РЕШЕНИЕ МИНИМАКСНОЙ РАСПРЕДЕЛИТЕЛЬНОЙ ЗАДАЧИ МЕТОДОМ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ С ПРИМЕНЕНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

А.М. Григорьев

1. Введение

Современный высокотехнологичный мир предъявляет высокие требования к безопасности окружающих нас сложных систем таких, например, как атомные электростанции и химическое производство. В случае непредвиденных ситуаций нередко слаженная и оперативная работа аварийно-спасательных формирований позволяет оперативно восстановить критические функции безопасности АЭС, существенно снизить вредные последствия и предотвратить возможную техногенную катастрофу. Оптимизация действий бригады работников в чрезвычайных ситуациях позволяет обеспечить скорейшее решение поставленной задачи при минимальном риске для здоровья спасателей и максимальной экономии средств. Формально задачу оптимизации действий такой бригады можно представить как задачу распределения конечного числа заданий среди конечного числа исполнителей. В случае равенства числа исполнителей и заданий имеет место известная задача о назначениях (assignment problem) [1], успешно решаемая с помощью методов линейного программирования за полиномиальное время (см., например, венгерский алгоритм) [2]. Обобщенная постановка задачи о назначениях [3], в которой множество заданий и множество работников могут не совпадать по мощности, а также присутствует ограничение на ресурсы работников, уже не только NP-трудна, но и APX-трудна [4], то есть не допускает существования полиномиального приближенного алгоритма, решающего задачу с заданной наперед точностью. Обе приведенные задачи рассматриваются обычно в двух постановках: на минимизацию совокупных затрат (максимизацию совокупного дохода) и на минимизацию наибольших по всем работникам затрат (максимизацию наименьшего дохода). Второй вариант (связываемый обычно с добавкой в названии проблемы "узкое горлышко" или "bottleneck") рассматривается в печати несколько реже. Именно этим вариантом мы будем интересоваться в настоящей статье, имея в виду, что бригаде исполнителей необходимо закончить работы в кратчайшие сроки как по причине скорейшей нейтрализации вредных последствий аварии, так и из соображений минимизации вредных воздействий на каждого из исполнителей. Далее мы считаем, что затрачиваемый исполнителями ресурс и получаемый доход - это один параметр. (Иными словами экономию мы и считаем выигрышем, будь то экономия времени, затрачиваемого на ликвидацию аварии, минимизация вреда для здоровья спасателей или уменьшение ущерба для техники.) В таких условиях рассмотренная выше обобщенная задача о назначениях теряет свою "рюкзачную" компоненту и становится близка к классической NP-полной задаче о разбиении (partition problem) с тем лишь отличием, что в данном случае разбиение производится не на два, а на произвольное целое число (количество исполнителей) подмножеств. Итак, распределительная компонента формулируемой в настоящем проекте оптимизационной задачи сочетает в себе элементы двух известных сложных проблем: обобщенной задачи о назначениях и задачи о разбиении.

Обычно, в распределительных задачах рассматриваются условия аддитивности функции агрегирования затрат. Предполагаемая в данной статье постановка допускает использование различных функций стоимости в части оценки «качества» распределения.

2. Обозначения общего характера

Через $\stackrel{df}{=}$ обозначаем равенство по определению. Всюду в дальнейшем $\mathbb{N} \stackrel{df}{=} \{1; 2; ...\}$, $\mathbb{N}_0 \stackrel{df}{=} \mathbb{N} \cup \{0\}$, $\mathbb{R}$ - вещественная прямая.

Задано число $N \in \mathbb{N}$, $N \sim 2$, где $i \in \overline{1, N}$ - номера задач, или просто задачи. Кроме того, задано число $n \in \mathbb{N}$, $n \sim 2$, где $i \in \overline{1, n}$ - номера участников или просто участники.

Разбиение множества $\overline{1, N}$ на $n \in \mathbb{N}$ подмножеств будем называть всякую совокупность n непустых подмножеств $\{K_1, ..., K_n\}$ таких, что:

1. $\bigcup_{i=1}^{n} K_i = \overline{1, N}$;
2. $\forall i, j \ (i \neq j) \Rightarrow (K_i \cap K_j = \varnothing)$.

