БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Власов В. З., Леонтьев Н. И. Балки, плиты, оболочки на упругом основании. М., 1960.
2. Королевич В. В., Медведев Д. Г. Интегральные уравнения Вольтерра 2-го рода в задачах изгиба вращающихся полярно-ортотропных дисков переменной толщины // Вестн. БГУ. Сер. 1. 2012. № 3. С. 108–116.
3. Краснов М. Л., Киселев А. И., Макаренко Г. И. Интегральные уравнения: задачи и примеры с подробными решениями. М., 2007.
4. Калиткин Н. Н. Численные методы. М., 1978.
5. Верлань А. Ф., Сизиков В. С. Интегральные уравнения: методы, алгоритмы, программы: справ. пособие. Киев, 1986.
6. Кинасошвили Р. С. Расчет на прочность дисков турбомашин. М., 1954.

Поступила в редакцию 21.06.13.

Владимир Васильевич Королевич – преподаватель Национального педагогического университета им. М. Драгоманова (Прага, Чехия).

Дмитрий Георгиевич Медведев – кандидат физико-математических наук, доцент, декан механико-математического факультета.

УДК 004.93’11

Ф. И. ТРЕТЬЯКОВ, Л. В. СЕРЕБРЯНАЯ

РАСПАРАЛЛЕЛИВАНИЕ АЛГОРИТМОВ КЛАССИФИКАЦИИ И КЛАСТЕРИЗАЦИИ ДАННЫХ

Рассмотрен общий алгоритм организации параллельных вычислений. Представлены сведения об алгоритмах *k*-средних и максимина. Рассмотрены средства платформы .NET для распараллеливания алгоритмов *k*-средних и максимина. Описаны особенности организации параллельных вычислений, определены критерии, указывающие на способность алгоритма к представлению в параллельном виде. Разработаны версии алгоритмов *k*-средних и максимина, построенные на основе параллельных вычислений. Решены задачи классификации и кластеризации с помощью параллельных вычислений с использованием алгоритмов соответственно *k*-средних и максимина. Оба алгоритма поддаются распараллеливанию, поскольку в каждом из них существует минимум две операции с некоррелирующими результатами. Распараллеливание вычислений демонстрирует уменьшение времени выполнения алгоритмов уже при двух процессорах. Увеличение производительности алгоритмов линейно зависит от увеличения числа вычислителей. С увеличением количества объектов классификации увеличивается производительность параллельных вычислений. Для алгоритма *k*-средних эта зависимость нелинейная, а для алгоритма максимин – линейная. С увеличением количества классов в алгоритме *k*-средних линейно увеличивается производительность параллельных вычислений. Полученные результаты подтвердили целесообразность распараллеливания вычислений в алгоритмах к-средних и максимина, что увеличивает эффективность классификации и кластеризации данных.

Ключевые слова: *k*-средние; максимин; параллельные вычисления; алгоритмизация; производительность.

There were considered parallel computing common algorithm. Common information about *k*-means and maximin algorithms were represented. There were considered *k*-means and maximin .NET paralleling tools. There were described parallel computing features; there were defined algorithm abilities to work as parallel. *K*-means and maximin parallel versions have been designed. There were solved classification and clustering tasks using parallel computing with *k*-means and maximin algorithms, respectively. Both algorithms can be paralleled, because each of them contains as minimum two operations with non-correlated result. Parallel computing starts performance increasing with two computers. Algorithms performance linearly depends on computer number and on objects’ number non-linearly for *k*-means and linearly on maximin. Multi-threading *k*-means performance linearly depends on classes number. The received results confirmed parallel *k*-means and maximin computing expediency that improves data classification and clustering quality.

Key words: *k*-means; maximin; parallel computing; algorithm; performance.

Современные задачи, решаемые с помощью IT-индустрии, связаны с большими объемами информации. При этом необходимо, чтобы поиск и обработка информации осуществлялись быстро и выдавалась она в доступной для понимания форме. Классификация и кластеризация – это формализованные задачи разделения объектов на множества [1, 2]. Их основу составляет процесс обучения, в задачу которого входит постепенное уточнение результата разделения объектов на группы. По количеству первоначальной информации алгоритмы распознавания делят на обучающиеся с учителем (контролируемое обучение) и самообучающиеся (неконтролируемое обучение).

