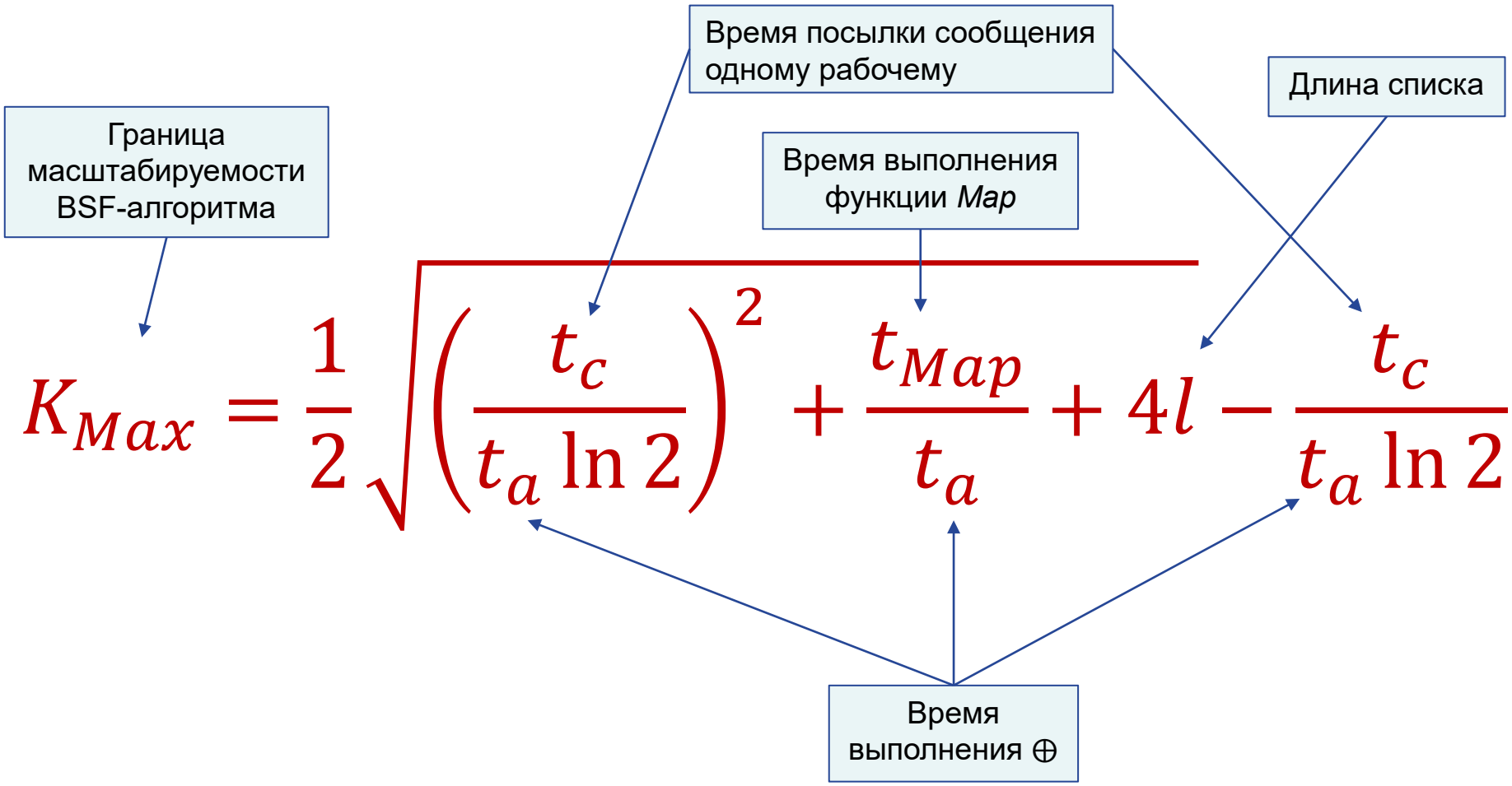
$a_{BSF}(K) > 0$

$\lim_{t_{comp} \to 0} a_{BSF}(K) = \frac{1}{\log_2(K)+1}$ – монотонно убывает на $[1; +\infty)$

