КОМПЬЮТЕРНЫЕ НАУКИ 2017. Т. 27. Вып. 1

УДК 519.6

© *А. М. Григорьев*

РЕШЕНИЕ ЗАДАЧИ ОБ ОПТИМАЛЬНОМ РАСПРЕДЕЛЕНИИ ЗАДАНИЙ МЕТОДОМ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ С ПРИМЕНЕНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ[1]

Исследование посвящено построению параллельного алгоритма решения задачи «на узкие места», связанного с поиском разбиения конечного множества заданий на конечное число исполнителей (работников). Описывается алгоритм нахождения оптимального разбиения заданий с использованием метода динамического программирования с элементами параллельных вычислений при построении массива значений функции Беллмана. Выполнена оценка вычислительной сложности двух алгоритмов (с использованием и без использования параллельной структуры). Создана программа, с помощью которой проведен вычислительный эксперимент по решению поставленной задачи на суперкомпьютере «УРАН». Выполнен сравнительный анализ реализации алгоритмов как с использованием, так и без использования параллельной структуры. Представлена зависимость времени счета реализованной программы на суперкомпьютере от количества вычислительных ядер.

Ключевые слова: динамическое программирование, разбиение, параллельный алгоритм.

DOI: 10.20537/vm170111

Введение

Современный высокотехнологический 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.

Среди возможных вариантов снижения дозовых нагрузок облучения персонала АЭС, подлежащих оптимизации, несомненный интерес представляет оптимизация дозовых затрат, получаемых персоналом при перемещении от объекта к объекту, поскольку данный вопрос практически не рассматривался в отечественной и зарубежной практике. Снижение «транзитных доз», полученных в пути на рабочее место и от рабочего места до выхода из зоны контролируемого доступа, представляет собой важную задачу в общем процессе оптимизации облучения ремонтного персонала. Для сокращения доз, получаемых при перемещении, на ряде зарубежных станций используются подробные карты, которые можно получить при входе в реакторное здание и на различные отметки внутри здания. Однако, по имеющимся данным, на предприятиях атомной энергетики и промышленности не используется оптимизация пути перемещения работников с целью сокращения доз, получаемых при перемещении [1]. При выполнении ремонтных работ выбор маршрута перемещения с минимальной дозой облучения не представляет сложности, так как количество обслуживаемых объектов ограничено одним-двумя. При обслуживании значительного количества объектов в радиационно опасных зонах требуется использование эффективных вычислительных программ, так как число возможных маршрутов перемещения составляет $N!$, где N — количество посещаемых объектов.

[1] Работа выполнена при финансовой поддержке РФФИ (грант 16–01–00649).

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) [2], успешно решаемая с помощью методов линейного программирования за полиномиальное время (см., например, венгерский алгоритм) [3]. Обобщенная постановка задачи о назначениях [4], 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 [5], 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) в оптимизационной постановке, с тем лишь отличием, что в данном случае разбиение производится не на два подмножества, а на произвольное целое число (количество исполнителей) подмножеств. Итак, распределительная компонента формулируемой в настоящей статье оптимизационной задачи сочетает в себе элементы двух известных сложных проблем: обобщенной задачи о назначениях и задачи о разбиении.

Обычно в задачах распределения заданий рассматриваются аддитивные функции агрегирования затрат. В данной работе предполагаемая постановка допускает использование произвольных функций оценки «качества» распределения.

В связи с тем, что рассматриваемая задача распределения является трудоемкой в вычислительном отношении (NP-трудной), появляется необходимость разработки параллельных алгоритмов для ее решения. В данной статье описывается параллельный алгоритм нахождения оптимального разбиения N заданий на n участников (работников) при помощи метода динамического программирования (МДП).

§ 1. Формальная постановка задачи

Обозначаем равенство по определению как $\stackrel{\triangle}{=}$. Всюду в дальнейшем $\mathbb{N} \stackrel{\triangle}{=} \{1; 2; \ldots\}$, $\mathbb{N}_0 \stackrel{\triangle}{=} \mathbb{N} \cup \{0\}$, $\mathbb{R}$ — вещественная прямая. Пусть задано число $N \in \mathbb{N}$ $(N \geqslant 2)$ — количество заданий; $i \in \overline{1, N}$ — номера заданий. Кроме того, задано число $n \in \mathbb{N}$ $(2 \leqslant n \leqslant N)$ — количество участников; $i \in \overline{1, n}$ — номера участников (работников).

