УДК 519.6

В.А. Богаенко

Об оптимизации вычислительного процесса математического моделирования сложных задач

Рассмотрена задача оптимизации вычислительного процесса при решении на кластерных системах сложных задач математического моделирования, допускающих декомпозицию на слабосвязанные подзадачи. Предложены модель вычислительной среды, методика оценки времени выполнения задач и эвристические алгоритмы решения задач оптимизации.

A computing process optimization problem while solving on cluster systems the complex problems of mathematical modelling, which can be decomposed on loosely coupled subproblems, is considered. A omputing environment model, method of problem solving time estimation and heuristic algorithms for solving optimization problem are suggested.

Розглянуто задачу оптимізації обчислювального процесу при розв'язанні на кластерних системах складних задач математичного моделювання, що допускають декомпозицію на слабкозв'язані підзадачі. Запропоновано модель обчислювального середовища, методику оцінки часу виконання задач та евристичні алгоритми розв'язання задач оптимізації.

Введение. Задача оптимизации вычислительного процесса возникает во многих областях информационных технологий, в частности, как задача оптимизации кода при построении компиляторов [1] или задача оптимизации процесса исполнения запроса к распределенной базе данных [2].

Возникает такая проблема и в математическом моделировании при решении сложных задач, допускающих декомпозицию на слабосвязанные подзадачи. При их решении возникают две проблемы – синхронизация решения подзадач по времени и оптимизация схемы вычислений.

Существуют два основных подхода к решению задачи синхронизации, влияющего в первую очередь на точность вычислений. Первый [3] состоит в разбиении системы дифференциальных уравнений, решаемой в нескольких масштабах времени, на подсистемы, каждая из которых решается в одном масштабе времени. Такой подход не есть универсальным, но позволяет получать достаточно точные решения и оценки погрешностей.

Второй подход – алгоритмический [4], рассматривает каждую подзадачу как «черный ящик», т.е. только как процесс, который в определенные моменты, обусловленные масштабом времени выполнения моделирования, обменивается данными с другими процессами. Поскольку этот подход не учитывает математической сути решаемых подзадач, при его использовании невозможно в общем случае получить оценки погрешностей. По этой же причине, которая наделяет этот подход свойством универсальности, точность решений будет меньшей, в сравнении с получаемой при использовании первого подхода.

Задача оптимизации процесса решения состоит в нахождении «самой быстрой» конфигурации вычислительного процесса исходя из имеющихся алгоритмов решения подзадач и ресурсов, таких как процессоры (ядра) в кластерной системе. В случае математического моделирования она имеет много общего с подобной задачей, возникающей при построении систем распределенных баз данных [2].

Вычислительный процесс на первом этапе как в одном, так и в другом случае можно представить в виде направленного графа, в котором узлы обозначают подзадачи, а дуги – зависимости между ними. Такой граф является моделью вычислительного процесса и для того, чтобы стать алгоритмом, должен быть конкретизирован: узлы-подзадачи должны быть заменены узлами-алгоритмами решения подзадач, а дуги-зависимости (в общем случае) – на дуги-зависимости по данным. Но этот процесс не однозначен: для одного графа-модели может существовать большое количество графов-алгоритмов. К тому же возможны операции эквивалентного преобразования графов-моделей.

Можно построить набор правил преобразования, применяя которые к определенному начальному графу-алгоритму (далее просто графу) получаем все множество возможных гра-

фов, т.е. построить порождающую грамматику, в общем случае – неограниченную (тип 0 по классификации Хомского). На множестве графов, порожденных этой грамматикой, ставится оптимизационная задача нахождения лучшего графа по критерию быстродействия.

Таким образом, система оптимизации должна содержать следующие составляющие:

- целевую функцию: подсистему оценки времени решения задачи согласно вычислительной схемы, представленной графом;
- множество правил преобразования графов;
- алгоритм решения оптимизационной задачи.