$t_{Map} + t_{Rdc} + t_p$

$t_{Rdc} = (l-1)t_a$

Граница масштабируемости BSF-алгоритма



Алгоритм Jacobi для приближенного решения СЛАУ

$$Ax = b$$

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

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

$$d = (d_1, \dots, d_n) \qquad d_i = b_i / a_{ii}$$

$$x^{(k+1)} = Cx^{(k)} + d$$

Функция для *Map*

$$F_x(j) = \left(c_{1j}x_j, \dots, c_{nj}x_j\right) = x_j \begin{pmatrix} c_{1j} \\ \vdots \\ c_{nj} \end{pmatrix}^T$$

j – номер столбца матрицы C

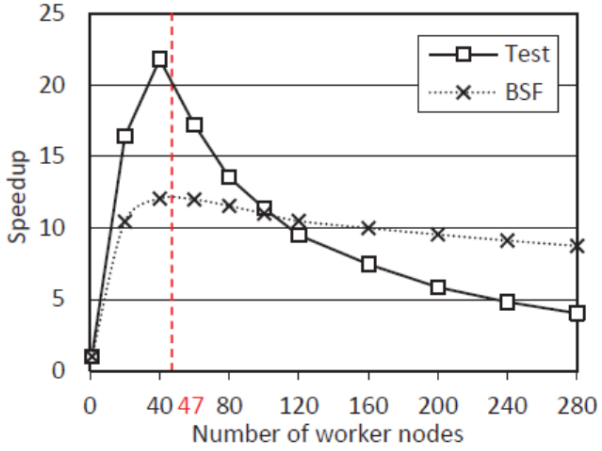
Алгоритм Jacobi-MR над списками

1. $Input(C, d); k := 0; x^{(0)} := d$
2. $\left[g^1, \ldots, g^n\right] := Map\left(F_{x^{(k)}}, [1, \ldots, n]\right)$
3. $g := Reduce\left(+, \left[g^1, \ldots, g^n\right]\right)$
4. $x^{(k+1)} := g + d; k := k + 1$
5. **if** $\left\|x^{(k)} - x^{(k-1)}\right\|^2 < \varepsilon$ **go to** 7
6. **go to** 2
7. $Output\left(x^{(k)}\right)$; **stop**

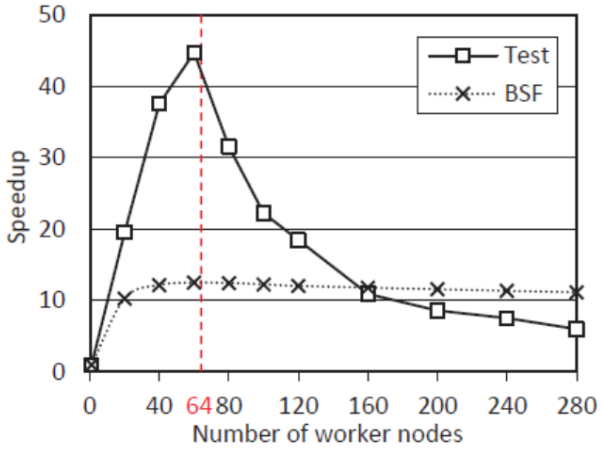
Алгоритм Jacobi для приближенного решения СЛАУ

$$K_{Jacobi} = O\left(\sqrt{n}\right)$$

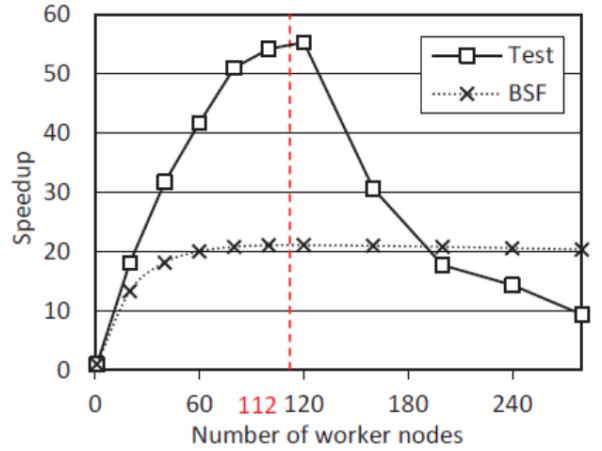
Алгоритм Jacobi для приближенного решения СЛАУ



