Моделирование и анализ информационных систем. Т. 25, № 2 (2018), с. 155–164
Modeling and Analysis of Information Systems. Vol. 25, No 2 (2018), pp. 155–164

Модели параллелизма

©Васильчиков В. В., 2017

DOI: 10.18255/1818-1015-2018-2-155-164

УДК 519.688: 519.85

О рекурсивно-параллельном алгоритме решения задачи о рюкзаке

Васильчиков В. В.

получена 27 декабря 2017

Аннотация. Предлагается эффективный параллельный алгоритм решения NP-полной задачи о рюкзаке в ее исходном, так называемом 0-1 варианте. Для нахождения ее точного решения издавна применяются алгоритмы, относящиеся к категории "методов ветвей и границ". Для ускорения получения результата с разной степенью эффективности применяются также различные варианты организации параллельных вычислений. Мы предлагаем здесь алгоритм решения задачи, основанный на парадигме рекурсивно-параллельных вычислений. Он представляется нам хорошо пригодным для задач такого рода, когда трудно сразу разбить вычисления на достаточное количество сравнимых по трудоемкости подзадач, поскольку она проявляется динамически во время вычислений. В качестве основного инструмента для программной реализации алгоритма использовалась разработанная автором библиотека RPM_ParLib, позволяющая создавать эффективные приложения для вычислений на локальной сети в среде .NET Framework. Такие приложения обладают способностью порождать параллельные ветви вычислений непосредственно во время выполнения программы и динамически перераспределять работу между вычислительными модулями. При этом в качестве языка программирования может использоваться любой язык с поддержкой .NET Framework. Для проведения экспериментов было написано несколько программ на языке C# с использованием упомянутой библиотеки. Основной целью этих экспериментов было исследование ускорения, достигаемого за счет рекурсивно-параллельной организации вычислений. Подробное описание алгоритма и эксперимента, а также полученные результаты приводятся в работе.

Ключевые слова: задача о рюкзаке, параллельный алгоритм, рекурсия, .NET

Для цитирования: Васильчиков В. В., "О рекурсивно-параллельном алгоритме решения задачи о рюкзаке", *Моделирование и анализ информационных систем*, **25**:2 (2018), 155–164.

Об авторах:
Васильчиков Владимир Васильевич, orcid.org/0000-0001-7882-8906, канд. техн. наук, зав. кафедрой вычислительных и программных систем,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: vvv193@mail.ru

Благодарности:
Работа выполнена в рамках инициативной НИР ВИП-004 (номер госрегистрации АААА-А16-116070610022-6).

Введение

Задача о рюкзаке является одной из классических задач дискретной оптимизации [1]. Для нее существует несколько различных вариантов постановки [2, 3] и алгоритмов решения [2–4]. В данной статье мы рассматривам только одну ее разновидность, а именно классический 0-1 вариант. В такой постановке задача относится к категории NP-полных задач, для которых не найдено алгоритмов точного решения за полиномиальное от размера задачи время [1]. Однако для решения ряда задач такого рода с успехом применяются алгоритмы, относящиеся с категории "методов ветвей и границ". Идея метода было предложена в [5] применительно к задаче целочисленного линейного программирования. Для задачи о рюкзаке алгоритм такого рода и целый ряд его модификаций был разработан и исследован Д. Пизингером [6, 7]. Для оценивания перспективности вычислений на очередной ветви, а это один из ключевых моментов такого рода алгоритмов, он использовал оценки, полученные Г. Данцигом в [8].

По своей природе метод ветвей и границ все-таки является перебором вариантов, хотя упомянутые выше оценки перспективности продолжения вычислений на очередной ветви позволяют существенно увеличить скорость получения результата. Однако совершенно естественным выглядит желание добиться ускорения за счет построения параллельного алгоритма решения задачи. Разные авторы посвящали свои работы построению параллельных алгоритмов решения задачи о рюкзаке, основанных на самых разных парадигмах и инструментах параллельного программирования, в качестве одного из примеров упомянем работу [9]. При этом основной проблемой при построении такого рода алгоритмов является разбиение целой задачи на подзадачи сравнимой трудоемкости и обеспечение достаточно равномерной загрузки вычислительных мощностей.