Множество таких разбиений будем обозначать $M_n(\overline{1, N})$.

Разбиение множества $K \subseteq \overline{1, N}$ на два подмножества K_1 и K_2 будем обозначать как $K = K_1 + K_2$.

Пусть задана функция стоимости выполнения отдельных заданий, $\tilde{d}:\overline{1,N}\to\mathbb{N}_0$. Введем аддитивную функцию агрегирования затрат при выполнении групп заданий:

$$\forall K\subseteq\overline{1,N}\quad d(K)=\sum_{i\in K}\tilde{d}(i).$$

Стоимостью распределения $\alpha=\{K_1,\dots,K_n\}\in M_n(\overline{1,N})$ называется значение функции $D:M_n(\overline{1,N})\to\mathbb{R}^+\cup\{0\}$, определяемое как $D(\alpha)=\max\limits_{i=\overline{1,n}}d(K_i)$.

Далее $\alpha^0\in M_n(\overline{1,N})$ называется оптимальным на $M_n(\overline{1,N})$, если $D(\alpha^0)=\min\limits_{\alpha\in M_n(\overline{1,N})}D(\alpha)$. В настоящей статье исследуется задача нахождения оптимального распределения заданного множества работ среди заданного числа исполнителей.

Для решения данной задачи по методу динамического программирования (МДП) (общие вопросы, связанные с МДП см. в [5,6]) следует построить ее естественное расширение [7-10], используя систему «укороченных» задач о разбиении неполных, вообще говоря, списков заданий между «частью» исполнителей. Получающиеся при этом экстремумы укороченных задач образуют систему функций, каждая из которых отвечает фиксированному набору исполнителей; эту систему логично рассматривать с позиции МДП как единую функцию Беллмана (подробнее см. в [7,10]), рассчитываемую рекурсивно по следующей схеме:

$$V_0(K)=d(K)\quad\forall K\subseteq\overline{1,N}$$

$$V_i(K)=\overset{\dots}{\min_{K_1+K_2=K}}\left(\max\{d(K_1),V_{i-1}(K_2)\}\right)\quad\forall K\subseteq\overline{1,N}$$

$$V_n(\overline{1,N})=\overset{\dots}{\min_{K_1+K_2=\overline{1,N}}}\left(\max\{d(K_1),V_{n-1}(K_2)\}\right)$$

Приведенный рекурсивный алгоритм обладает высокой вычислительной сложностью, поэтому помимо данной схемы точного решения, актуальны некоторые «быстрые» эвристические алгоритмы решения задачи распределения, которые были построены в [11,12].

3. Алгоритм

В данном разделе описывается алгоритм нахождения оптимального разбиения N – заданий, на n – участников (работников) при помощи МДП. На входе задано число работ $N\in\mathbb{N}, N\sim 2$, где $i\in\overline{1,N}$ - номера работ и число участников $n\in\mathbb{N}, n\sim 2$, где $i\in\overline{1,n}$ - номера участников. Кроме того, задано число процессоров $k\in\mathbb{N}, k\sim 2$, где $i\in\overline{1,k}$ - номер процессора, участвующего в вычислении.

Нам необходимо найти оптимальное распределение $\{K_1,\dots,K_n\}$ заданий $\overline{1,N}$, среди n работников, минимизируя работу выполняемую наиболее загруженным участником.

$$\max_{i=\overline{1,n}}\{D(K_i)\}\to\min_{M_n(\overline{1,N})}$$

1. На первом шаге нашего алгоритма, мы должны считать входные данные , такие как N – количество заданий, n – количество участников, $\tilde{d}(i)$ - стоимость выполнения i-ого задания, где $i\in\overline{1,N}$.
2. На втором шаге алгоритма выполняется построение слоев функции Беллмана.