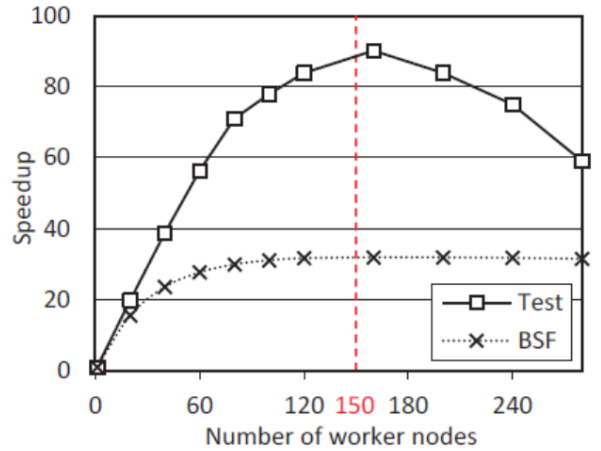
(a) $n = 1\,500$



(b) $n = 5\,000$



(c) $n = 10\,000$



(c) $n = 16\,000$

Ошибка для алгоритм Jacobi

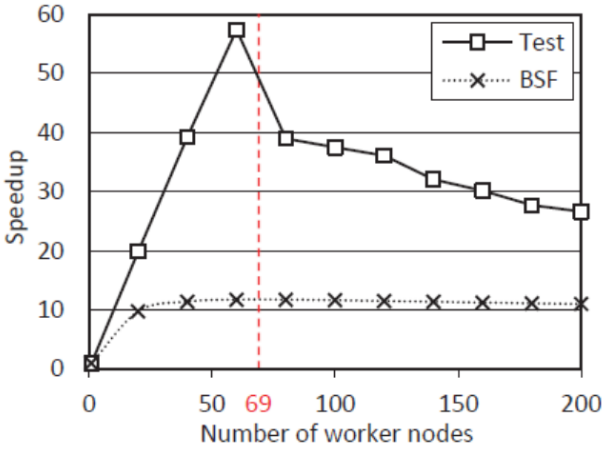
n	**1 500**	**5 000**	**10 000**	**16 000**
K_{BSF}	47	64	112	150
K_{test}	40	60	120	160
$Error$	**0.15**	**0.06**	**0.07**	**0.06**