Модель вычислительной среды и оценка времени решения

Модель вычислительной среды необходима для формулирования правил преобразования и оценки времени решения. При численном решении задач математического моделирования на кластерных системах среду можно представить как множество процессоров, на которых выполняются вычислительные задачи. Вариантов параллелизма может быть два: «внутриподзадачный» (как аналог внутризапросного или внутритранзакционного параллелизма в базах данных) и «межподзадачный» (как аналог межзапросного или межтранзакционного). Первый – это наличие параллельного алгоритма численного решения подзадачи, второй – решение двух или более подзадач параллельно, каждая на отдельной вычислительной подсистеме (подмножестве общего множества процессоров).

В такой модели каждый узел графа имеет три характеристики:

- метод решения подзадачи;
- вычислительную подсистему, на которой подзадача будет решаться;
- алгоритм: последовательный или параллельный.

Каждая же дуга – зависимость по данным характеризуется объемом данных, которыми процессы, решающие подзадачи, должны обмениваться.

Обмены эти происходят в трех случаях:

- когда зависимые одна от другой подзадачи решаются на разных подсистемах, т.е. на разных процессорах, что приводит независимо от других обстоятельств к необходимости проведения обменов;
- когда одна из подзадач решается последовательным, а другая – параллельным алгоритмом (параллельный алгоритм сохраняет данные распределенными по вычислительной подсистеме, тогда как последовательный – сосредоточенными на одном узле, соответственно, в этой ситуации должна выполняться операция «сбора» или «распределения» данных);
- когда подзадачи решаются параллельными алгоритмами, использующими разные схемы распределения данных (необходима операция «перераспределения»).

Поскольку количество процессоров в подсистемах не фиксируется, а лишь ограничивается общим количеством процессоров, при оценке времени работы графа алгоритма возникает дополнительная оптимизационная задача – задача выбора оптимального количества процессоров и оптимального их распределения по подсистемам:

$$\vec{N}_m = (n_1, ..., n_m),\ n_i \geq 1,\ \sum_{i=1}^{m} n_i \leq P,$$

$$\vec{N}_{opt} = \min_{m, \vec{N}_m} T(\vec{N}_m),$$

где m – количество подсистем, P – общее количество процессоров, $T(\vec{N}_m)$ – оценка времени работы графа алгоритма при заданном распределении процессоров по подсистемам ($\vec{N}_m$). Эта задача может решаться градиентными методами.

Оценка времени работы графа алгоритма при заданном распределении процессоров может проводиться путем симуляции вычислительного процесса, используя на каждом шаге оценку времени работы конкретного алгоритма решения подзадачи, времени выполнения обменов данными между подзадачами и информации о необходимости синхронизации работы подсистем. Оценку времени можно рекурсивно представить следующим образом:

$$T = \max_s T_s,\ T_s = \max_j \{T_{sj}, S_j = s, e_j = \varnothing\},$$

$$T_{si} = T_i^c + \sum_{j, e_{ji}=1} T_{ij}^x + \max_k \{T_{S_k k}, e_{ki} = 1\},\ S_i = s,$$

где T – общая оценка времени работы графа алгоритма, T_s – общая оценка времени работы подсистемы s, T_{si} – оценка времени работы подсистемы s после решения подзадачи i, S_j – подсистема, которой принадлежит подзадача j, e_j – множество подзадач, зависящих от решения подзадачи j, e_{ij} – элемент матрицы связности графа: $e_{ij} = 1$ в случае, когда подзадача i зависит от подзадачи j, T_i^c – оценка времени исполнения подзадачи i, T_{ij}^x – оценка времени обмена данными при синхронизации между подзадачами i и j.

Отметим, что оценки зависят от аппаратной платформы. Интегральным показателем здесь может быть отношение быстродействия вычислительных узлов к скорости обмена данными между ними $\left(\frac{t_*}{t_s}\right)$. Значение этого показателя может существенно влиять как на оптимальное количество задействованных узлов при решении подзадачи параллельным алгоритмом, так и в целом на целесообразность использования конкретных параллельных алгоритмов.