- Первый слой.
  На первом слое функции Беллмана, мы выполним расчет стоимости каждого подмножества заданий $d(K)\ \forall K, K\subseteq\overline{1,N}$ одним работником. То есть, суммируем стоимости заданий, которые составляют данное подмножество.
- Второй слой.
  На данном этапе, для каждого подмножества $K\subseteq\overline{1,N}$ находим оптимальное на $M_2(\overline{1,N})$ распределение заданного подмножества на две группы заданий, соответствующие двум участникам. Иными словами, мы получаем такое распределение подмножеств, при котором работа выполняемая наиболее загруженным участником будем минимальна.
- Третий слой.
  На третьем слое функции Беллмана, мы находим оптимальное распределение каждого подмножества среди трех исполнителей. Для этого используется следующий алгоритм. Мы берем любое подмножество $K\subseteq\overline{1,N}$, выделяем в нем часть работ и отдаем одному из работников. А оптимальное распределение оставшейся части работ между двумя участниками нам уже известно (данную информацию мы получаем из вычислений, сделанных на предыдущем слое). Таким образом, мы перебираем всевозможные варианты, пока не получим

результат, при котором минимизируется работа наиболее загруженного участника.

- Аналогичным образом происходит построение слоев функции Беллмана для всякого $i \in \overline{4, n-1}$.
- Последний слой *(i=n)*.

  На *n*-том слое мы находим решение искомой задачи, то есть оптимальное разбиение множества заданий $\overline{1, N}$ на *n* участников. Для этого нам уже не нужно перебирать все имеющиеся подмножества, а необходимо взять только полное множество заданий $\overline{1, N}$. Для этого используется такой же метод, как и ранее, то есть мы берем какую-то часть заданий из множества $\overline{1, N}$, отдаем ее одному участнику, а оптимальное распределение оставшейся части заданий среди *n*-1 участников известно из предыдущего слоя. Так мы перебираем все варианты заданий, которые можно отдать одному участнику, и находим оптимальное распределение заданий множества $\overline{1, N}$ на *n* участников. Полученное значение является решением поставленной выше задачи распределения.

4. Параллельная реализация алгоритма

В данном разделе рассматривается реализация параллельного алгоритма для решения задачи нахождения оптимального распределения.

На первом шаге алгоритма нам необходимо разослать считанные начальные данные из корневого процессора, всем участникам вычислительного процесса. Для этого используется функция MPI_Bcast (см. Приложение 1).

На втором шаге алгоритма выполняется построение функции Беллмана. На данном этапе, на каждом слое, выполняется оптимальное распределение подмножеств $K \subseteq \overline{1, N}$. Распараллелим трудоемкий процесс перебора подмножеств. Для этого мы должны разделить подмножества $K \subseteq \overline{1, N}$ так, чтобы все процессоры были загружены равномерно. Каждый процессор, участвующий в вычислении, будет находить оптимальное разбиение выделенных ему подмножеств. Подмножества $K \subseteq \overline{1, N}$ упорядочены по возрастанию их мощности, то есть от одноэлементных до подмножества содержащего все элементы множества $\overline{1, N}$. Если мы разделим число подмножеств на количество заданий, и раздадим их поровну группами упорядоченными по возрастанию индексов, то первому процессору достанутся маломощные подмножества, в то время как последнему достанутся более трудоемкие задания. В такой ситуации первые процессоры закончат работу раньше последних, и будут вынуждены простаивать в ожидании обмена полученными результатами. Для того, чтобы избежать подобной ситуации, мы будем распределять подмножества на процессоры по «модулю *k*». Проиндексируем все подмножества $K \subseteq \overline{1, N}$ в порядке возрастания мощности. Поскольку количество процессоров равно *k*, первый процессор получит подмножества с индексами 1, 1+*k*, 1+2*k*, ... второй процессор получит подмножества 2, 2+*k*, 2+2*k*, ... и так далее. В результате такой процедуры, подмножества различной мощности, равномерно распределяются по процессорам. После того, как все процессоры выполнили вычисления на определенном слое, им необходимо обменятся полученными результатами, это необходимо для расчетов на следующем слое. Для этого используется функция MPI_Allgather (см. Приложение 1).