$$Error = \frac{|K_{test} - K_{BSF}|}{\max(K_{test}, K_{BSF})}$$

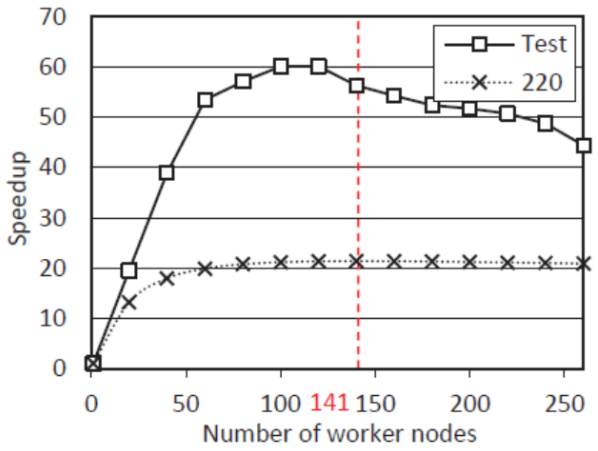
Задача n-тел

$$K_{Max} = O(\sqrt{n})$$

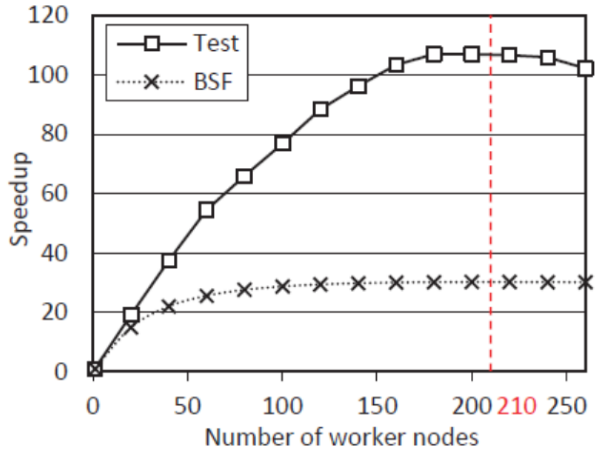
Задача n-тел



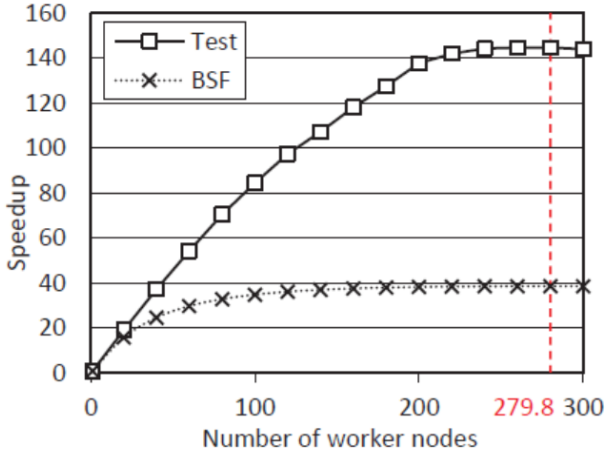
(a) $n = 1\,500$



(b) $n = 5\,000$



(c) $n = 10\,000$



(c) $n = 16\,000$

Ошибка для задачи n-тел

n	**300**	**600**	**900**	**12 000**
K_{BSF}	69	141	210	279.1
K_{test}	60	140	200	280
$Error$	**0.13**	**0.01**	**0.05**	**3.6E-4**

$$Error = \frac{|K_{test} - K_{BSF}|}{\max(K_{test}, K_{BSF})}$$

Параллельный BSF-каркас (C++ & MPI)

- 100% автоматическое распараллеливание
- Поддержка потоков работ
- Поддержка OpenMP
- Инкрементная компиляция
- Исходные коды + документация
  https://github.com/leonid-sokolinsky/BSF-skeleton

В чем отличие BSF от MapReduce?

- Технология MapReduce ориентирована на обработку данных (обмены с внешними устройствами и пересылки по сети превалируют над вычислениями)
- Модель BSF ориентирована на алгоритмы с высокой вычислительной сложностью (вычисления превалируют над обменами с внешними устройствами и пересылками по сети)

Можно ли адаптировать модель BSF для конфигурации, в которой присутствуют два или более узла-мастера?

Нет!

Можно ли использовать BSF для алгоритма, где есть *Map*, но нет *Reduce*?

- Да!
- Вариант алгоритма Якоби без *Reduce*: https://github.com/leonid-sokolinsky/BSF-Jacobi-Map

Можно ли использовать BSF для списка из одного элемента?

- Да!
- Генератор случайных задач линейного программирования: https://github.com/leonid-sokolinsky/BSF-LPP-Generator

Можно ли использовать BSF для неитерационного алгоритма?

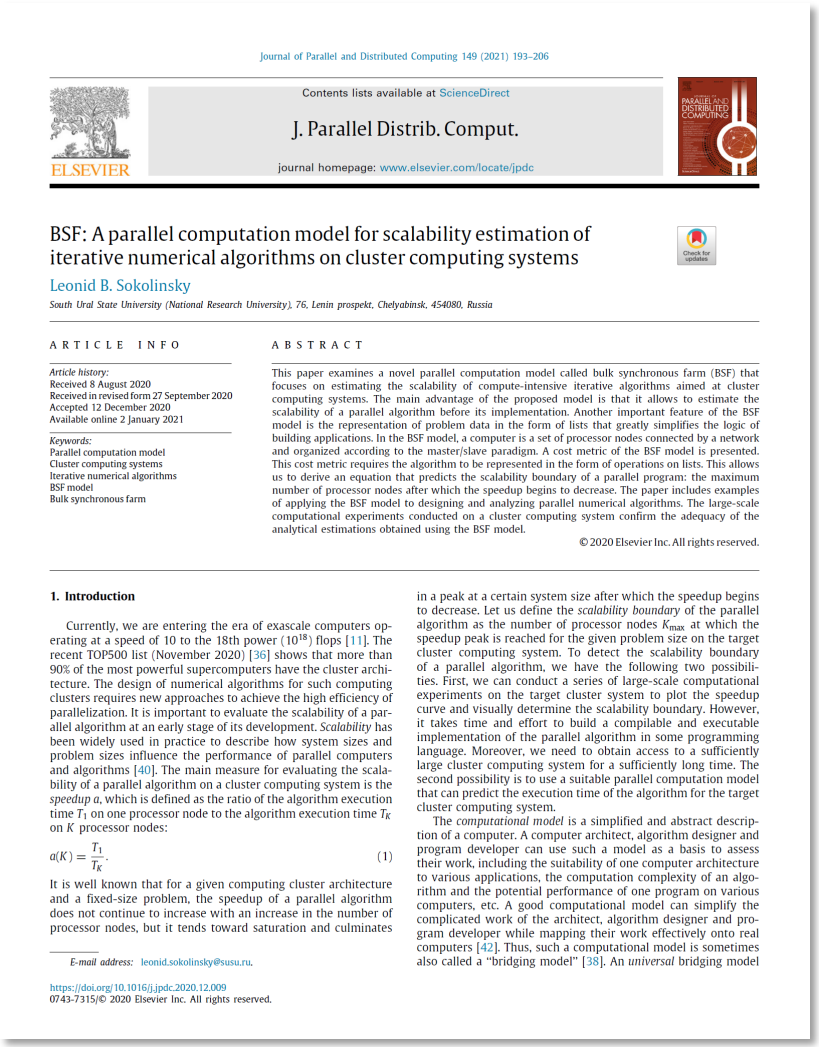
- Да!
- Валидатор решений задач линейного программирования: https://github.com/leonid-sokolinsky/BSF-LPP-Validator