Метод ветвей и границ практически всегда допускает описание в виде рекурсии, поэтому автору показалось совершенно естественным использовать для распараллеливания алгоритма концепцию рекурсивно-параллельного (РП) программирования и соответствующие библиотеки поддержки. В [10] изложены основные принципы организации рекурсивно-параллельных вычислений и описаны основные алгоритмы и механизмы поддержки этого стиля программирования. Библиотеки [11,12] позволяют относительно легко создавать, отлаживать и эксплуатировать РП-приложения в среде .NET Framework. В [13] подробно описаны функциональные возможности упомянутых библиотек, а также процесс разработки и исследования РП-алгоритма решения NP-полной задачи о клике. В качестве другого примера использования данного стиля программирования и упомянутых библиотек приведем работу [14], в которой разработан и исследован рекурсивно-параллельный алгоритм решения задачи коммивояжера, также относящейся к категории NP-полных.

1. Задача о рюкзаке

Напомним формулировку задачи. При этом мы будем в основном придерживаться обозначений, использованных в [6], поскольку именно один из алгоритмов, изложенных в данной работе, положен в основу нашего параллельного алгоритма.

Существует несколько постановок задачи о рюкзаке. Все эти постановки предполагают, что имеется некоторое количество предметов, для каждого из которых задана ценность p_i и размер c_i. Рюкзак характеризуется предельной вместимостью c. Все перечисленные величины p_i, c_i, c являются целыми положительными числами. Нас будет интересовать постановка задачи в так называемом 0-1 варианте. Она задается следующим образом.

Пусть у нас имеется n предметов с заданной ценностью и размером. Требуется максимизировать величину ценности выбранных предметов $\sum_{i=1}^{n} p_i x_i$ при ограничении на вместимость рюкзака $\sum_{i=1}^{n} c_i x_i \leq c$, где . Здесь x_i – бинарная величина, равная 1, если мы положили предмет в рюкзак, и 0 в противном случае. Другие варианты формулировки задачи о рюкзаке мы здесь не рассматриваем.

2. Последовательный алгоритм решения задачи

Вкратце опишем основные шаги алгоритма, приведенного в [6], поскольку именно он является базовым для построения нашего параллельного алгоритма.

1. Упорядочиваем предметы в порядке неувеличения эффективности $e_i = p_i/w_i$, так что $e_i \geq e_j$ при $i < j$.

2. Набираем предметы в рюкзак "жадным" образом, пока позволяет его вместимость. Пусть первый элемент, не поместившийся в рюкзак, имеет номер b, назовем его предельным. Если обозначить через W_k суммарный размер первых k элементов из нашего упорядоченного множества, имеем соотношение $W_{b-1} \leq c < W_b$. Таким образом мы построили базовый набор из $b-1$ предметов стоимости P_{b-1}. Ему соответствует набор значений x_i, такой что $x_i = 1$ для $1 \leq i \leq b-1$ и $x_i = 0$ для $b \leq i \leq n$.

3. Дальнейшая работа алгоритма заключается в попытках улучшить базовое решение. Изменения фиксируются в виде так называемого списка исключений, содержащего номера элементов из массива $\{x_i\}$, значения которых следует изменить на противоположные.

Действия из последнего пункта выполняются рекурсивно в соответствии с методологией "ветвей и границ". При этом множество потенциальных исключений расширяется в обе стороны от значения b. Оно представляет собой целые значения из промежутка с нижней границей s и верхней границей t. Основными параметрами рекурсивной процедуры являются ценность построенного набора, его размер, s и t. При первом вызове они задаются значениями P_{b-1}, W_{b-1}, $b-1$ и b соответственно.

На очередном шаге рекурсии, если размер предмета с индексом t позволяет его добавить в построенный набор, рекурсивно вычисляется ценность решения для этого варианта, t включается в список исключений и увеличивается на единицу. Если предмет с индексом t не помещается в построенный набор, пытаемся удалить из набора предмет с индексом s, применяя для этого соответствующий рекурсивный вызов. Более понятно этот процесс иллюстрируется приведенной ниже блок-схемой заключительной фазы рекурсивно-параллельного алгоритма, поскольку она фактически повторяет последовательный алгоритм.