Итак, предполагается, что у нас имеется N заданий и n работников (исполнителей). Требуется распределить все задания между работниками. При этом предполагается, что каждое задание выделяется ровно одному участнику.

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

1) $\bigcup\limits_{i=1}^{n} K_i = \overline{1, N}$;

2) $\forall i, j \ (i \neq j) \Rightarrow (K_i \cap K_j = \varnothing)$.

Множество таких разбиений будем обозначать как $M_n(\overline{1, N})$. Разбиение множества $K \subseteq \overline{1, N}$ на два подмножества K_1 и K_2 будем обозначать как $K = K_1 \bigsqcup K_2$. Пусть задана функция

стоимости выполнения групп заданий $D : \mathcal{P}(\overline{1,N}) \to \mathbb{R}$, где $\mathcal{P}(X)$ — это традиционно множество всех подмножеств множества X.

Стоимостью распределения $\alpha = (K_1, \ldots, K_n) \in M_n(\overline{1,N})$ называется значение функции $\mathcal{D} : M_n(\overline{1,N}) \to \mathbb{R}^+$, определяемое как $\mathcal{D}(\alpha) = \max\limits_{i=\overline{1,n}}\{D(K_i)\}$. Далее, $\alpha^0 \in M_n(\overline{1,N})$ называется оптимальным на $M_n(\overline{1,N})$, если $\alpha^0 \in \operatorname{argmin}\limits_{\alpha \in M_n(\overline{1,N})} \mathcal{D}(\alpha)$. В настоящей статье исследуется задача нахождения оптимального распределения заданного множества работ среди заданного числа исполнителей с помощью параллельных вычислений:

$$V = \max_{i=\overline{1,N}}\{D(K_i)\} \to \min, \quad \bigsqcup_{i=1}^{n} K_i = \overline{1,N}. \tag{1}$$

Строгая постановка имеет вид

$$V = \max_{i=\overline{1,n}}\{D(K_i)\} \to \min, \quad (K_i)_{i\in\overline{1,n}} \in M_n(\overline{1,N}).$$

§ 2. Метод динамического программирования

В статье рассматривается схема динамического программирования [6–9] (подробное изложение см. в [8, часть 5]). Предлагаемый ниже алгоритм содержит особенности, связанные с использованием параллельной структуры для расчета на супервычислителе (см. § 3).

Алгоритм.

1. Инициализация функции Беллмана (первый слой): для всякого подмножества $K \subseteq \overline{1,N}$ выполняем $V_1(K) := D(K)$.

2. Последовательно увеличивая i от 2 до $n-1$, выполняем расчет промежуточных значений функции Беллмана, опираясь на i-м шаге на результат вычислений, полученный на предыдущем шаге. Для всякого $K \subseteq \overline{1,N}$ $\quad V_i(K) := \min\limits_{\widetilde{K}\subseteq K} \Big\{\max\{D(\widetilde{K}), V_{i-1}(K \setminus \widetilde{K})\}\Big\}$.

3. Вычисляем последнее значение функции Беллмана, совпадающее со значением (экстремумом) задачи (1):

$$V_n(\overline{1,N}) := \min_{\widetilde{K}\subseteq\overline{1,N}} \Big\{\max\{D(\widetilde{K}), V_{n-1}(\overline{1,N} \setminus \widetilde{K})\}\Big\}.$$

Рассмотрим краткое доказательство корректности алгоритма (см. более подробно [6–8]).

Утверждение 1. *Значение $V_m(K)$, вычисляемое в алгоритме, есть решение задачи оптимального разбиения множества K среди m исполнителей.*

Доказательство. Покажем индукцией по числу исполнителей m, что

$$V_m(K) \leqslant \min_{\bigsqcup_{i=1}^{m} K_i = K} \left\{\max_{i=\overline{1,m}}\{D(K_i)\}\right\}.$$

База индукции: $m = 1$. Очевидно, так как $V_1(\overline{1,N}) = D(\overline{1,N})$.