Правила преобразования графов алгоритмов

Исходя из приведенной модели вычислительной среды, можно сформулировать следующие правила преобразования графов алгоритмов, которые можно разбить на две группы. Правила из первой (1–4) применяются к конкретному узлу и влияют только на его характеристики.

Вторая группа (5–6) изменяет структуру графа:

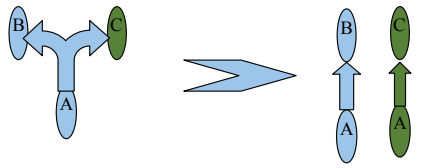
1. Изменение метода решения подзадачи.

2. Замена алгоритма решения подзадачи (переход от последовательного к параллельному, или наоборот).

3. Изменение подсистемы, на которой решается подзадача. Удаление подсистемы в случае, когда, после применения правила, на ней не решается ни одна задача.

4. Создание дополнительной подсистемы, на которой должна решаться подзадача.

5. Дубляж по подсистемам выглядит так:



Разными цветами обозначены подсистемы, на которых решаются соответствующие подзадачи. Смысл этого преобразования состоит в том, чтобы в случае, когда несколько подзадач (B и C) зависимы от подзадачи A, причем эти подзадачи решаются на разных подсистемах, дублировать решение подзадачи A на всех подсистемах, где решаются зависимые от нее подзадачи. Такое преобразование дает возможность избавиться от затратной операции обмена данными между подзадачами.

6. Обратное к предыдущему правило объединения дублированных в разных подсистемах подзадач.

Алгоритмы оптимизации

Множество графов, которые генерируются из определенного исходного графа применением этих правил, можно, в случае исключения идентичных графов, представить в виде дерева, которое является конечным, так как размерность множества графов зависит от количества узлов в исходном графе. Каждый граф из множества может быть представлен как последовательность конечной длины применения правил преобразования (в том числе «пустого» правила) к исходному графу. Такое представление позволяет использовать для поиска оптимального графа методы комбинаторной оптимизации или же метаэвристические методы, такие как генетические алгоритмы или роевой интеллект. Для небольших задач применимым и полный или ограниченный некоторым числом уровней дерева перебор.

Набор правил можно ограничить, оставив в нем только правила 3, 4 и 5. При этом считается, что все алгоритмы параллельные и предварительно был выбран лучший метод решения каждой подзадачи. Правило 6 отбрасывается как обратное к правилу 5, не ограничивая пространство поиска, а на правило 3 можно наложить ограничение относительно того, что перенос подзадачи в другую подсистему не должен уменьшать общее количество подсистем (при уменьшении числа подсистем правило 3 будет обратным к правилу 4).

В таком ограниченном наборе правил, полный перебор возможных графов решения задачи (в дальнейшем, вариантов) можно разбить на два этапа:

- перебор всех вариантов, продуцируемых применением только правила 5;
- перебор вариантов, получаемых при применении правил 3 и 4 для каждого графа, полученного на первом этапе. Этот перебор можно проводить, последовательно рассматривая все варианты с фиксированным количеством подсистем.

Таким образом, на первом этапе рассматриваются возможные изменения самого графа, тогда как на втором – его отображение на множество подсистем.

Сложность целевой функции не позволяет аналитически сузить пространство поиска, что обуславливает использование эвристических и метаэвристических методов приближенного решения оптимизационной задачи.

