УДК 004.421

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ АНАЛИЗА РЫНОЧНОЙ КОРЗИНЫ НА ПРОЦЕССОРАХ CELL

К. С. Пан, М.Л. Цымблер

A PARALLEL ALGORITHM FOR MARKET BASKET ANALYSIS ON THE CELL PROCESSOR

C.S. Pan, M.L. Zymbler

В работе рассматривается задача глубинного анализа данных — задача нахождения часто втречающихся наборов товаров. Предложен параллельный алгоритм, адаптированный для вычислительных систем на базе процессоров с архитектурой Cell Broadband Engine. Представлены результаты вычислительных экспериментов, показывающие эффективность предложенного алгоритма.

Ключевые слова: глубинный анализ данных (data mining), анализ рыночной корзины, Cell

The paper is devoted to solving market-basket analysis problem of data mining. We describe a parallel algorithm adapted for the Cell Broadband Engine architecture. The experimental results demonstrate the effectiveness of the proposed algorithm.

Keywords: data mining, market-basket problem, Cell

Введение

Задача *анализа рыночной корзины (market-basket problem)* заключается в нахождении всех наборов (множеств) товаров, которые часто приобретаются совместно [?]. Для формального описания задачи и алгоритма ее решения в работе используются следующие термины и обозначения.

Корзина (basket) — набор товаров, приобретенных совместно (в рамках одной покупки). Обозначим за *B (baskets)* множество анализируемых корзин, а за J *(items)* — множество всех товаров, т.е. $I = \bigcup_{b \in B} b$.

Опорное число (support) заданного набора товаров X — количество корзин во множестве Л, каждая из которых содержит данный набор товаров X, т.е. $support(X, B) = card\{b \in B : X \subset b\}$.

Обозначим за s_{min} минимальное значение опорного числа, при котором набор товаров считается часто встречающимся. Обозначим за L *(large itemsets)* множество часто встречающихся наборов товаров, т.е. $L = \{l \subset I : support(l, B) \geqslant s_{min}\}$. Множество L содержит все наборы товаров, опорное число которых не меньше s_{min}.

В соответствии со введенными обозначениями задача анализа рыночной корзины формулируется следующим образом: для заданных множества B и числа s_{min} найти множество L.

Далее мы рассмотрим последовательный алгоритм решения задачи анализа рыночной корзины, а затем на его основе построим параллельный алгоритм, адаптированный для вычислительных систем на базе процессоров с архитектурой IBM Cell Broadband Engine.

1. Последовательный алгоритм Apriori

Алгоритм анализа рыночной корзины на процессорах Cell построен на основе алгоритма Apriori, предложенного в **[1].** Идея алгоритма Apriori состоит в использовании свойства *антимонотонности* опорного числа, которое заключается в следующем: опорное число множества товаров не превосходит опорного числа любого его подмножества, т.е.

$$\forall \gamma \subset c \;\; support(\gamma, B) \geqslant support(c, B).$$

В описании алгоритма Apriori используются следующие дополнительные термины и обозначения.

Кандидат c — набор товаров, для которого в ходе выполнения алгоритма выдвигается и проверяется гипотеза $c \in L$. На k-**м** шаге выполнения алгоритма вычисляются два множества: C_k и L_k.

C_k *(candidate k-itemsets)* — множество кандидатов длины k, где *длина кандидата* — это количество элементов в нем.

Каждый кандидат из C_k, опорное число которого не меньше s_{min}, попадает во множество L_k — множество часто встречающихся наборов товаров длины k, т.е.

$$L_k = \{c \in C_k : support(c, B) \geqslant s_{min}\}.$$

В ходе выполнения алгоритма вычисляются все множества L_k, объединение которых по экончании вычислений дает множество L, т.е. $L = \bigcup_k L_k$.