Шаг индукции: пусть утверждение доказано для $\forall i < m$; докажем утверждение для $i = m$. Для любого распределения $\{L_1, \ldots, L_m\} : \bigsqcup\limits_{j=1}^{m} L_j = K$ по определению

$$\max_{i=\overline{1,m}}\{D(L_i)\} = \max\Big\{D(L_m), \max_{i=\overline{1,m-1}}\{L_1, \ldots, L_{m-1}\}\Big\} \geqslant \max\left\{D(L_m), \left\{V_{m-1}\left(\bigcup_{k=1}^{m-1} L_k\right)\right\}\right\} \geqslant$$

$$\geqslant \min_{L\subseteq K}\{\max\{D(L), V_{m-1}(K \setminus L)\}\} = V_m(K).$$

Очевидно, стоимость $V_m(K)$ достигается на разбиении $\{K_1^0, \ldots, K_m^0\}$, получающемся при движении «вверх» по алгоритму:

$$K_m^0 : V_m(K) = \max\{D(K_m^0), V_{m-1}(K \setminus K_m^0)\},$$

$$K_{m-1}^0 : V_{m-1}(K \setminus K_m^0) = \max\left\{D(K_{m-1}^0), V_{m-2}\left((K \setminus K_m^0) \setminus K_{m-1}^0\right)\right\}$$

и так далее, до K_1^0.

В итоге имеем $V_m(K) = \max\{D(K_m^0), D(K_{m-1}^0), \ldots, D(K_1^0)\} = \mathcal{D}((K_1^0, \ldots, K_m^0))$. □

Общие вопросы теории динамического программирования рассматривались в [10] (см. также подробное изложение абстрактных вариантов МДП в [11]).

Приведенный рекурсивный алгоритм обладает высокой вычислительной сложностью, поэтому, помимо данной схемы точного решения, актуальны «быстрые» эвристические алгоритмы решения задачи распределения, которые были построены, например, в [9].

В следующем параграфе рассматривается параллельный алгоритм построения точного решения этой задачи, реализованный на суперкомпьютере. В настоящей работе конструируется алгоритм, реализованный на суперкомпьютере, что отличает ее от [6–8,12], где использовались подобные алгоритмы для решения на ПК. Результаты настоящей работы анонсированы в [13].

§ 3. Параллельная реализация алгоритма

В данном параграфе рассматривается параллельная реализация алгоритма для решения задачи нахождения оптимального распределения.

В наших вычислениях участвует k процессоров.

- На первом шаге алгоритма корневой процессор рассылает начальные данные всем участникам вычислительного процесса. Для этого используется функция MPI Bcast.

- На втором шаге алгоритма выполняется построение функции Беллмана:

$$V_i(K) := \min_{\widetilde{K} \subseteq K} \left\{\max\{D(\widetilde{K}), V_{i-1}(K \setminus \widetilde{K})\}\right\}.$$

  На данном этапе для каждого слоя выполняется оптимальное распределение подмножеств $K \subseteq \overline{1, N}$. Распараллелим наиболее трудоемкий процесс перебора подмножеств. Для этого необходимо распределить подмножества $K \subseteq \overline{1, N}$ так, чтобы все процессоры были загружены равномерно. Каждый процессор, участвующий в вычислении, находит оптимальное разбиение любого из выделенных ему подмножеств. Подмножества $K \subseteq \overline{1, N}$ упорядочиваем по возрастанию их мощности: от одноэлементных до подмножества, содержащего все элементы множества $\overline{1, N}$. Если мы разделим число подмножеств на количество процессоров и раздадим их поровну группами, упорядоченными по возрастанию индексов, то первому процессору достанутся подмножества малой мощности, 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, 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 $1, 1 + k, 1 + 2k, \ldots$, второй процессор получит подмножества $2, 2 + k, 2 + 2k, \ldots$ и так далее. В результате такой процедуры подмножества различной мощности относительно равномерно распределяются по процессорам.

- После того как все процессоры выполнили вычисления на определенном слое, им необходимо обменяться полученными результатами для расчетов на следующем слое. Для этого используется функция MPI Allgather.

КОМПЬЮТЕРНЫЕ НАУКИ 2017. Т. 27. Вып. 1

§ 4. Оценка вычислительной сложности

В данном параграфе проводится оценка трудоемкости описанного выше алгоритма, как в исходной, так и в параллельной реализации.