Алгоритм ограниченного поиска

Алгоритм ограниченного поиска можно построить, проводя полный перебор на первом этапе приведенного алгоритма, а на втором ограничивать поиск с помощью следующих эвристических правил:

- если оценка времени работы наилучшего варианта с N подсистемами большая, чем оценка времени работы наилучшего варианта с $N-1$ подсистемами, то и для числа подсистем $M > N$ она также будет большей;
- поиск локального минимума: от некоторого начального варианта последовательно выполняется переход к варианту из некоторой окрестности текущего, имеющего наименьшую оценку времени работы. Поиск заканчивается, когда найден локальный минимум. Множество вариантов, формирующих окрестность текущего, можно строить исключительно как множество всех возможных результатов однократного применения к текущему варианту правил грамматики или же с учетом особенностей задачи. Если «цепочка» зависимых одна от другой подзадач отображена на одну подсистему, это минимизирует количество обменов и синхронизаций между ними. Соответственно, окрестностью варианта можно считать множество, в котором для определенного узла, находящегося в некоторой подсистеме, содержатся все варианты с «цепочками», проходящими через этот узел.

Реализованный эвристический алгоритм ограниченного поиска можно описать следующим образом:

Генерация полного множества S_5 вариантов преобразований, применяя правило 5;
foreach $v_5 \in S_5$
{
for (*l* = 1;*i*<количество узлов в v_5 ;*i*++)
{
Генерация начального варианта с разбитием множества процессоров на *i* подсистем;
*s*1: Поиск локального минимума, начиная из текущего «наилучшего» варианта;
*s*2: Поиск локального минимума, начиная от «наилучшего» варианта из окрестности текущего «наилучшего» варианта;
Если решение, найденное на этапе *s*2, «лучше», чем решение, найденное на этапе *s*1, переходим к этапу *s*1;
Если решение, найденное на текущей итерации цикла, «хуже» чем решение, найденное на предыдущей итерации, то выходим из цикла;
}
Сохранение «наилучшего» решения для варианта v_5;
}
Нахождение «наилучшего» решения среди оптимальных решений для вариантов из множества S_5;

Недостаток методов ограниченного поиска для рассматриваемой задачи заключается в том, что для первого этапа их работы, построить адекватные эвристические правила очень сложно.

Метаэвристический алгоритм поиска

Метаэвристический алгоритм поиска можно построить, отталкиваясь от алгоритма муравьиных колоний [5, 6]. Введем ограничение на правило 3 такое, что изменение подсистемы разрешается только в направление увеличения ее номера. С учетом такого ограничения, пространство поиска можно представить в виде направленного графа с одним начальным узлом и одним финальным узлом, для которого дальнейшие преобразования невозможны. Последовательность преобразований, которые из начального варианта приводят к финальному, назовем путем, длинной которого будем считать наименьшую оценку времени работы для входящих в него вариантов. В предлагаемом алгоритме используется последовательное случайное генерирование путей с учетом определенного количества наилучших предыдущих путей (аналог распыления и испарения феромона в алгоритме муравьиных колоний) при определении вероятности каждого перехода (аналог действия феромона). В качестве критерия завершения поиска можно взять следующий: алгоритм завершает поиск, когда в множестве наилучших путей наличествует определенный процент одинаковых вариантов.

Реализованный метаэвристический алгоритм можно описать следующим образом:

Инициализация
do
{
Случайный выбор начального варианта генерируемого пути: или как начального узла множества графов, или как одного из вариантов в множестве наилучших путей;
do
{
– При возможности, с заданной вероятностью, применение к текущему варианту одного из преобразований, присутствующих в множестве наилучших путей;
– Случайный выбор следующего варианта из окрестности текущего;
– Сохранение выбранного варианта;
}
while (последний вариант в пути – финальный);
– В случае, если построенный путь «лучший», чем «наихудший» путь множества наилучших путей, вытеснить «наихудший» путь из множества;
}
while (количество одинаковых вариантов в множестве наилучших путей меньше, чем заданный процент от общего числа наилучших путей);

Результаты решения тестовых задач

Результаты решения тестовых оптимизационных задач представлены на рис. 1–6. Узлы, обозначенные черным цветом, – входные и выходные параметры подзадач, требующие синхронизации, а узлы, обозначенные другими цветами, – собственно подзадачи. Цветом в этом случае обозначается подсистема, в рамках которой решается подзадача. Данные про эффективность и быстродействие алгоритмов поданы в табл. 1–3.