Отметим, что важнейшим элементом алгоритмов и категории "методов ветвей и границ" является оценка перспективности вычислений на отдельно взятой ветви. Как в базовом алгоритме [6], так и у нас используется оценка, предложенная Г. Данцигом в [8]. В соответствии с ней, верхней оценкой ценности итогового набора предметов для нашей задачи является

$$u = \lfloor P_{b-1} + \frac{(c - W_{b-1})p_b}{w_b} \rfloor,$$

где $\lfloor a \rfloor$ – наибольшее целое, не превышающее a.

Соответственно, на ветви, где осуществляется проверка перспективности добавления предмета с индексом t, то есть в ситуации $W \leq c$, верхняя оценка не превысит z, если

$$u = \lfloor P + \frac{(c - W)p_t}{w_t} \rfloor \leq z,$$

а при оценке перспективности ветви с удалением предмета s $(W > c)$ следует проверять условие

$$u = \lfloor P + \frac{(c - W)p_s}{w_s} \rfloor \leq z.$$

В программном коде удобнее использовать эквивалентные проверки: для $W \leq c$

$$det(P - z - 1, W - c, p_t, w_t) < 0,$$

а для $W > c$

$$det(P - z - 1, W - c, p_s, w_s) < 0.$$

3. Рекурсивно-параллельный алгоритм

Для задач такого рода, когда алгоритм естественным образом описывается рекурсивно с разбиением на две подзадачи, построение РП-алгоритма, казалось бы, не представляет сложности. Вместе с тем опыт разработки и исследования таких программ [13,14] показывает, что такой прямолинейный подход не всегда является наилучшим, поскольку не позволяет в полной мере использовать возможности механизма динамической балансировки загрузки, заложенные в библиотеки поддержки данного стиля программирования. Поэтому при решении задач, описанных в [13,14], мы использовали специальные приемы оформления подзадач для достижения большего ускорения за счет параллельного выполнения. Какой прием окажется лучше, каждый раз определяется экспериментально.

В случае задачи о рюкзаке достаточно хорошие результаты показал, однако, именно такой прямолинейный подход, когда в качестве подзадачи на каждом шаге рекурсии оформлялась ветка, удовлетворяющая условию $W \leq c$ или $W > c$ соответственно. Здесь, правда, на первый план выступают другие аспекты оформления подзадачи, связанные в основном с необходимостью эффективной организации структур данных, передаваемых подзадаче в качестве исходных, в виде так называемого блока параметров, а также возвращаемых через блок параметров результатов. К слову сказать, в [6] вопросам принадлежности данных к тому или иному

классу внимания уделено не было, а неправильное понимание этого момента приводит к неверной работе всего алгоритма. Поэтому ниже перед описанием собственно алгоритма мы опишем структуру блока параметров рекурсивно-параллельной активации. Другой особенностью данной задачи и алгоритма ее решения является то, что результирующее множество строится на обратном ходе рекурсии, а значит, неизвестно в момент принятия решения о включении или исключении из него очередного предмета. И наконец, проверку перспективности очередной ветви мы производим еще до ее порождения, чтобы не создавать фактически пустых потенциально-мигрирующих активаций РП-процедур.

Основными значениями, включенными в блок параметров, в нашем варианте являются следующие:

- Текущий уровень вложенности рекурсии, он увеличивается только при удвоении их числа. В качестве ограничения рекурсивного деления мы используем некоторое предельное значение уровня вложенности. В нашем случае это позволяет достаточно хорошо управлять процессом рекурсивного деления.

- Стартовые значения W, P, s и t, а также признак того, что на данной ветви результат был улучшен ($improved$), и соответствующее улучшенное значение ценности ($localBest$).

- Список исключений, накопленных данной активацией и ее потомками. Создается на обратном ходе рекурсии в тех случаях, когда он требуется для возврата.

На рисунке 1 приводится блок-схема, представляющая логику исполнения параллельного вызова основной вычислительной процедуры нашего алгоритма. Она позволяет понять, как и в каких случаях осуществляется разбиение задачи на две подзадачи, как осуществляется синхронизация выполнения и формирование результата.