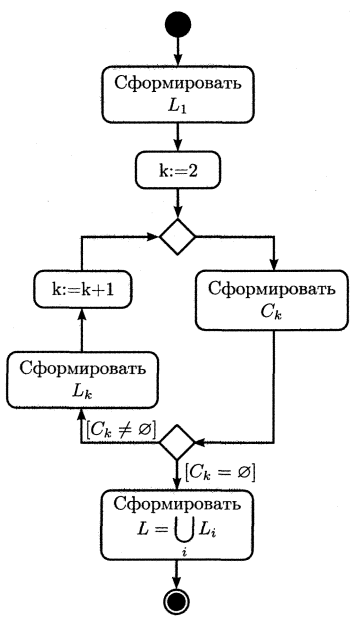


Рис. 1. Последовательный алгоритм Apriori

Последовательный алгоритм Apriori, представленный на рис. 1, кратко может быть описан следующим образом.

На первом шаге алгоритма формируется множество одноэлементных кандидатов C_k $(k = 1)$. Для этого производится перебор всех корзин B, в ходе которого каждый встреченный товар рассматривается в качестве одноэлементного кандидата и увеличивается его опорное число. Из всего множества полученных кандидатов выбираются те, опорное число которых не меньше s_{mim} и обозначаются как L_1.

На втором шаге алгоритма из часто встречающихся наборов товаров длины k (L_k) формируется множество кандидатов с большей на единицу длиной (C_{k+1}). Если на данном шаге множество C_{k+1} получилось пустым, то выполнение алгоритма завершается. Вычисление C_{k+1} на основе L_k осуществляется с помощью суперпозиции операций *self join* и *prune*.

Если представить множество L_k в виде реляционного отношения, имеющего атрибуты $item_1$, $item_2$, ...$item_k$, то операция *self join* представляет собой реляционную операцию Θ-соединения отношения L_k с самим собой: $selfjoin(L_k) = L_k \underset{\Theta}{\bowtie} L_k$, где Θ — это условие $(p.item_1 = q.item_1) \wedge (p.item_2 = q.item_2) \wedge \cdots \wedge (p.item_{k-1} = q.item_{k-1}) \wedge (p.item_k < q.item_k)$.

Операция *prune (отсечение)* заключается в сокращении множества кандидатов путем отбрасывания тех из них, у которых хотя бы одно подмножество не входит в L_k, т.е. $prune\ (C) = \{c \in C : \forall \hat{c} \subset c \quad \hat{c} \in L_k\}$.

Таким образом, $C_{k+1} = prune(selfjoin(L_k))$.

Третий шаг алгоритма заключается в том, что для новых кандидатов C_{k+1} вычисляются опорные числа. Для этого производится перебор всех корзин J3, в ходе которого каждый кандидат из C_{k+1} проверяется на вхождение в каждую корзину. Выбираются все кандидаты с опорным числом не меньше s_{min} и обозначаются как множество L_{k+1}. После этого k увеличивается на единицу и выполнение алгоритма продолжается со второго шага.

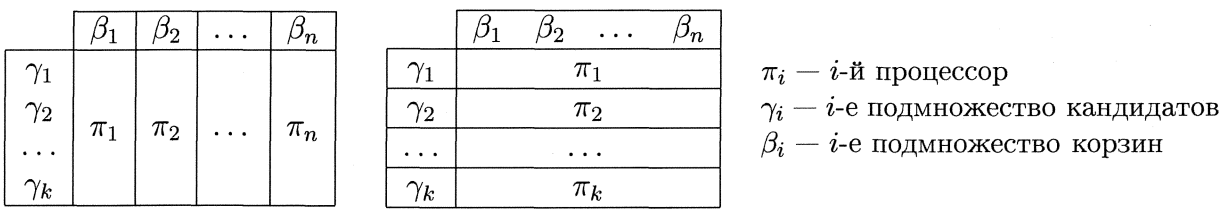
Вычисление опорных чисел кандидатов является наиболее затратной вычислительной операцией алгоритма и требует $|C_k| \cdot |B|$ операций проверки вхождения кандидата в корзину, т.е. всего $\sum_{i=1}^{|C_k|} \sum_{j=1}^{|B|} |c_i| \cdot |b_j|$ операций сравнения, где $c \in C_k$ и $b \in B$.

2· Работы по тематике исследования

В соответствии с архитектурой Cell Broadband Engine (Cell BE) [2], процессор Cell представляет собой асимметричный многоядерный процессор, состоящий из одного управляющего ядра (Power Processing Element, PPE) и восьми вычислительных ядер (Synergistic Processing Element, SPE), которые поддерживают набор векторных инструкций (SIMD-функций) для обработки 128-битных векторов.