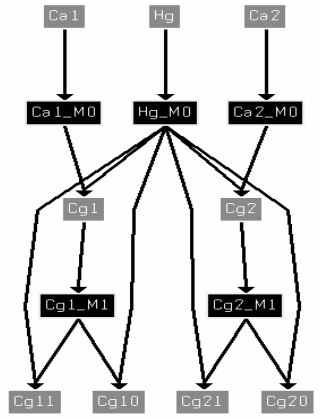


Рис. 1. Начальный вариант для задачи № 1

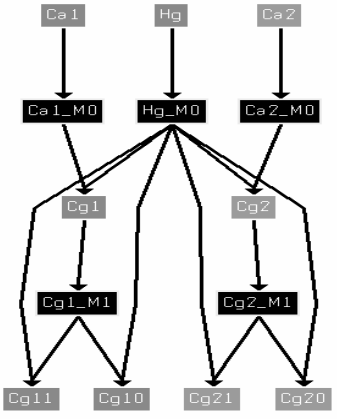
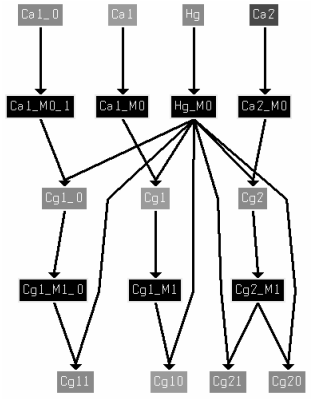


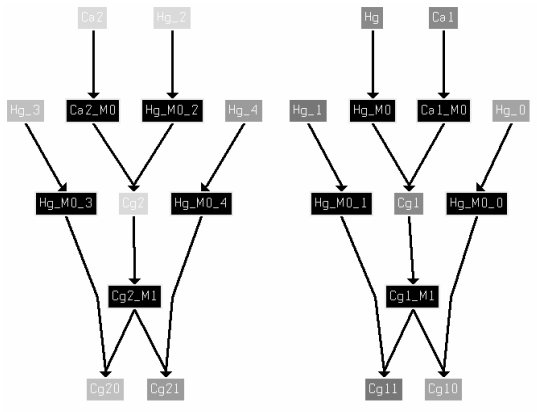
Рис. 2. Результат оптимизации при $\frac{t_*}{t_s}=1000$

Заключение. Из полученных результатов вычислительных экспериментов делаем следующие выводы:

• Использование процедур оптимизации вычислительного процесса позволяет, согласно теоретическим оценкам, ускорить, в некоторых случаях вдвое, решение сложных задач моделирования физических процессов.



а



б

Рис. 3. Результаты оптимизации при $\frac{t_*}{t_s}$=10: *а* – при P = 128; *б* – при P = 500