Для контролируемого обучения совокупность классов известна заранее. Роль разработчика заключается в определении наилучших критериев классификации, учитывающих различия между признаками, характерными для отдельных классов. Главной задачей в этом случае становится поиск оптимальных методов разделения. В случае самообучения классы будем называть кластерами. Кластеры, составляющие обучающую выборку, заранее не известны и их требуется определить в ходе процедуры самообучения. Разделение объектов на группы в обоих случаях может оказаться сложной и продолжительной задачей, поэтому требуется найти эффективные способы ее решения.

Работа посвящена реализации алгоритмов автоматической классификации и кластеризации данных на основе параллельных вычислений.

В настоящее время широкое распространение получили многоядерные центральные процессоры, однако для таких аппаратных конфигураций актуальна проблема оптимизации вычислений, так как большинство программ до сих пор выполняется синхронно в рамках одного процесса [3]. В связи с этим желательно максимально использовать ресурсы компьютера, минимизировав время простоя всех ядер процессора.

Особенности параллельных вычислений

Эффекта от распараллеливания удастся добиться не в любом алгоритме. Многопоточное программирование лучше всего проявляет себя в операциях обработки данных, не требующих использования результатов предыдущих вычислений. Кроме того, важным условием является предотвращение конфликтов по чтению/записи данных.

Рассмотрим общий алгоритм организации параллельных вычислений. Пусть существует коллекция данных a размерности m. Необходимо для каждого элемента a_i вычислить новый элемент, используя функцию $f(a_i)$, и сформировать последовательность b размерностью m.

1. Основной поток программы получает на вход массив a и функцию f.
2. Основной поток программы запрашивает у операционной системы информацию о ресурсах: количестве установленных ядер/процессоров.
3. На основе каждого ядра/процессора создается поток.
4. Основной поток приостанавливает свою работу и передает управление пулу потоков.
5. Пул потоков выбирает поочередно все потоки и выдает каждому на обработку очередной элемент массива.
6. Пул ждет, пока какой-либо из потоков не освободится, чтобы загрузить его данными для обработки. Это продолжается, пока не закончатся элементы массива.
7. Основной поток программы объединяет результаты и готов выполнять следующие операции.

Распараллеливание алгоритмов

Для реализации параллельных вычислений в работе были выбраны два алгоритма группирования данных: *k-средних* – алгоритм классификации, *максимин* – алгоритм кластеризации.

Популярность алгоритма *k-средних* связана с его малой вычислительной сложностью и быстрой сходимостью к качественному решению. Вычислительная сложность метода *k-средних* равна $O(nkt)$, где n – размер набора данных (количество классифицируемых объектов), k – число классов, а t – число итераций, которые требуются алгоритму для приведения классов к стабильному состоянию.

Традиционный алгоритм *k-средних* обрабатывает n векторов, задающих исходные данные, пытаясь разделить их на k классов. Итеративно вычисляется центр масс (ядро) для каждого класса, полученного на предыдущем шаге, затем векторы разбиваются на классы вновь в соответствии с тем, какой из новых центров оказался ближе к каждому из векторов по выбранной метрике. Обычно метрикой близости выбирается евклидово расстояние. Алгоритм завершается, когда на какой-то итерации не происходит изменения классов. При обработке больших массивов данных алгоритм тратит большую часть времени на вычисление расстояний между всеми векторами данных и центрами классов. Очевидно, что нахождение расстояния от одного вектора до центра не связано с другими векторами. Поэтому распараллеливание вычислений является одним из очевидных усовершенствований.

В алгоритме *максимин* автоматическое устройство самостоятельно устанавливает кластеры, на которые делится исходное множество объектов. Для разделения данных необходимо определить критерии в условиях, когда не известны ни классы, ни их признаки, ни их количество. Поэтому процесс организуется так, чтобы среди всех возможных вариантов группировок найти такой, когда группы обладают наибольшей компактностью и при этом максимально отличны друг от друга. Необходимым минимумом информации для реализации таких алгоритмов являются данные для назначения словаря признаков объектов.