В работе [3] рассматривается построение параллельных алгоритмов Data Mining для решения задач кластеризации и классификации на процессорах Cell.

В настоящее время распараллеливание алгоритма Apriori осуществляется с помощью следующих основных подходов [4]: Count Distribution и Data Distribution. Распределение данных по вычислительным узлам, используемое в этих подходах, показано на рис. 2.



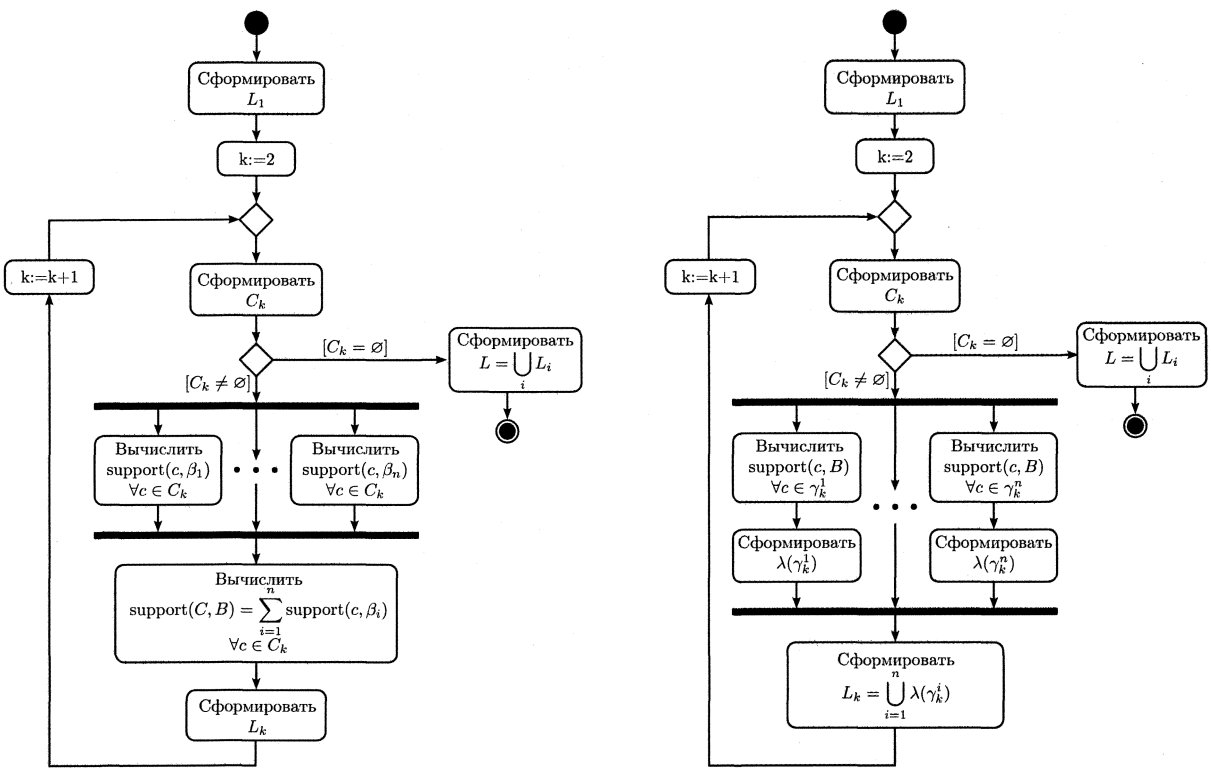
а) Подход Count Distribution б) Подход Data Distribution

Рис. 2. Подходы к распараллеливанию алгоритма Apriori

Подход *Count Distribution* [5] заключается в том, что множество корзин B разбивается на подмножества $\beta_1, \beta_2, \ldots, \beta_n$. Каждое множество β_i обрабатывается на своем процессоре π_i.

Диаграмма деятельности, которая иллюстрирует подход Count Distribution, приведена на рис. 3а. Данный подход реализован для Cell в работе [6].