1. Оценка трудоемкости алгоритма без использования параллельной реализации.

- На шаге 1 алгоритма выполняется считывание начальных данных. Трудоемкостью данной процедуры можно пренебречь.

- На втором шаге алгоритма для каждого $i \in \overline{2, n-1}$ идет построение слоев функции Беллмана, при этом выполняется полный перебор всевозможных разбиений $K = K_1 \sqcup K_2$ для каждого подмножества $\overline{1, N}$, $V_i(K) := \min\limits_{K_1 \subseteq K} \{\max\{D(K_1), V_{i-1}(K_2)\}\}$. Мощность $|K| = j$, где $j \in \overline{1, N}$. Количество подмножеств $K \subseteq \overline{1, N}$ таких, что $|K| = j$, равно $\dfrac{N!}{(N-j)! \cdot j!}$, количество вариантов выбора $\widetilde{K} \subseteq K$ операций по перебору данного подмножества равно 2^j; в результате для каждого $i \in \overline{1, n-1}$ получаем оценку числа операций i-го слоя:

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

- Для $i = n$ выполняется перебор подмножеств $\overline{1, N}$; получаем оценку числа операций 2^N.

- Итоговая трудоемкость алгоритма без использования параллельной реализации получается путем суммирования трудоемкостей каждого слоя:

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

2. Оценка трудоемкости алгоритма с использованием параллельной реализации.

- На шаге 1 алгоритма выполняется считывание начальных данных и их пересылка процессорам, участвующим в вычислении. Трудоемкостью данной процедуры вновь пренебрегаем.

- На втором шаге алгоритма для $\forall i \in \overline{1, n-1}$ трудоемкость рассчитывается аналогично трудоемкости алгоритма без использования параллельной реализации. Учитывая, что расчет оптимального разбиения подмножеств выполняется на k процессорах, получаем оценку числа операций i-го слоя в виде

$$\frac{\sum\limits_{j=0}^{N} \left(\dfrac{N!}{(N-j)! \cdot j!} \cdot 2^j \right)}{k}.$$

- Для $i = n$ выполняется перебор подмножеств $\overline{1, N}$; данная процедура рассчитывается на одном процессоре, а значит, получаем оценку числа операций 2^N.

- Число операций по обмену полученных данных между процессорами рассчитывается следующим образом. В конце i-го слоя каждый процессор должен отправить 2^N значений $(V_i,\ K \subseteq \overline{1, N})$ каждому из $k-1$ оставшихся процессоров. Учитывая, что такой обмен в конце слоя осуществляется одновременно всеми процессорами, а число слоев равно n, общее число операций будет $n \cdot 2^N$. Для приведения времен выполнения операций передачи данных и операций сравнения к единым единицам измерения необходимо ввести коэффициент M, $t_{\text{пер}} \approx M \cdot t_{\text{срав}}$, где, исходя из характеристик вычислителя, M может варьироваться в пределах 200–1000. В нашем случае, исходя из вычислительного эксперимента, $M = 500$.

- Итоговая оценка числа операций сравнения алгоритма с использованием параллельной реализации может быть выражена как

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

§ 5. Вычислительный эксперимент

В данном параграфе рассматривается решение задачи нахождения оптимального разбиения N заданий на n участников (работников) методом динамического программирования с использованием супервычислителя «УРАН». Также выполнен сравнительный анализ зависимости времени счета задачи от количества вычислительных ядер кластера.

Расчеты были произведены для $n = 3, 4, 5$ работников, количество заданий N варьируется от 20 до 25. Стоимость всех заданий формируется случайным образом в интервале от 1 до 100. Для сравнения времени вычисления для разного количества ядер супервычислителя берутся идентичные начальные данные.

На основании ранее описанного алгоритма была написана программа на языке программирования $C{+}{+}$ с использованием библиотеки MPI [14]. 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 супервычислителе используются узлы на основе двухпроцессорных блейд-серверов HP ProLiant BL460c CTO Blade со следующими характеристиками:

— два четырехъядерных процессора Intel® Xeon® E5450 (3.0 GHz, 1333 FSB, 80W);
— кэш-память 2×6 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.