По достижении заданного предела дробления задачи дальнейшие вычисления осуществляет процедура $RecursiveBranch(W, P, s, t)$, которая реализует последовательное решение. Блок-схема ее работы представлена на рисунке 2.

4. Результаты тестирования

Исходные данные для тестирования генерировались случайным образом. При этом размер и ценность предметов варьировались не слишком значительно, чтобы дополнительно усложнить задачу. Было сгенерировано несколько десятков наборов исходных данных объемом в 60 и 80 предметов. Размер и ценность их брались случайно из диапазона от 1850 до 2150. Размер рюкзака позволял взять примерно половину предметов. Но даже для таких примерно однотипных наборов исходных данных время работы последовательного алгоритма варьировалось как минимум в десятки тысяч раз. Некоторые решались быстрее секунды, для других не хватало двух часов, в этих случаях работу программы приходилось завершать, не дождавшись получения окончательного решения. Поэтому при исследовании эффективности параллельного алгоритма мы отобрали только те варианты исходных данных,

на которых последовательный алгоритм выдавал решение за время в пределах от 3 минут до 1 часа.

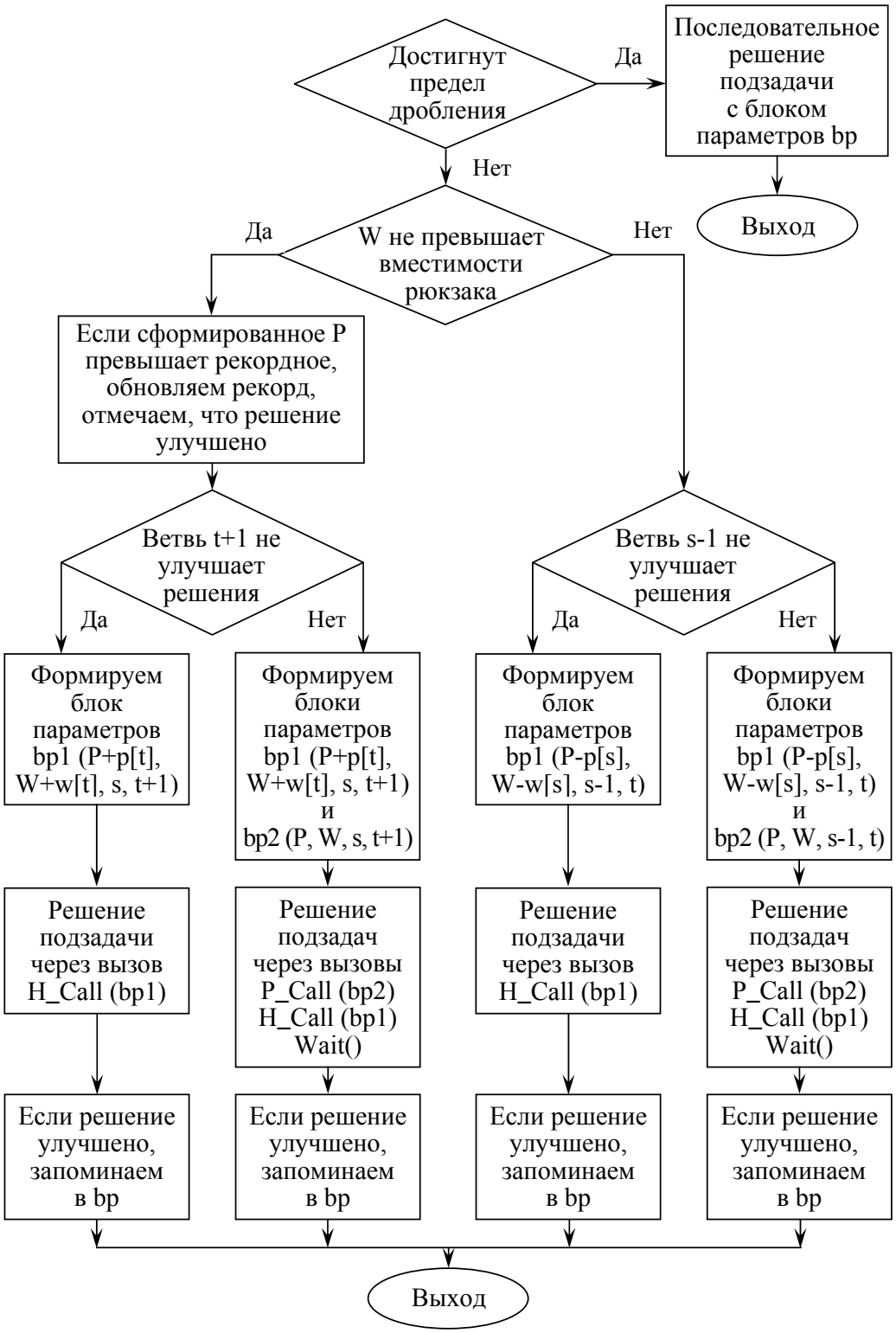

Рис. 1. Блок-схема параллельного выполнения основной процедуры
Fig. 1. Block diagram of parallel execution of the main procedure