Подход *Data Distribution* [5] предполагает, что множество кандидатов C_k разбивается на подмножества $\gamma_k^1, \gamma_k^2, \ldots, \gamma_k^n$. Каждое множество γ_k^i обрабатывается своим процессором π_i. Диаграмма деятельности, которая иллюстрирует данный подход, представлена на рис. 3б. На данной диаграмме введена дополнительная функция $\lambda(\gamma)$, которая обозначает множество часто встречающихся наборов из подмножества кандидатов γ, т.е. $\lambda(\gamma) = \{c \in \gamma : support(c, B) \geqslant s_{min}\}$



а) Подход Count Distribution б) Подход Data Distribution

Рис. 3. Подходы к распараллеливанию алгоритма Apriori

В настоящее время подход Data Distribution, насколько нам известно, не реализован для архитектуры Cell BE.

3. Параллельный алгоритм для Cell

На основе подхода *Data Distribution* нами был разработан алгоритм *DDCapriori,* который реализует параллельный анализ рыночной корзины на процессорах Cell. В изложении алгоритма мы исходим из допущения, что анализируемое множество корзин может быть целиком размещено в оперативной памяти.

3.1. Проектирование

В алгоритме DDCapriori используется модель «мастер-рабочие». Нить-мастер запускается на управляющем ядре PPE и распределяет задания для рабочих. Нити-рабочие запускаются на вычислительных ядрах SPE и выполняют обработку данных, получаемых от мастера.

На рис. 4 представлены диаграммы деятельности мастера и рабочего, описывающие алгоритм DDCapriori. Для упрощения записи нами введены операции Send, Recv и MakeTask.