Существует ли пример BSF-приложения с использованием потоков работ?

- Да!
- Решатель задач линейного программирования апекс-методом: https://github.com/leonid-sokolinsky/Apex-method

Основная публикация по модели BSF

Sokolinsky L.B. BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems // Journal of Parallel and Distributed Computing. 2021. Vol. 149. P. 193-206. DOI:10.1016/j.jpdc.2020.12.009. [Full Text in PDF]



Journal of Parallel and Distributed Computing 149 (2021) 193–206

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems

Leonid B. Sokolinsky

South Ural State University (National Research University), 76, Lenin prospekt, Chelyabinsk, 454080, Russia

ARTICLE INFO

Article history:
Received 8 August 2020
Received in revised form 27 September 2020
Accepted 12 December 2020
Available online 2 January 2021

Keywords:
Parallel computation model
Cluster computing systems
Iterative numerical algorithms
BSF model
Bulk synchronous farm

ABSTRACT

This paper examines a novel parallel computation model called bulk synchronous farm (BSF) that focuses on estimating the scalability of compute-intensive iterative algorithms aimed at cluster computing systems. The main advantage of the proposed model is that it allows to estimate the scalability of a parallel algorithm before its implementation. Another important feature of the BSF model is the representation of problem data in the form of lists that greatly simplifies the logic of building applications. In the BSF model, a computer is a set of processor nodes connected by a network and organized according to the master/slave paradigm. A cost metric of the BSF model is presented. This cost metric requires the algorithm to be represented in the form of operations on lists. This allows us to derive an equation that predicts the scalability boundary of a parallel program: the maximum number of processor nodes after which the speedup begins to decrease. The paper includes examples of applying the BSF model to designing and analyzing parallel numerical algorithms. The large-scale computational experiments conducted on a cluster computing system confirm the adequacy of the analytical estimations obtained using the BSF model.

1. Introduction

Currently, we are entering the era of exascale computers operating at a speed of 10 to the 18th power (10^{18}) flops [11]. The recent TOP500 list (November 2020) [36] shows that more than 90% of the most powerful supercomputers have the cluster architecture. The design of numerical algorithms for such computing clusters requires new approaches to achieve the high efficiency of parallelization. It is important to evaluate the scalability of a parallel algorithm at an early stage of its development. *Scalability* has been widely used in practice to describe how system sizes and problem sizes influence the performance of parallel computers and algorithms [40]. The main measure for evaluating the scalability of a parallel algorithm on a cluster computing system is the *speedup a*, which is defined as the ratio of the algorithm execution time T_1 on one processor node to the algorithm execution time T_K on K processor nodes:

$$a(K) = \frac{T_1}{T_K}. \quad (1)$$

It is well known that for a given computing cluster architecture and a fixed-size problem, the speedup of a parallel algorithm does not continue to increase with an increase in the number of processor nodes, but it tends toward saturation and culminates in a peak at a certain system size after which the speedup begins to decrease. Let us define the *scalability boundary* of the parallel algorithm as the number of processor nodes K_{max} at which the speedup peak is reached for the given problem size on the target cluster computing system. To detect the scalability boundary of a parallel algorithm, we have the following two possibilities. First, we can conduct a series of large-scale computational experiments on the target cluster system to plot the speedup curve and visually determine the scalability boundary. However, it takes time and effort to build a compilable and executable implementation of the parallel algorithm in some programming language. Moreover, we need to obtain access to a sufficiently large cluster computing system for a sufficiently long time. The second possibility is to use a suitable parallel computation model that can predict the execution time of the algorithm for the target cluster computing system.