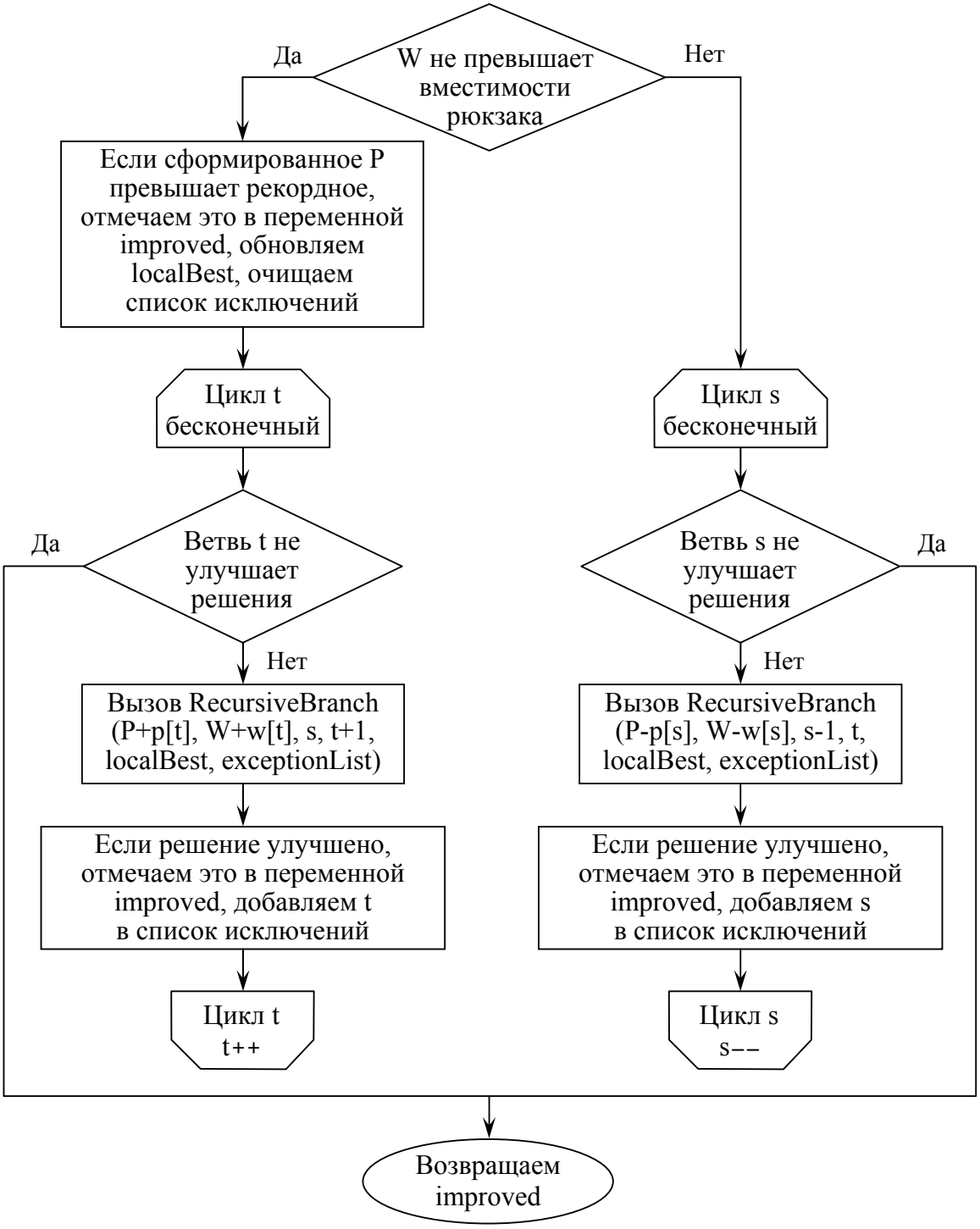

Рис. 2. Блок-схема последовательного выполнения основной процедуры
Fig. 2. Block diagram of serial execution of the main procedure

В процессе тестирования использовались компьютеры на базе четырехъядерного процессора Intel Core i3 с тактовой частотой 3.07 GHz и 4 GB оперативной памяти, работающие под управлением 64-разрядной ОС Windows 7. Пропускная способность сети равнялась 100 Mb/s.

В таблице 1 приведены некоторые результаты вычислений, позволяющие оценить трудоемкость решения задачи, время решения и ускорение, полученное для нескольких случайных наборов исходных данных из 80 предметов. Трудоемкость задачи можно оценить по указанному количеству порожденных параллельных ветвей (суммарно для всех процессорных модулей – ПМ).

Таблица 1. Трудоемкость и ускорение РП-алгоритма на задаче с 80 предметами

Table 1. The complexity and acceleration of the RP-algorithm on a problem with 80 objects

№ п/п	**Кол-во ПМ**	**1**	**2**	**4**	**6**	**8**	**12**	**16**
1	Ветвей (млн)	*10 551.23*	*154.76*	*304.54*	*241.29*	*298.51*	*220.68*	*222.12*
	Время (с)	547.62	8.33	7.37	5.72	5.78	5.79	5.25
	Ускорение	***1.00***	***65.77***	***74.35***	***95.69***	***94.81***	***94.56***	***104.39***
2	Ветвей (млн)	*15 114.60*	*25 538.38*	*51 228.36*	*39 633.71*	*33 172.25*	*16 164.25*	*19 662.56*
	Время (с)	762.86	663.94	654.11	338.66	209.18	70.86	67.27
	Ускорение	***1.00***	***1.15***	***1.17***	***2.25***	***3.65***	***10.77***	***11.34***
3	Ветвей (млн)	*45 510.13*	*33 072.12*	*30 943.51*	*30 197.71*	*15 252.44*	*21 327.09*	*11 504.56*
	Время (с)	2 263.17	859.64	393.76	260.25	98.73	93.61	41.95
	Ускорение	***1.00***	***2.63***	***5.75***	***8.70***	***22.92***	***24.18***	***53.95***