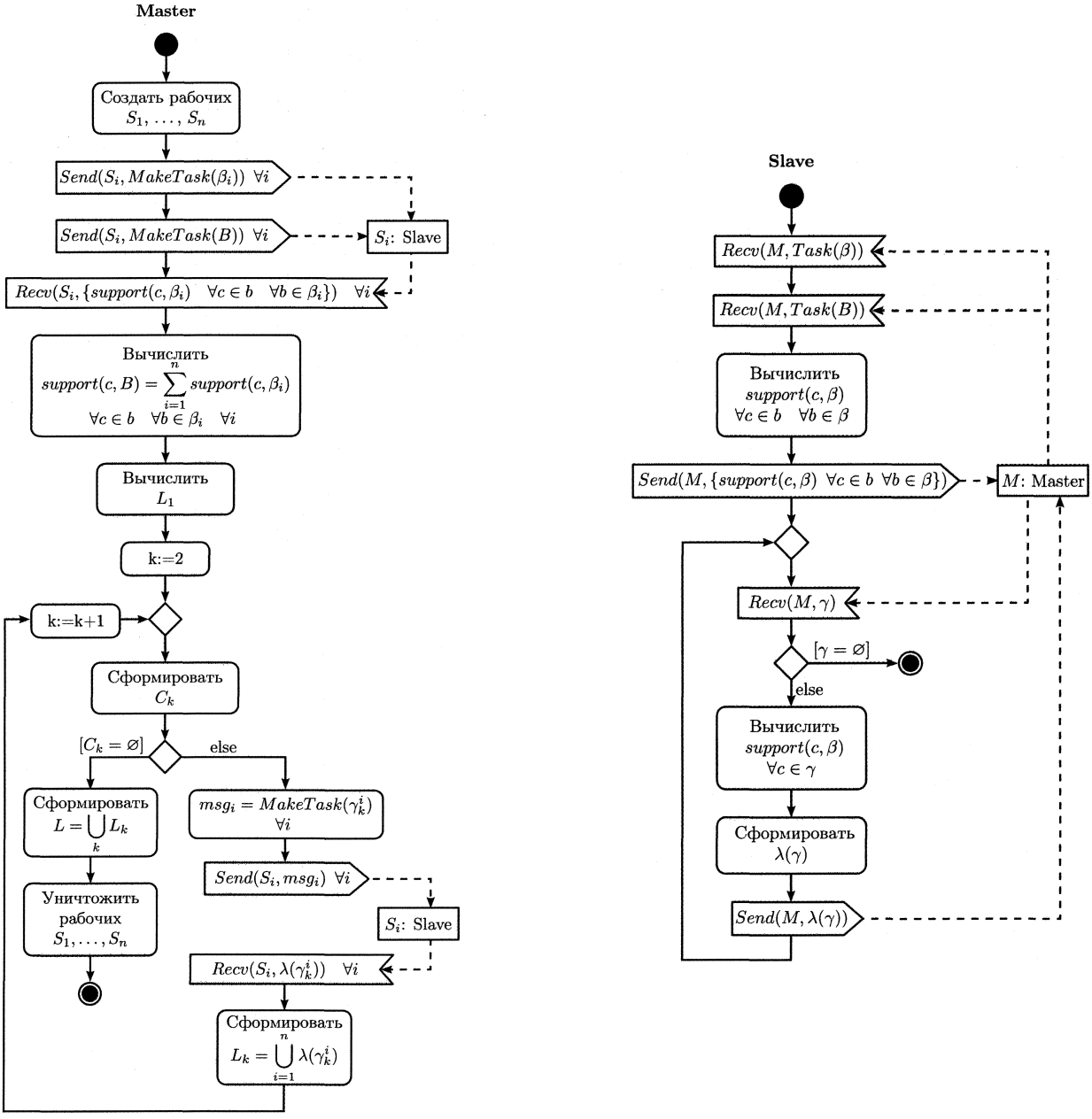


Рис. 4. Диаграммы деятельности мастера и рабочего

Процедура *Send(dst,msg)* выполняет асинхронную отправку сообщения *msg* получателю *dst.*

Процедура *Recv(src,msg)* выполняет синхронное получение сообщения *msg* от отправителя *src.*

Функция *MakeTask(d)* создает и возвращает задание на обработку данных *d. Задание* представляет собой совокупность адреса, по которому располагаются данные в оперативной памяти, и размера этих данных. В качестве данных может выступать множество корзин либо множество кандидатов.

Идея предлагаемого параллельного алгоритма заключается в том, чтобы возложить на мастера задачу формирования множеств C_k и L_k., а на рабочих — вычисление опорных чисел для кандидатов из C_k.

В отличие от подхода *Data Distribution,* при вычислении множества L_1 множество корзин *B* разбивается на подмножества, которые затем назначаются для обработки разным рабочим. Рабочий рассматривает каждый товар в своем подмножестве корзин как одноэлементный кандидат и увеличивает его опорное число всякий раз, когда этот кандидат встречается в корзинах. При вычислении множества L_k $(k > 1)$ на рабочих распределяются кандидаты, а не корзины.

Деятельность мастера кратко может быть описана следующим образом. После создания рабочих мастер отправляет каждому из них первое задание и все множество корзин, после чего переходит в состояние ожидания. По получении результатов мастер выполнянет их агрегацию, формируя таким образом множество кандидатов единичной длины, и отсечение редко встречающихся кандидатов, формируя множество часто встречающихся наборов товаров единичной длины. Далее мастер полагает счетчик κ равным 1 и циклически выполняет следующую последовательность действий. Из множества часто встречающихся наборов товаров, имеющих длину k, формируется множество кандидатов длины $\kappa + 1$. Затем мастер формирует задания на обработку полученного множества кандидатов, отправляет их рабочим и ожидает от них результаты вычислений (опорные числа). Если мастеру не удается сформировать кандидаты длины $k + 1$, то мастер прерывает цикл, уничтожает рабочих и вычисляет результирующее множество.

Деятельность рабочего состоит в следующем. Получив задание от мастера, рабочий формирует множество кандидатов единичной длины из своего подмножества корзин, после чего отправляет результаты мастеру. Далее рабочий циклически выполняет следующую последовательность действий: ожидание от мастера подмножества кандидатов, вычисление опорных чисел и отправка результата вычислений мастеру. Цикл прерывается, если получено задание на обработку пустого множества кандидатов.

3.2. Реализация

Диаграмма классов, реализующих предложенный алгоритм, представлена на рис. 5.