Для анализа зависимости времени счета задачи от количества вычислительных ядер кластера был выполнен вычислительный эксперимент, в котором число работников $n = 5$, количество заданий $N = 20$, количество ядер k варьируется от 1 до 256. Полученные данные представлены на рис. 1. Можно заметить, что при увеличении количества ядер до 32 идет существенное уменьшение времени счета. Далее наблюдается более плавное уменьшение времени счета; это объясняется тем, что при большом числе вычислительных ядер необходимо выполнять много операций обмена данными между каждым процессором; так как эти данные большие по объему, на это затрачивается много времени. Это уменьшает выигрыш по времени счета задачи от добавления дополнительного количества вычислительных ядер.

В качестве примера приводится результат одного из вычислительных экспериментов, выполненных на супервычислителе «УРАН». В данном примере количество работников $n = 3$, количество заданий $N = 25$. Функция стоимости $D(K)$, $K \subset \overline{1,N}$, получается суммированием стоимостей отдельных заданий. При этом стоимости выполнения упомянутых заданий следующие:

$$d_1 = 36,\quad d_2 = 68,\quad d_3 = 52,\quad d_4 = 28,\quad d_5 = 99,\quad d_6 = 79,\quad d_7 = 39,\quad d_8 = 40,\quad d_9 = 5,$$
$$d_{10} = 24,\quad d_{11} = 11,\quad d_{12} = 39,\quad d_{13} = 98,\quad d_{14} = 74,\quad d_{15} = 33,\quad d_{16} = 35,\quad d_{17} = 67,$$
$$d_{18} = 41,\quad d_{19} = 84,\quad d_{20} = 87,\quad d_{21} = 52,\quad d_{22} = 100,\quad d_{23} = 97,\quad d_{24} = 99,\quad d_{25} = 36.$$
$$D(\oslash) \triangleq 0,\qquad D(K) = \sum_{i\in K} d_i.$$

В результате выполнения программы были получены следующие результаты. Первый работник получил работы с индексами $K_1 = \{1, 2, 3, 4, 5, 6, 7, 10, 11, 12\}$. Стоимость распределения равна

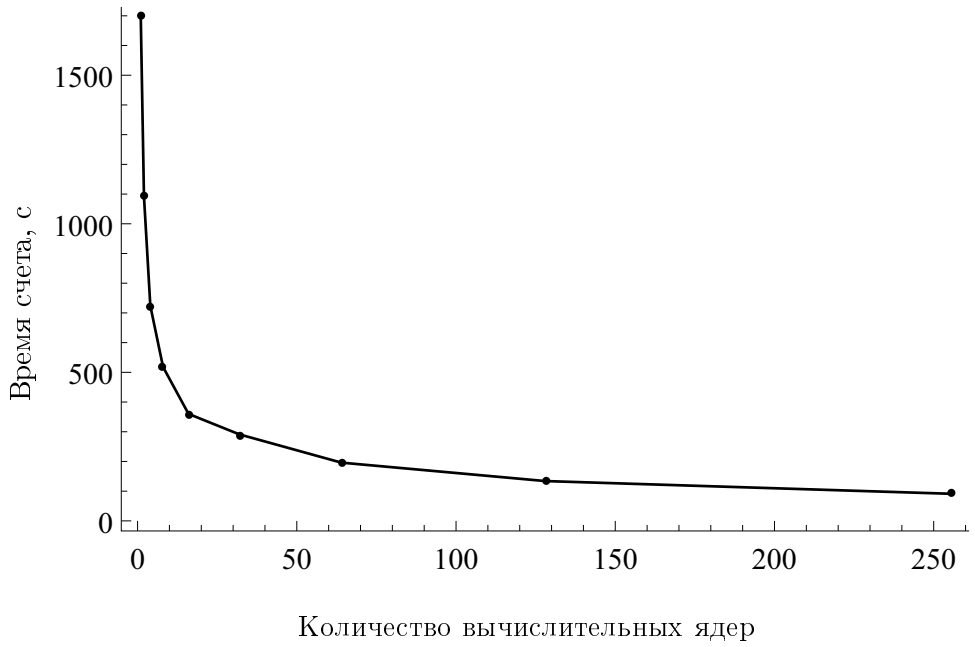

Рис. 1. Диаграмма зависимости времени счета задачи на супервычислителе от количества вычислительных ядер

$D(K_1) = 475$. Второй работник получил работы с индексами $K_2 = \{9, 13, 14, 15, 18, 19, 20, 21\}$. Стоимость распределения равна $D(K_2) = 474$. Третий работник получил работы с индексами $K_3 = \{8, 16, 17, 22, 23, 24, 25\}$. Стоимость распределения равна $D(K_3) = 474$. Значение задачи V равно 475.