В алгоритме *максимин* исходные данные заданы n векторами, которые требуется разделить на кластеры, выполняя поиск представительных элементов каждого кластера, чье количество заранее не известно. На первом шаге случайным образом назначается центр первого кластера, затем максимально удаленный от него объект становится центром второго кластера. Оставшиеся объекты делятся на два кластера по критерию минимального расстояния до центра кластера. На следующем шаге в каждом кластере определяется максимально удаленный от центра объект и из найденных объектов выбирается максимум среди максимумов. Если это расстояние оказалось больше половины среднего расстояния между центрами всех построенных кластеров, то соответствующий ему объект становится центром очередного кластера. Изложенный процесс повторяется, пока не перестанет выполняться указанное условие [5].

В статье идея распараллеливания алгоритмов основывается на использовании процессоров с многоядерной архитектурой и реализацией алгоритма на языке C# платформы .NET 4.0 с применением Task Parallel Library, динамически масштабирующей степень параллелизма для наиболее эффективного использования всех доступных процессоров.

Для того чтобы распараллелить указанные алгоритмы, необходимо, чтобы в каждый момент времени могло выполняться сразу *n* операций, где *n* – количество ядер/процессоров, т. е. необходимо создать *n*-размерное множество асинхронных операций. При этом существенный эффект получится только в том случае, если это будет не набор единичных операций для каждого устройства, а набор последовательностей, так как при параллельной обработке данных над одной коллекцией все равно сохраняется риск простоя вычислителей, и в случае отсутствия данных для обработки возникает ситуация простоя из-за невозможности перехода к следующей логической операции в связи с необходимостью времени на синхронизацию результатов.

Очевидно, что наиболее эффективный способ распараллеливания этих алгоритмов – запуск пула асинхронных потоков-обработчиков для отдельной коллекции. При этом на итоговую производительность будет в значительной мере влиять количество синхронных операций.

Принципы распараллеливания алгоритма *k-средних*:

1. Все объекты классифицируются: планировщик, работая с объектами и потоками, поочередно выдает каждому свободному потоку новый объект, за счет чего удается увеличить производительность.

2. Когда определены все классы, каждый с набором своих объектов, планировщик может работать не с объектами и потоками, а с классами и потоками. Это позволяет выиграть в производительности, так как асинхронность операций проявляется на уровне целого класса, а не отдельного объекта [4].

Для алгоритма *максимин* применяется тот же принцип распараллеливания, что и для *k-средних*, однако здесь удается добиться асинхронности только на уровне объектов. Поэтому производительность для *k-средних* будет априори выше, поскольку в нем больше асинхронных операций.

Распараллеленный алгоритм *k-средних* будет выглядеть следующим образом:

1. Случайным образом генерируется *k* объектов-центров классов, где *k* – количество классов.

2. Каждый объект, который не является центром класса, относится к ближайшему ему центру. Для каждого объекта запускается отдельный поток, поэтому на данном шаге производительность повышается и распараллеливание приносит эффект.

3. Для каждого класса выполняется поиск нового центра. Новый поток запускается для каждого класса, что является лучшим решением, так как данные в каждом классе изолированы. И на данном шаге производительность увеличивается в *n* раз, где *n* – количество процессоров/ядер.

4. Выполняется проверка, изменился ли центр класса. Если да, то выполняется переход к шагу 2. Если нет, то классификация завершена.

Распараллеливание алгоритма *максимин*:

1. Случайным образом генерируется первый центр класса.

2. Создается второй центр класса, максимально удаленный от первого.

3. Для каждого объекта выбирается расстояние до ближайшего центра. Для каждого объекта выделяется отдельный поток, поэтому получается выигрыш в производительности на данном шаге.

4. Из минимальных расстояний выбирается максимальное.

5. Выполняется проверка, больше ли максимальное расстояние, чем половина среднего расстояния между существующими центрами. И если да, то новым центром становится объект, для которого было выбрано максимальное расстояние, и происходит переход к шагу 3. Иначе – кластеризация завершена. На данном шаге распараллеливание возможно только для операций поиска и вычисления среднего расстояния. Это дает небольшой прирост производительности.