4	Ветвей (млн)	*5 332.43*	*2 359.90*	*2 020.09*	*2 894.13*	*3 846.42*	*5 689.95*	*7 175.60*
	Время (с)	272.35	65.69	31.37	27.42	28.26	27.01	27.40
	Ускорение	***1.00***	***4.15***	***8.68***	***9.93***	***9.64***	***10.08***	***9.94***
5	Ветвей (млн)	*21 477.44*	*3 453.82*	*5 502.24*	*6 892.83*	*8 233.19*	*3 530.34*	*1 559.01*
	Время (с)	1 068.25	94.26	73.25	62.82	56.19	18.59	9.49
	Ускорение	***1.00***	***11.33***	***14.58***	***17.00***	***19.01***	***57.48***	***112.54***
6	Ветвей (млн)	*3 956.54*	*1 843.44*	*3 768.49*	*1 394.06*	*1 518.84*	*2 242.82*	*1 046.64*
	Время (с)	209.42	52.65	51.37	14.37	12.32	12.71	6.87
	Ускорение	***1.00***	***3.98***	***4.08***	***14.58***	***17.00***	***16.48***	***30.47***
7	Ветвей (млн)	*47 836.56*	*33 829.43*	*12 369.67*	*17 925.07*	*9 858.63*	*11 678.20*	*13 985.47*
	Время (с)	2 466.22	878.55	161.71	155.08	64.90	52.94	48.84
	Ускорение	***1.00***	***2.81***	***15.25***	***15.90***	***38.00***	***46.58***	***50.49***
8	Ветвей (млн)	*4 650.47*	*1 041.19*	*1 178.17*	*1 317.48*	*1 432.33*	*1 730.72*	*2 169.74*
	Время (с)	229.81	27.93	16.01	14.47	12.10	10.63	13.60
	Ускорение	***1.00***	***8.23***	***14.36***	***15.88***	***19.00***	***21.62***	***16.90***
9	Ветвей (млн)	*52 898.13*	*92 754.95*	*36 778.80*	*47 898.28*	*21 868.91*	*29 560.00*	*29 560.00*
	Время (с)	2 612.83	2 365.98	489.68	435.12	150.86	139.26	77.43
	Ускорение	***1.00***	***1.10***	***5.34***	***6.00***	***17.32***	***18.76***	***33.74***
10	Ветвей (млн)	*45 674.54*	*9 553.20*	*29 971.79*	*13 136.65*	*24 774.09*	*5 885.24*	*10 219.88*
	Время (с)	2 345.08	252.49	383.23	119.37	160.02	28.44	38.31
	Ускорение	***1.00***	***9.29***	***6.12***	***19.64***	***14.66***	***82.46***	***61.21***