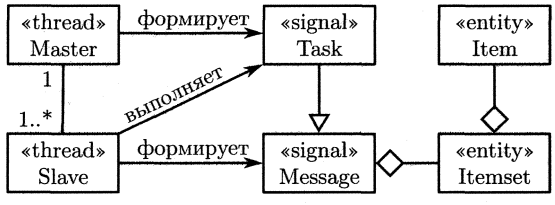


Рис. 5. Диаграмма классов, реализующих алгоритм DDCapriori

Класс *Master* реализует нить-мастер и выполняет следующие основные функции: управление рабочими и формирование множеств C_k и L_k. Экземпляр класса Master исполняется на управляющем ядре PPE.

Класс *Slave* реализует нить-рабочего и выполняет расчет опорных чисел для кандидатов из множества C_k. Экземпляры класса Slave создаются экземпляром класса Master на вычислительных ядрах SPE (по одному на каждом вычислительном ядре).

Класс *Task* служит для управления рабочими, выполняет роль сигнала и хранит входные данные для рабочего. Класс *Message* выполняет роль сигнала, отправляемого мастеру рабочим, и хранит результаты вычислений рабочего.

Помимо увеличения производительности параллельного анализа рыночной корзины по сравнению с последовательным, мы получаем дополнительный прирост производительности

за счет использования векторных функций Cell **[8]** при вычислении опорных чисел кандидатов.

Процессор Cell оперирует векторами длиной 128 битов. В зависимости от длины идентификатора товара, в одном векторе могут быть размещены от 16 до 2 целочисленных идентификаторов. В нашей реализации используются 32-битные идентификаторы, то есть в одном векторе размещаются 4 идентификатора.

При проверке вхождения кандидата в корзину $c \subset b$ кандидат c и корзина b разбиваются на вектора $\vec{c}_i$ и $\vec{b}_j$. Каждый из векторов кандидата сравнивается с каждым вектором корзины с помощью последовательности векторных операций, которая показана на рис. 6.

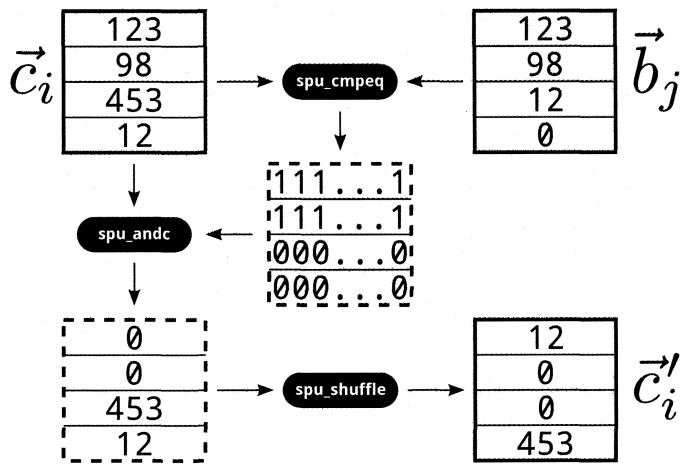


Рис. 6. Операция проверки вхождения кандидата в корзину

С помощью функции сравнения векторов *spu__cmpeq* формируется вектор-маска, в котором биты элемента установлены, если соответствующие элементы векторов $\vec{c}_i$ и $\vec{b}_j$ равны, и сброшены в противном случае. Далее к исходному вектору кандидата $\vec{c}_i$ и полученной маске применяется векторная функция *spu_andc,* которая выполняет побитовую операцию $A \wedge \neg B$, обращая в нуль все элементы исходного вектора кандидата $\vec{c}_i$, совпадающие с соответствующим элементом вектора корзины $\vec{b}_j$. Затем с помощью векторной функции *spu_shuffle* выполняется циклический сдвиг полученного вектора на 1 элемент.

Данная процедура, примененная N раз, где N — длина вектора, позволяет обнулить все элементы вектора $\vec{c}_i$, которые содержатся в векторе $\vec{b}_j$.

Таким образом, после применения этой процедуры ко всем парам векторов из кандидата и корзины в кандидате останутся только те элементы, которые не входят в корзину. Если таких элементов не осталось, то кандидат входит в корзину.