Оценка результатов распараллеливания алгоритмов

В качестве объектов для последующего разбиения на множества были выбраны текстовые данные, так как это один из самых явных примеров использования алгоритмов *k-средних* и *максимина*. Для того чтобы абстрагироваться от конкретной предметной области, в текстах были выделены множества признаков и каждый текст был представлен в виде вектора. Далее, текст будет называться объектом, так как с точки зрения решаемой задачи на уровне алгоритмов работа ведется исключительно с объектами без привязки к какой-либо предметной области.

Оценка производительности проводилась для обоих алгоритмов в однопоточном и многопоточном режимах с различными наборами исходных данных. Количество объектов изменялось от 50 000 до 1 600 000; количество классов – от 2 до 16.

Исследования проводились на четырехъядерном процессоре семейства Core i5 без использования технологии *Hyper-Threating* (*HTT*) для одного, двух и четырех потоков в пуле.

На рис. 1 показано время выполнения алгоритма *k-средних* при различных количествах потоков, где ось абсцисс пронумерована стартами модуля классификации, а на оси ординат отложено количество секунд, затраченных на выполнение классификации.

Все старты можно разделить на четыре группы по следующему принципу. Количество классов в группе принимает значения 2, 4, 8 и 16. Для заданного числа классов алгоритм выполняется с количеством объектов 50 000, 100 000, 200 000, 400 000, 800 000.

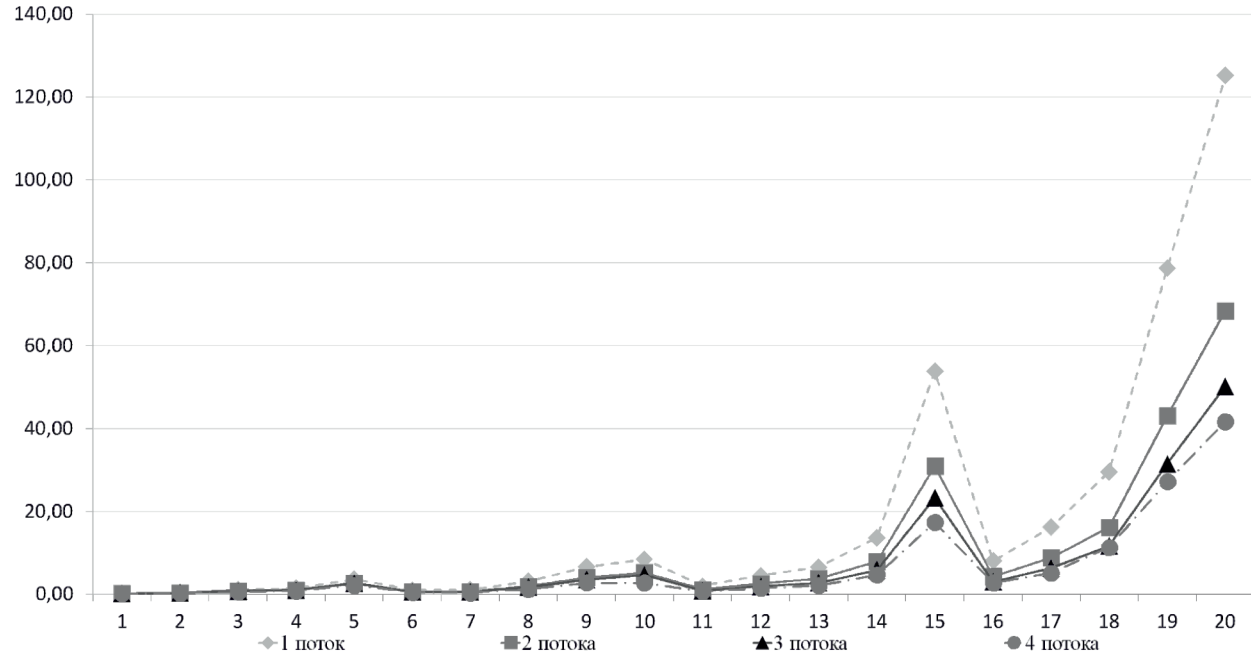

Рис. 1. Время классификации данных с помощью алгоритма *k-средних* при различных наборах исходных данных

В среднем алгоритм *k-средних* демонстрирует уменьшение времени работы программы на 66 % для двух потоков, на 104 % – для трех и на 159 % – для четырех потоков. Кроме того, на результат значительно влияет количество классов, именно этим обусловлены все «вспышки» на графике.