5. Оценка вычислительной сложности.

В настоящем разделе будет производиться оценка вычислительной сложности описанного выше алгоритма, как с реализацией параллельного алгоритма, так и без нее.

1. Оценка вычислительной сложности алгоритма без использования параллельной реализации.

- На шаге 1 алгоритма выполняется считывание начальных данных. Вычислительной сложностью данной процедуры можно пренебречь.
- На втором шаге алгоритма $\forall i \in \overline{1, n-1}$ идет построение слоев функции Беллмана, при этом выполняется полный перебор всевозможных разбиений $K = K_1 + K_2$ для каждого подмножества $K \subseteq \overline{1, N}$. Мощность $|K| = j$, где $j \in \overline{1, N}$. Количество подмножеств $K \subseteq \overline{1, N}$ мощности *j* равно $\frac{N!}{(N-j)! \cdot j!}$, количество операций по перебору данного подмножества равно 2^j, в результате $\forall i \in \overline{1, n-1}$ получаем оценку вычислительной сложности *i*-ого слоя
- $O_i\left(\sum_{j=0}^{N} \frac{N!}{(N-j)! \cdot j!} \cdot 2^j\right)$.
- Для $i = n$ выполняется перебор множества мощности $|K| = N$, получаем оценку вычислительной сложности $O_n(2^N)$.
- Итоговая вычислительная сложность алгоритма без использования параллельной

реализации получается путем сложения вычислительных сложностей каждого слоя $O = O_1 + O_2 + ... + O_n$:

- $$O\left((n-1)\cdot\sum_{j=0}^{N}\left(\frac{N!}{(N-j)!\cdot j!}\cdot 2^j\right) + 2^N\right).$$

2. Оценка вычислительной сложности алгоритма с использованием параллельной реализации.
   - На шаге 1 алгоритма выполняется считывание начальных данных и их пересылка процессорам, участвующих в вычислении. Вычислительной сложностью данной процедуры можно пренебречь.
   - На втором шаге алгоритма для $\forall i \in \overline{1, n-1}$, вычислительная сложность рассчитывается аналогично вычислительной сложности алгоритма без использования параллельной реализации. Учитывается, что расчет оптимального разбиения подмножеств выполняется на k процессорах, получаем вычислительную сложность i-ого слоя в виде:
   - $$O_i\left(\frac{\sum_{j=0}^{N}\frac{N!}{(N-j)!\cdot j!}\cdot 2^j}{k}\right).$$
   - Для $i = n$ выполняется перебор множества мощности $|K| = N$, данная процедура рассчитывается на одном процессоре, получаем оценку вычислительной сложности $O_n(2^N)$
   - Итоговая вычислительная сложность алгоритма с использованием параллельной реализации получается путем сложения вычислительных сложностей каждого слоя $O = O_1 + O_2 + ... + O_n$:
   - $$O\left(\frac{n-1}{k}\cdot\sum_{j=0}^{N}\left(\frac{N!}{(N-j)!\cdot j!}\cdot 2^j\right) + 2^N\right).$$

6. Вычислительный эксперимент.

В данном разделе рассматривается решение задачи, по нахождению оптимального разбиения N – заданий, на n – участников (работников) методом динамического программирования, выполненного с использованием МВС «Уран» и персонального компьютера. Также сравнивается время затраченного на решение задачи на МВС и на ПК.

Итак, расчеты будем производить для $n = 3,4,5$ работников, количество заданий N варьируется от 13 до 24. Стоимость заданий формируется всякий раз случайным образом, в интервале от 1 до 100: $\tilde{d}(i) \in \overline{1,100}$ $\forall i \in \overline{1, N}$. Для сравнения времени вычисления, на персональном компьютере и МВС, берутся идентичные начальные данные.