The *computational model* is a simplified and abstract description of a computer. A computer architect, algorithm designer and program developer can use such a model as a basis to assess their work, including the suitability of one computer architecture to various applications, the computation complexity of an algorithm and the potential performance of one program on various computers, etc. A good computational model can simplify the complicated work of the architect, algorithm designer and program developer while mapping their work effectively onto real computers [42]. Thus, such a computational model is sometimes also called a "bridging model" [38]. An *universal* bridging model

E-mail address: leonid.sokolinsky@susu.ru.

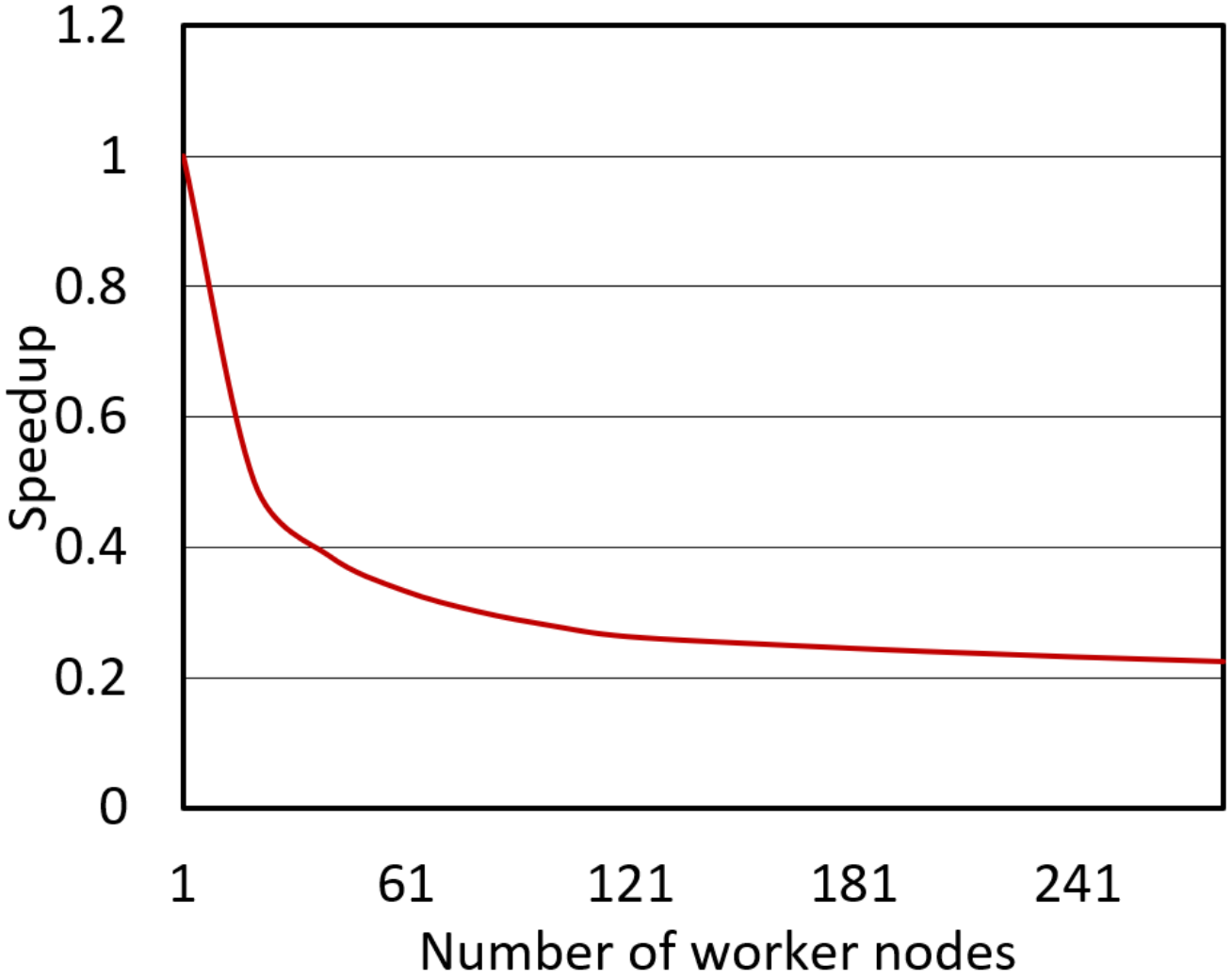
https://doi.org/10.1016/j.jpdc.2020.12.009
0743-7315/

Спасибо за внимание!