Первые восемь стартов модуля не показывают превосходства распараллеливания алгоритма, а однопоточный режим иногда даже превосходит многопоточный по производительности. Это обусловлено тем, что реализация параллельных вычислений требует выполнения некоторого дополнительного количества операторов, которое в каждом старте приблизительно одинаково. Поэтому результат проявляется только для относительно большого количества исходных данных.

На рис. 2 показаны результаты распараллеливания алгоритма *максимин*. На оси абсцисс приведены номера стартов модуля кластеризации, а на оси ординат отложено количество секунд, затраченных на выполнение кластеризации. Первый старт модуля выполнялся для 50 000 объектов, второй – для 100 000, третий – для 200 000, четвертый – для 400 000, пятый – для 800 000 и шестой – для 1 600 000 объектов.

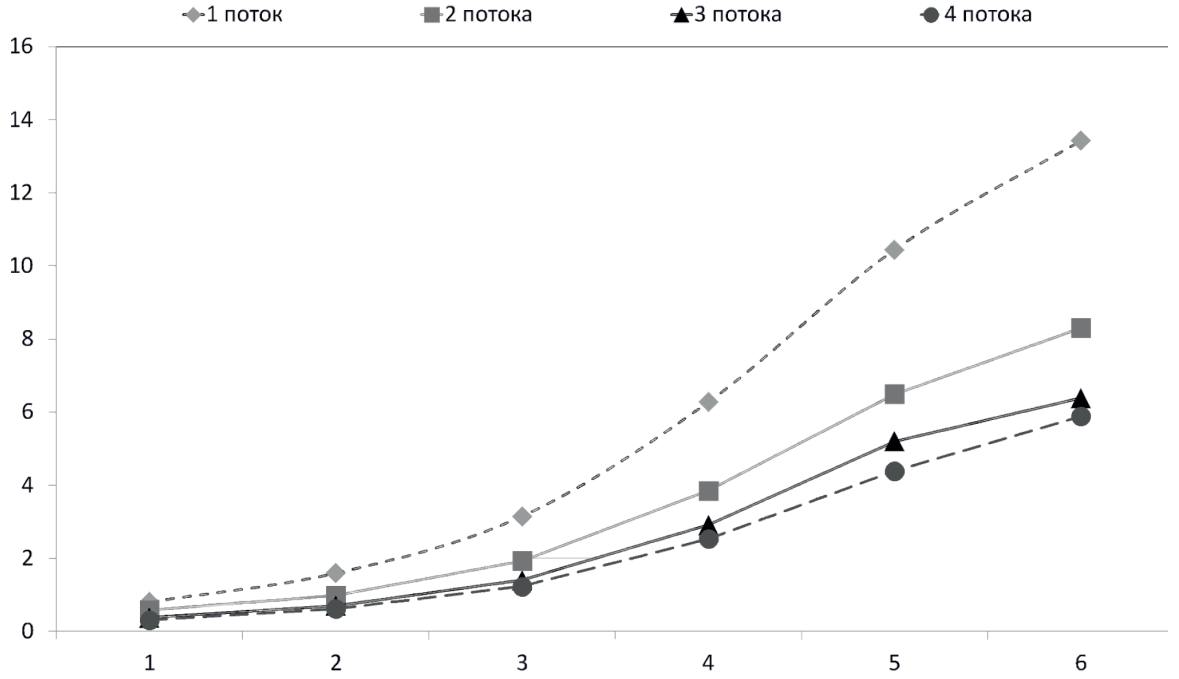

Рис. 2. Время кластеризации данных с помощью алгоритма *максимин* при различных наборах исходных данных

В среднем время кластеризации данных с помощью алгоритма *максимин* относительно одного потока уменьшилось на 58 % для двух потоков, на 115 % – для трех и на 147 % – для четырех потоков. Здесь с увеличением количества объектов увеличивается и эффект от распараллеливания вычислений.

* * *

Исследования алгоритмов *k-средних* и *максимина* на предмет возможности их выполнения в параллельном режиме позволили сформулировать следующие заключения.