На основании ранее описанного алгоритма была написана программа для МВС "Уран" на языке программирования C++ с использованием библиотеки MPI [13]. Программа работает в среде 64-разрядной операционной системы семейства Linux. Вычислительный эксперимент проводился на супервычислителе ”Уран” с процессорной мощностью 1664 вычислительных ядра и 3584 Гбайт оперативной памяти. На данном МВС используются вычислительные модули на основе двухпроцессорных блейд-серверов HP ProLiant BL460c CTO Blade со следующими характеристиками:

- два 4-х ядерных процессора Intel® Xeon® E5450 (3.0 GHz, 1333 FSB,80W);
- кэш-память 2 x 6 MB Level 2 cache (5400 Sequence);
- оперативная память 2 GB PC2-5300, Registered DDR2-667 на ядро;
- сетевой адаптер Dual 1GbE NC326i plus (1) additional 10/100 NIC dedicated to iLO 2;
- контроллер дисков Embedded SATA;
- жесткие диски HDD 90GB Non-Hot Plug SFF SATA.

В нашем эксперименте использовались 64 вычислительных ядра.

Для сравнительного тестирования была написана программа на языке программирования C++ для ПЭВМ в основе которой, был заложен такой же алгоритм, но без параллельной структуры. Программа работает в среде 64-разрядной операционной системы семейства Windows, начиная с Windows XP. Вычислительный эксперимент проводился на персональном компьютере с центральным процессором Intel Core i5, объем ОЗУ 4 Гб с установленной операционной системой Windows 7 Professional.

Полученные результаты представлены в сравнительной таблице 1, для $n = 3,4,5$ и $N \in \overline{13,24}$.

n=3			*n*=4			*n*=5		
Таблица 1. Зависимость времени вычисления на ПК и МВС от количества заданий.*- время вычисления для данных размерностей, превосходит 48 часов.								
Кол-во заданий (*N*)	Время счета на ПК (сек.)	Время счета на МВС (сек.)	Кол-во заданий (*N*)	Время счета на ПК (сек.)	Время счета на МВС (сек.)	Кол-во заданий (*N*)	Время счета на ПК (сек.)	Время счета на МВС (сек.)
13	<1	1	13	1	1	13	1	1
14	1	1	14	2	2	14	2	2
15	4	1	15	4	3	15	6	3
16	8	2	16	14	4	16	19	4
17	27	2	17	48	5	17	67	9
18	94	5	18	167	11	18	237	15
19	336	18	19	596	34	19	860	46
20	1214	60	20	2160	112	20	3139	152
21	4436	192	21	8052	357	21	11559	520
22	16357	631	22	30597	1214	22	42960	1786
23	61278	2140	23	117839	4077	23	-*	6068
24	-*	7332	24	-*	13968	24	-*	21055

В качестве примера приводится результат одного из вычислительных экспериментов, выполненных на МВС «Уран». В данном примере количество работников $n=3,$ количество заданий $N=25.$ Стоимость выполнения заданий равна:

$$\tilde{d}(1)=47,\tilde{d}(2)=88,\tilde{d}(3)=95,\tilde{d}(4)=67,\tilde{d}(5)=45,\tilde{d}(6)=84,\tilde{d}(7)=22,\tilde{d}(8)=68,\tilde{d}(9)=22,$$
$$\tilde{d}(10)=85,\tilde{d}(11)=91,\tilde{d}(12)=100,\tilde{d}(13)=20,\tilde{d}(14)=9,\tilde{d}(15)=13,\tilde{d}(16)=96,\tilde{d}(17)=44,$$
$$\tilde{d}(18)=12,\tilde{d}(19)=8,\tilde{d}(20)=47,\tilde{d}(21)=11,\tilde{d}(22)=57,\tilde{d}(23)=52,\tilde{d}(24)=18,\tilde{d}(25)=17.$$

В результате выполнения программы, были получены следующие результаты:

Первый работник получил работы с индексами $i_1=1,3,4,5,6,8.$ Стоимость распределения равна $D(K_1)=406.$

Второй работник получил работы с индексами $i_2=2,7,10,11,12,13.$ Стоимость распределения равна $D(K_2)=406.$

Третий работник получил работы с индексами $i_3=9,14,15,16,17,18,19,20,21,22,23,24,25.$ Стоимость распределения равна $D(K_3)=406.$

Время счета программы составило 7 часов 4 минуты 11 секунд.