Анализируя полученные результаты, можно сделать следующие выводы. Конечно, точность оценки решения на исследуемой ветви вычислений имеет очень важное значение для скорости получения результата, однако это далеко не единственный фактор, определяющий время вычислений как при последовательном, так и при параллельном решении задачи.

Как показывают результаты экспериментов, едва ли не более важным фактором, определяющим скорость решения задачи, является то, как быстро будет найдено лучшее или достаточно близкое к нему решение. После нахождения такового количество ветвей вычислений, отвергаемых на ранних стадиях ввиду их бесперспективности, резко возрастает. По этой причине время решения конкретной задачи является непредсказуемым, и становится весьма некорректным делать какие-то выводы о зависимости этого времени, скажем, от размера задачи, если только он не изменяется очень сильно.

Мы наблюдали эту особенность и при исследовании других задач [13,14], но при исследовании параллельной версии решения задачи о рюкзаке она проявилась очень заметно. Так, именно ею объясняется тот факт, что при параллельной работе алгоритма очень часто ускорение превышает (и весьма значительно) количество вычислительных модулей. Причиной, очевидно, является то, что одновременно исследуя несколько ветвей, мы почти всегда раньше находим достаточно хорошее решение, что позволяет отказаться от анализа большего количества неперспективных подзадач. Этот факт, на наш взгляд, является веским аргументом в пользу применения РП-программирования для решения задач такого рода.

Список литературы / References

[1] Гэри М., Джонсон Д., *Вычислительные машины и труднорешаемые задачи*, Мир, М., 1982; In English: Garey M.R., Johnson D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co, San Francisco, 1979.

[2] Kellerer H., Pferschy U., Pisinger D., *Knapsack Problems*, Springer, Berlin, 2004.

[3] Martello S., Toth P., *Knapsack Problems Algorithms and Computer Implementation*, Wiley, New York, 1990.

[4] Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К., *Алгоритмы: построение и анализ*, Вильямс, М., 2006; In English: Cormen T., Leiserson Ch., Rivest R., Stein C., *Introduction to Algorithms*, The MIT Press, 2001.

[5] Land A. H., Doig A. G., “An Autmatic Method of Solving Discrete Programming Problems”, *Econometrica*, **28** (1960), 497–520.

[6] Pisinger D., “An expanding-core algorithm for the exact 0-l knapsack problem”, *Eur. J. Oper. Res.*, **87**:1 (1995), 175–187.

[7] Pisinger D., “A fast algorithm for strongly correlated knapsack problems”, *Discrete Applied Mathematics*, **89**:1–3 (1998), 197–212.

[8] Dantzig G. B., “Discrete Variable Extremum Problems”, *Operation Ressearch*, **5** (1957), 266–277.