Время счета программы составило 3 часа 56 минут.

§ 6. 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. Приведен вычислительный эксперимент для сравнения этих двух алгоритмов. Полученные результаты показали, что алгоритм с использованием параллельной структуры дает существенный выигрыш по времени вычисления для 32 вычислительных ядер. Далее идет более плавное сокращение времени счета программы. При вычислении слоев функции Беллмана процессоры обмениваются промежуточными данными, вследствие чего возникают накладные расходы по времени, что приводит к уменьшению эффекта от использования параллельного алгоритма для большого количества вычислительных ядер, участвующих в расчетах.

СПИСОК ЛИТЕРАТУРЫ

1. Коробкин В.В., Сесекин А.Н., Ташлыков О.Л., Ченцов А.Г. Методы маршрутизации и их приложения в задачах повышения эффективности и безопасности экспуатации атомных станций. М.: Новые технологии, 2012. 234 с.
2. Burkard R., Dell'Amico M., Martello S. Assignment problems. SIAM Philadelphia, 2009. DOI: 10.1137/1.9781611972238
3. Kuhn H.W. The Hungarian method for the assignment problem // Naval Research Logistics Quarterly. 1955. Vol. 2. No. 1–2. P. 83–97. DOI: 10.1002/nav.3800020109
4. Kellerer H., Pferschy U., Pisinger D. Knapsack problems. Springer, 2005. DOI: 10.1007/978-3-540-24777-7
5. Papadimitriou C.H., Yannakakis M. Optimization, approximation and complexity classes // Journal of Computer and System Sciences. 1991. Vol. 43. Issue 3. P. 425–440. DOI: 10.1016/0022-0000(91)90023-X

6. 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.
7. Ченцов А.Г., Ченцов П.А. Динамическое программирование в задаче оптимизации разбиений // Автоматика и телемеханика. 2002. № 5. С. 133–146.
8. Ченцов А.Г. Экстремальные задачи маршрутизации и распределения заданий: вопросы теории. М.–Ижевск: НИЦ «Регулярная и хаотическая динамика», Институт компьютерных исследований, 2008. 240 с.
9. Ченцов П.А. О некоторых алгоритмах распределения заданий между участниками // Автоматика и телемеханика. 2006. № 8. С. 77–91.
10. Беллман Р. Динамическое программирование. М.: ИЛ, 1960. 400 с.
11. Мину М. Математическое программирование. М.: Наука, 1990. 488 с.
12. Коротаева Л.Н., Назаров Э.М., Ченцов А.Г. Об одной задаче о назначениях // Журнал вычислительной математики и математической физики. 1993. Т. 33. № 4. С. 483–494.
13. Григорьев А.М. Решение минимаксной распределительной задачи методом динамического программирования с применением параллельных вычислений // Научный сервис в сети Интернет: экзафлопсное будущее: Труды Международной суперкомпьютерной конференции. М.: Изд-во МГУ, 2011. С. 580–586.
14. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. М.: Изд-во МГУ, 2004. 71 с.

Поступила в редакцию 25.01.2017

Григорьев Алексей Михайлович, заведующий отделом, Институт математики и механики УрО РАН, 620219, Россия, г. Екатеринбург, ул. С. Ковалевской, 16.
E-mail: ag@uran.ru

A. M. Grigoryev

Solution of the problem of optimal task distribution by the method of dynamic programming with parallel computing

Citation: *Vestnik Udmurtskogo Universiteta. Matematika. Mekhanika. Komp'yuternye Nauki*, 2017, vol. 27, issue 1, pp. 129–137 (in Russian).

Keywords: dynamic programming, partition, parallel algorithm.