7. Заключение

В результате проделанной работы, был реализован алгоритм нахождения оптимального разбиения *N* – заданий, на *n* – участников (работников) при помощи метода динамического программирования, как с использованием, так и без использования параллельной структуры. Поставлен вычислительный эксперимент, сравнивающий эти два алгоритма. Полученные результаты показали, что алгоритм с использованием параллельной структуры, дает выигрыш по времени вычисления приблизительно в $k/2$ раз. Не смотря на то, что при использовании параллельной структуры, вычисления выполняются на *k* процессорах, последний слой вычисляется на одном (корневом) процессоре. Кроме того, при вычислении слоев функции Беллмана, процессоры обмениваются промежуточными данными, возникают накладные расходы по времени, что приводит к небольшому уменьшению эффекта от использования параллельного алгоритма.

Приложение 1.

1. Синтаксис MPI_Bcast задается следующим образом:

int MPI_Bcast (void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
где:
buffer - стартовый адрес буфера,
count - целое число, указывающее число элементов данных в буфере,
datatype - определенная константа MPI, указывающая тип элементов данных в буфере,
root - является целым числом, указывающим ранг корневого процесса широкого распространения,
comm - является коммуникатором.

- функция MPI_Bcast определяет коллективную операцию, и, тем самым, при выполнении необходимых рассылок данных вызов функции MPI_Bcast должен быть осуществлен всеми процессами указываемого коммуникатора;
- указываемый в функции MPI_Bcast буфер памяти имеет различное назначение у разных процессов: для процесса с рангом root, которым осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение, а для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных;

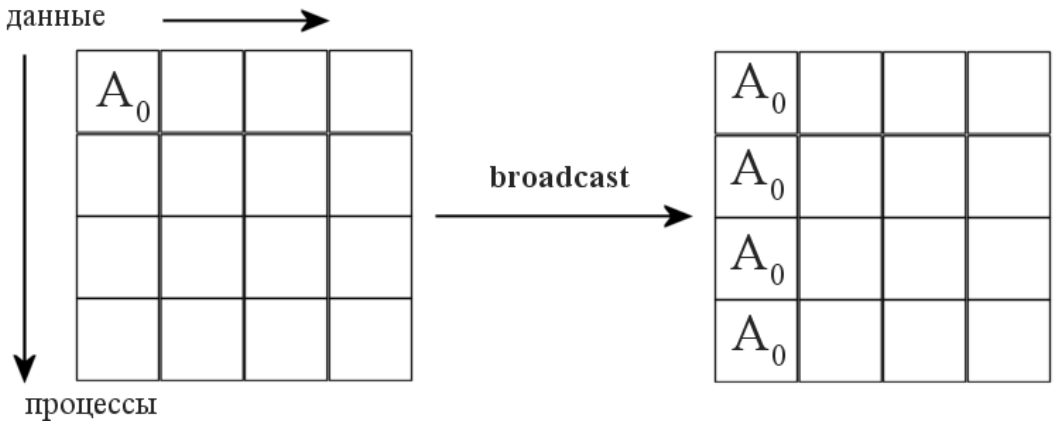

Рис.1.

2. Синтаксис MPI_Allgather задается следующим образом:

int MPI_Allgather (void* *sbuf,* int *scount,* MPI_Datatype *stype,* void* *rbuf,* int *rcount,* MPI_Datatype *rtype,* MPI_Comm *comm*)

Переменные функции Allgather:
sbuf - стартовый адрес буфера отправителя,
scount - количество элементов в буфере отправки,
stype - тип данных элементов буфера отправки,
rbuf - стартовый адрес буфера получателя,
rcount - количество элементов, полученных от произвольного процесса,
rtype - тип данных элементов буфера получателя,
comm - коммуникатор группы.