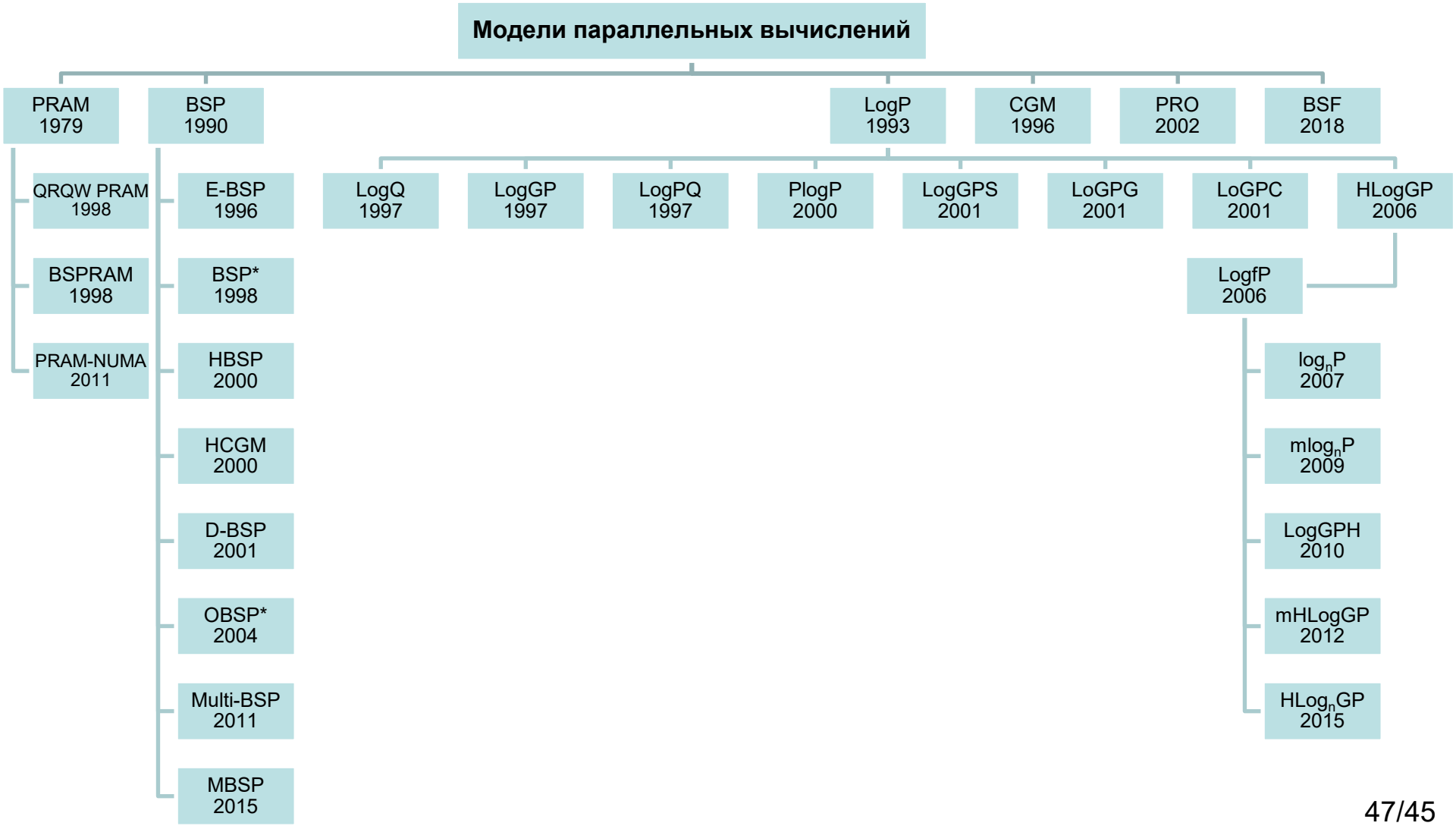
Вопросы?

Исходные коды + документация
https://github.com/leonid-sokolinsky/BSF-skeleton

График функции $a(K) = \frac{1}{\log_2(K)+1}$



Дерево моделей параллельных вычислений



Модели вычислений – мосты между алгоритмами компьютерами