1. Оба алгоритма поддаются распараллеливанию, поскольку в каждом из них существует минимум две операции с некоррелирующими результатами.

2. Распараллеливание вычислений демонстрирует уменьшение времени выполнения алгоритмов уже при двух процессорах.

3. Увеличение производительности алгоритмов линейно зависит от увеличения числа вычислителей.

4. С увеличением количества объектов классификации увеличивается производительность параллельных вычислений. Причем для алгоритма *k-средних* эта зависимость нелинейная, а для алгоритма *максимин* – линейная.

5. С увеличением количества классов в алгоритме *k-средних* линейно увеличивается производительность параллельных вычислений.

Таким образом, полученные результаты подтвердили целесообразность распараллеливания вычислений в алгоритмах *k-средних* и *максимина*, что увеличивает эффективность классификации и кластеризации данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Серебряная Л. В. Метод автоматической классификации текстовой информации по образцу / Л. В. Серебряная, С. В. Чебаков // Информационные системы и технологии: материалы VI Междунар. конф. (Минск, 24–25 нояб. 2010 г.). Минск, 2010. С. 244–247.

2. Серебряная Л. В., Чебаков С. В. Методы автоматической классификации и кластеризации текстовой информации // Информатизация образования. 2011. № 2. С. 52–61.

3. Таненбаум Э., Стеен М. ван. Распределенные системы: принципы и парадигмы. СПб., 2003. (Классика computer science).

4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4: пер. с англ. 5-е изд. М., 2011.

5. Theodoridis S., Koutroumbas K. Pattern Recognition. 4th edition. Athens, 2009.

Поступила в редакцию 26.03.13.

Федор Игоревич Третьяков – аспирант кафедры программного обеспечения информационных технологий Белорусского государственного университета информатики и радиоэлектроники. Научный руководитель – Л. В. Серебряная.

Лия Валентиновна Серебряная – кандидат технических наук, доцент кафедры программного обеспечения информационных технологий Белорусского государственного университета информатики и радиоэлектроники.

УДК 517.95

Ф. Е. ЛОМОВЦЕВ, А. В. МОТЕВИЧ

ДВУМЕРНАЯ ЗАДАЧА ГУРСА ДЛЯ ПОЛНЫХ ДИФФЕРЕНЦИАЛЬНО-ОПЕРАТОРНЫХ УРАВНЕНИЙ ВТОРОГО ПОРЯДКА С ПЕРЕМЕННЫМИ ОБЛАСТЯМИ ОПРЕДЕЛЕНИЯ ОПЕРАТОРОВ

Доказана корректность по Адамару задачи Гурса для полных гиперболических дифференциально-операторных уравнений с двумерным параметром (временем) при зависящих от времени областях определения зависящих от времени неограниченных операторных коэффициентов. Доказательство корректности осуществляется путем обобщения и развития известного метода энергетических неравенств. Единственность сильных решений этой задачи вытекает из энергетического неравенства, выведенного с помощью абстрактных сглаживающих операторов Иосиды – Ломовцева. Для доказательства существования ее сильных решений устанавливается плотность множества значений задачи Гурса новой для задач Гурса техникой сопряженных операторов к произведению операторов. Эта техника основана на лемме Ломовцева о сопряженном операторе к произведению ограниченного и неограниченного операторов. Приведен пример новой корректной частично характеристической краевой задачи для гиперболического уравнения в частных производных второго порядка с двумерным временем при условиях типа Гурса и зависящих от одной из временных переменных граничных условиях по пространственной переменной.

Ключевые слова: задача Гурса; сильное решение; корректность по Адамару; переменная область определения; частично характеристическая краевая задача.

The correctness by Hadamard of the Goursat problem for complete hyperbolic operator-differential equations with respect to two-dimensional parameter (time) with time-dependent domains of time-dependent unbounded operator coefficients is proved. Proof of the correctness is performed by means of generalization and development of well-known method of energy inequalities. The uniqueness of strong solutions of this problem follows from the energy inequality derived with the help of abstract smoothing Yosida – Lomovtsev's operators. In order to prove the existence of its strong solutions the density of region of the Goursat problem is established by means of new technique of conjugates to product of bounded and unbounded operators at first for Goursat problems. This technique is based