MSC2010: 49L20, 90C39

DOI: 10.20537/vm170111

The aim of the study is to construct a parallel algorithm for solving a bottleneck (minmax) problem connected with partitioning a finite set of tasks between a finite number of agents. We describe the algorithm of finding an optimal partition of tasks through dynamic programming with a parallel computation of the Bellman function and provide a computational complexity estimate for the two algorithms (with and without the parallel construction). The algorithm was implemented for the Uran supercomputer, and a computational experiment was conducted; computation time was measured for the serial algorithm and for the parallel one on varying numbers of processor cores.

REFERENCES

1. Korobkin V.V., Sesekin A.N., Tashlykov O.L., Chentsov A.G. *Metody marshrutizatsii i ikh prilozheniya v zadachakh povysheniya effektivnosti i bezopasnosti ekspluatatsii atomnykh stantsii* (Routing methods and their applications in problems of improving the efficiency and safety of operation of nuclear power plants), Moscow: Novye tekhnologii, 2012, 234 p.
2. Burkard R., Dell'Amico M., Martello S. *Assignment problems*, SIAM Philadelphia, 2009.
DOI: 10.1137/1.9781611972238

3. Kuhn H.W. The Hungarian method for the assignment problem, *Naval Research Logistics Quarterly*, 1955, vol. 2, no. 1–2, pp. 83–97. DOI: 10.1002/nav.3800020109
4. Kellerer H., Pferschy U., Pisinger D. *Knapsack problems*, Springer, 2005. DOI: 10.1007/978-3-540-24777-7
5. Papadimitriou C.H., Yannakakis M. Optimization, approximation and complexity classes, *Journal of Computer and System Sciences*, 1991, vol. 43, issue 3, pp. 425–440. DOI: 10.1016/0022-0000(91)90023-X
6. Chentsov A.G., Chentsov P.A. On the construction of a procedure for partitioning a finite set, based on the dynamic programming method, *Automation and Remote Control*, 2000, vol. 61, issue 4, pp. 658–670.
7. Chentsov A.G., Chentsov P.A. Dynamic programming in the problem of decomposition optimization, *Automation and Remote Control*, 2002, vol. 63, issue 5, pp. 815–828. DOI: 10.1023/A:1015406222958
8. Chentsov A.G., Chentsov A.G. *Ekstremal'nye zadachi marshrutizatsii i raspredeleniya zadanii: voprosy teorii* (Extremal problems of routing and assignment of tasks: questions of theory), Moscow–Izhevsk: Regular and Chaotic Dynamics, Institute of Computer Science, 2008, 240 p.
9. Chentsov P.A. Job distribution algorithms, *Automation and Remote Control*, 2006, vol. 67, issue 8, pp. 1251–1264. DOI: 10.1134/S0005117906080054
10. Bellman R. *Dynamic programming*, Princeton, New Jersey: Princeton University Press, 1957. Translated under the title *Dinamicheskoe programmirovanie*, Moscow: Inostr. Lit., 1960, 400 p.
11. Minoux M. *Programation mathematique. Theorie et algorithmes*, Paris: Dunod, 1983, tome 1: 294 p., tome 2: 236 p. Translated under the title *Matematicheskoe programmirovanie. Teoriya i algoritmy*, Moscow: Nauka, 1990, 488 p.
12. Korotaeva L.N., Nazarov E.M., Chentsov A.G. An assignment problem, *Computational Mathematics and Mathematical Physics*, 1993, vol. 33, issue 4, pp. 443–452.
13. Grigoryev A.M. Solution of minimax distribution problem by method of dynamic programming with the use of parallel computing, *Nauchnyi servis v seti Internet: ekzaflopsnoe budushchee: Trudy Mezhdunarodnoi superkomp'yuternoi konferentsii* (Scientific service in the Internet: Exaflop future: Proceedings of the International Supercomputer Conference), Moscow: Lomonosov Moscow State University, 2011, pp. 580–586 (in Russian).
14. Antonov A.S. *Parallel'noe programmirovanie s ispol'zovaniem tekhnologii MPI* (Parallel programming using the MPI technology), Moscow: Lomonosov Moscow State University, 2004, 71 p.

Received 25.01.2017

Grigoryev Alexey Mikhailovich, Head of Department, N.N. Krasovskii Institute of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences, ul. S. Kovalevskoi, 16, Yekaterinburg, 620219, Russia. E-mail: ag@uran.ru