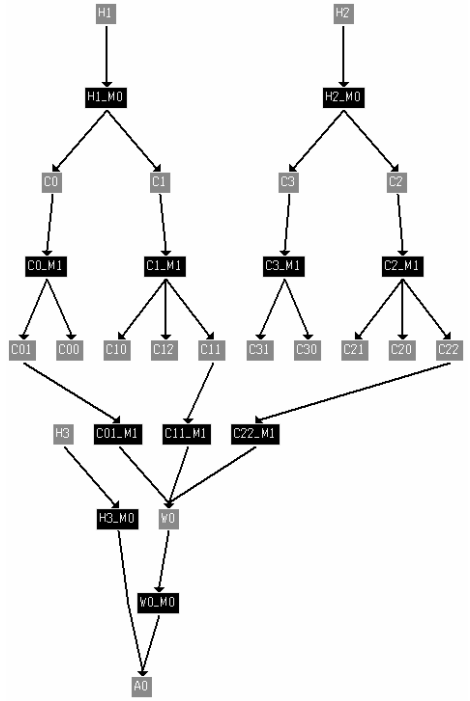
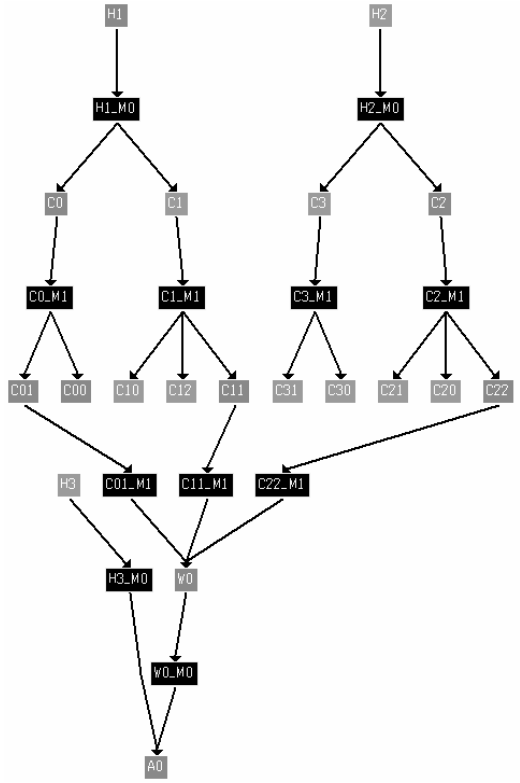
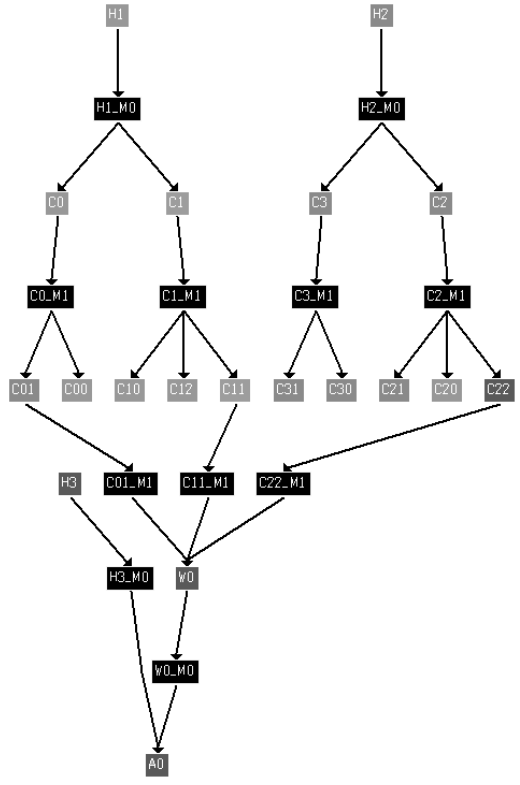