Функция MPI_Allgather выполняется так же, как MPI_Gather (данная функция производит сборку блоков данных, посылаемых всеми процессами группы, в один массив процесса с номером root. Длина блоков предполагается одинаковой. Объединение происходит в порядке увеличения номеров процессов-отправителей. То есть данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf процесса root. Длина массива, в который собираются данные, должна быть достаточной для их размещения.), но получателями являются все процессы группы. Данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf каждого процесса. После завершения операции содержимое буферов приема recvbuf у всех процессов одинаково.

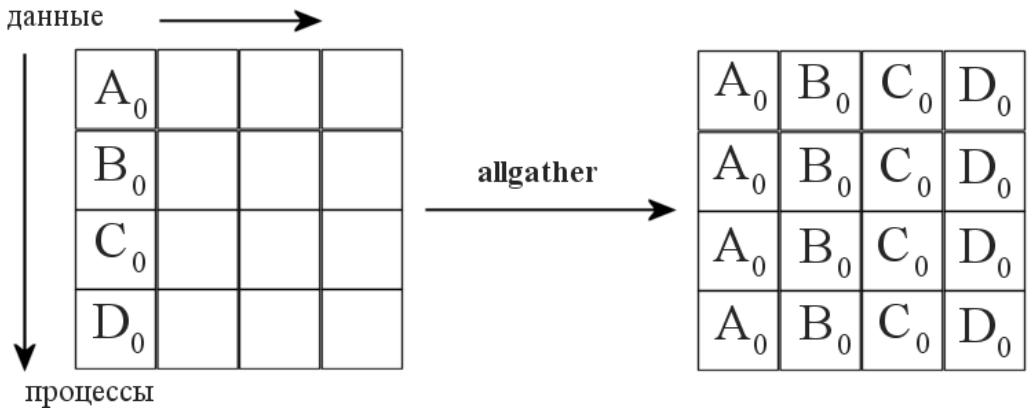

Рис.2.

ЛИТЕРАТУРА:

1. R.E. Burkard, M. Dell'Amico, S. Martello. Assignment Problems.- SIAM Philadelphia, 2009.
2. Harold W. Kuhn. The Hungarian Method for the assignment problem. Naval Research Logistics Quarterly, 1955, №2 P. 8397.
3. H. Kellerer, U. Pferschy, D. Pisinger. Knapsack Problems. Springer Verlag, 2005.
4. C. Papadimitriou, M. Yannakakis. Optimization, approximation and complexity classes. Journal of Computer and System Sciences, 1991. №43 P. 425440.
5. Беллман Р. Применение динамического программирования к задаче о коммивояжере / Кибернетический сборник. - М.: Мир, 1964. - Т. 9. - С. 219-228.
6. Беллман Р. Динамическое программирование. - М.: ИЛ, 1960. - 400 с.
7. Коротаева Л. Н., Назаров Э. М., Ченцов А. Г. Об одной задаче о назначениях // Журн. вычисл. математики и мат. физики. - 1993. - Т. 33, № 4. - С. 483-494.
8. Ченцов А. Г., Ченцов П. А. К вопросу о построении процедуры разбиения конечного множества на основе метода динамического программирования // Автоматика и телемеханика. - 2000. - № 4. - С. 129-142.
9. Ченцов А. Г., Ченцов П. А. Динамическое программирование в задаче оптимизации разбиений // Автоматика и телемеханика. - 2002. - № 5. - С. 133-146.
10. Ченцов А.Г. Экстремальные задачи маршрутизации и распределения заданий: вопросы теории. М.; Ижевск: НИЦ “Регулярная и хаотическая динамика”, Ижевский институт компьютерных исследований. 2008. 240 с.
11. Ченцов П. А. О некоторых алгоритмах распределения заданий между участниками // Алгоритмы и програм. средства параллел. вычислений. - Екатеринбург: УрО РАН, 2006. - Вып.9. - С. 234-247.
12. Ченцов П. А. О некоторых алгоритмах распределения заданий между участниками // Автоматика и телемеханика — 2006. - № 8. - С. 77-91.
13. Антонов А. С. Параллельное программирование с использованием технологии MPI: Учебное пособие. - М.: Изд-во МГУ, 2004. - 71 с.