[9] Посыпкин М.А., Сигал И.Х., “Комбинированный параллельный алгоритм решения задачи о ранце”, *Труды четвертой международной конференции "Параллельные вычисления и задачи управления"*, 2008, 177–189; [Posypkin M.A., Sigal I.Kh., “Kombinirovannyj parallelnyj algoritm reshenija zadachi o rance”, *Proceedings of the Fourth International Conference "Parallel Computations and Control Problems"*, 2008, 177–189, (in Russian).]

[10] Васильчиков В.В., *Средства параллельного программирования для вычислительных систем с динамической балансировкой загрузки*, Ярославль, 2001; [Vasilchikov V.V., *Sredstva parallelnogo programmirovaniya dlya vychislitelnykh sistem s dinamicheskoy balansirovkoy zagruzki*, Yaroslavl, 2001, (in Russian).]

[11] Васильчиков В.В., *Коммуникационный модуль для организации полносвязного соединения компьютеров в локальной сети с использованием .NET Framework*, Свидетельство о государственной регистрации программы для ЭВМ № 2013619925, 2013; [Vasilchikov V.V., *Kommunikatsionnyy modul dlya organizatsii polnosvyaznogo soedineniya kompyuterov v lokalnoy seti s ispolzovaniem .NET Framework*, Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2013619925, 2013, (in Russian).]

[12] Васильчиков В.В., *Библиотека поддержки рекурсивно-параллельного программирования для .NET Framework*, Свидетельство о государственной регистрации программы для ЭВМ № 2013619926, 2013; [Vasilchikov V. V., *Biblioteka podderzhki rekursivno-parallelnogo programmirovaniya dlya .NET Framework*, Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2013619926, 2013, (in Russian).]

[13] Васильчиков В.В., "О поддержке рекурсивно-параллельного программирования в .NET Framework", *Модел. и анализ информ. систем*, **21**:2 (2014), 15–25; [Vasilchikov V. V., "On the Recursive-Parallel Programming for the .NET Framework", *Modeling and Analysis of Information Systems*, **21**:2 (2014), 15–25, (in Russian).]

[14] Васильчиков В.В., "Об оптимизации и распараллеливании алгоритма Литтла для решения задачи коммивояжера", *Модел. и анализ информ. систем*, **23**:4 (2016), 401–411; [Vasilchikov V. V., "On Optimization and Parallelization of the Little Algorithm for Solving the Travelling Salesman Problem", *Modeling and Analysis of Information Systems*, **23**:4 (2016), 401–411, (in Russian).]

Vasilchikov V. V., "On a Recursive-Parallel Algorithm for Solving the Knapsack Problem", *Modeling and Analysis of Information Systems*, **25**:2 (2018), 155–164.

DOI: 10.18255/1818-1015-2018-2-155-164

Abstract. In this paper, we offer an efficient parallel algorithm for solving the NP-complete Knapsack Problem in its basic, so-called 0-1 variant. To find its exact solution, algorithms belonging to the category "branch and bound methods" have long been used. To speed up the solving with varying degrees of efficiency, various options for parallelizing computations are also used. We propose here an algorithm for solving the problem, based on the paradigm of recursive-parallel computations. We consider it suited well for problems of this kind, when it is difficult to immediately break up the computations into a sufficient number of subtasks that are comparable in complexity, since they appear dynamically at run time. We used the RPM_ParLib library, developed by the author, as the main tool to program the algorithm. This library allows us to develop effective applications for parallel computing on a local network in the .NET Framework. Such applications have the ability to generate parallel branches of computation directly during program execution and dynamically redistribute work between computing modules. Any language with support for the .NET Framework can be used as a programming language in conjunction with this library. For our experiments, we developed some C# applications using this library. The main purpose of these experiments was to study the acceleration achieved by recursive-parallel computing. A detailed description of the algorithm and its testing, as well as the results obtained, are also given in the paper.

Keywords: Knapsack Problem, parallel algorithm, recursion, .NET

On the authors:
Vladimir V. Vasilchikov, orcid.org/0000-0001-7882-8906, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., 14, Yaroslavl, 150003, Russia, e-mail: vvv193@mail.ru

Acknowledgments:
This work was supported by initiative program VIP-004 (state registration number AAAA-A16-116070610022-6).