Рис. 4. Начальный вариант для задачи № 2

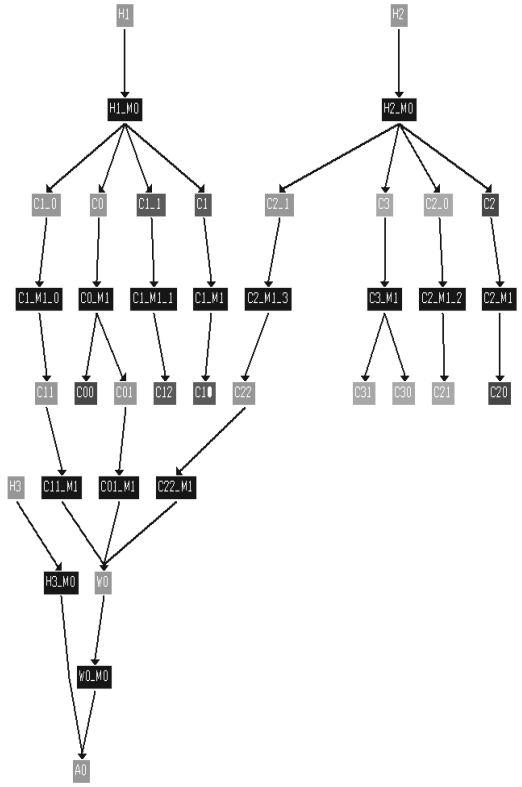


а

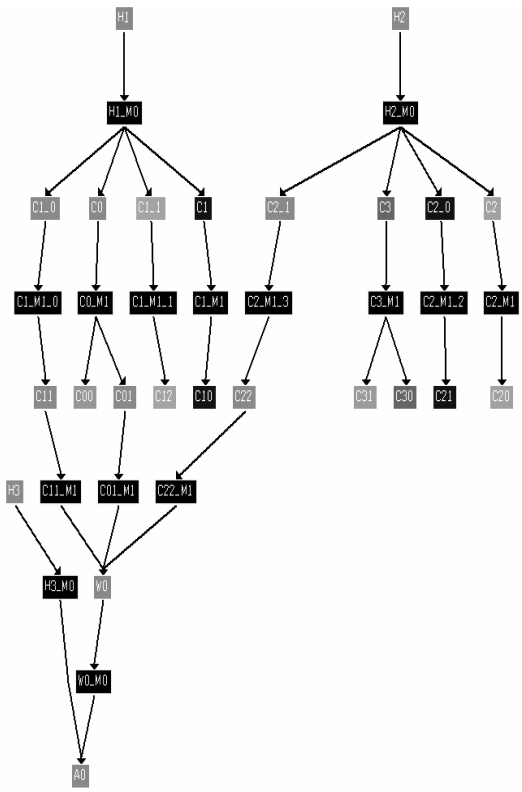


б

Рис. 5. Результаты оптимизации при $\frac{t_*}{t_s}$ = 1000: *а* – при P = 128; *б* – при P = 500



а



б

Рис. 6. Результаты оптимизации при $\frac{t_*}{t_s} = 10$: *а* – при $P = 128$; *б* – при $P = 500$

Таблица 1. Эффективность оптимизационных процедур для задачи 1

	Начальная оценка времени работы	Оценка времени работы оптимизированной схемы	Ускорение	Количество задействованных процессоров в оптимизированной схеме
$\frac{t_*}{t_s} = 1000$; $P = 128$	130928	69147,5	47%	93
$\frac{t_*}{t_s} = 1000$; $P = 500$	130928	67147	49%	186
$\frac{t_*}{t_s} = 10$; $P = 128$	1881630	1697040	10%	128
$\frac{t_*}{t_s} = 10$; $P = 500$	1308240	682000	48%	498

Таблица 2. Эффективность оптимизационных процедур для задачи 2

	Начальная оценка времени работы	Оценка времени работы оптимизированной схемы	Ускорение	Количество задействованных процессоров в оптимизированной схеме
$\frac{t_*}{t_s} = 1000$; $P = 128$	759000	432208	43%	124
$\frac{t_*}{t_s} = 1000$; $P = 500$	759000	432208	43%	217
$\frac{t_*}{t_s} = 10$; $P = 128$	10908000	10393700	5%	128
$\frac{t_*}{t_s} = 10$; $P = 500$	7584000	3778670	50%	498

Таблица 3. Время работы оптимизационных алгоритмов в секундах

	Задача 1 9 подзадач, 12 зависимостей	Задача 2 19 подзадач, 19 зависимостей
Алгоритм ограниченного поиска	12,8	639,5
Алгоритм ограниченного поиска с расширенным определением окрестности	48,4	6091,5
Метаэвристический алгоритм	11,4	269,7

Окончание статьи на стр. 24