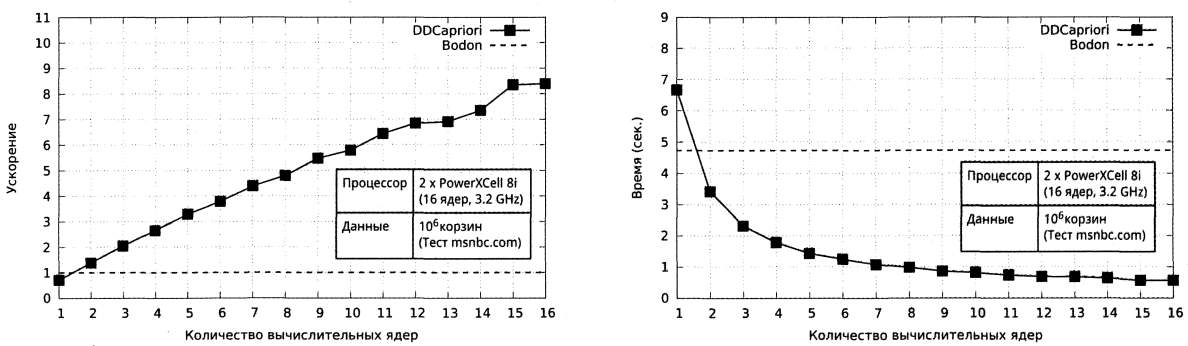
Поскольку каждый вектор кандидата проверяется на вхождение в каждый вектор корзины, то для вычисления опорных чисел кандидатов из множества C_k потребуется $\underline{O}\left(\sum_{i=1}^{|C_k|}\sum_{j=1}^{|B|}|c_i|\cdot|b_j|\right)$ векторных операций. Заметим, что в случае, когда векторные функции не используются, для выполнения проверки требуется такое же количество скалярных операций, скорость выполнения которых на вычислительных ядрах SPE существенно ниже **[2].**

4. Вычислительные эксперименты

Для оценки эффективности разработанного алгоритма нами были проведены три серии вычислительных экспериментов. В качестве исходных данных экспериментов был взят

стандартный тестовый набор данных о посещении страниц web-сайта msnbc. com, который также использовался для оценки эффективности алгоритмов Data Mining, предложенных в [10]. Множество B в тестовой задаче представляет собой записи о посещениях страниц сайта. Каждая запись содержит отметку о том, к какой семантической категории принадлежат посещенные за один сеанс страницы. В экспериментах осуществляется поиск наборов категорий страниц, часто посещаемых совместно (в течение одной сессии пользователя).

В первой серии экспериментов мы определили время работы и ускорение алгоритма в зависимости от количества задействованных вычислительных ядер. Результаты экспериментов представлены на рис. 7. При вычислении ускорения за единицу принята производительность последовательного алгоритма, предложенного в [9], который в настоящее время считается одним из лучших последовательных алгоритмов решения задачи анализа рыночной корзины [4-7]. Эксперименты показывают, что алгоритм DDCapriori демонстрирует ускорение, близкое к линейному.



а) Ускорение б) Время анализа

Рис. 7. Производительность алгоритма DDCapriori

Кроме того, мы сравнили масштабируемость разработанного алгоритма и алгоритма Count Distribution для Cell, используя результаты экспериментов, опубликованные авторами этого алгоритма в [6]. Сравнение показывает (см. рис 8) несколько лучшую масштабируемость разработанного алгоритма.

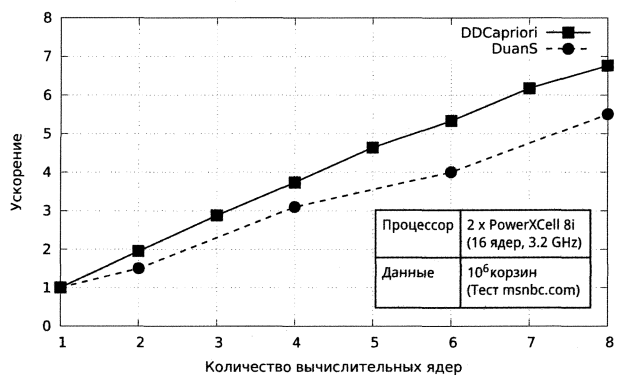
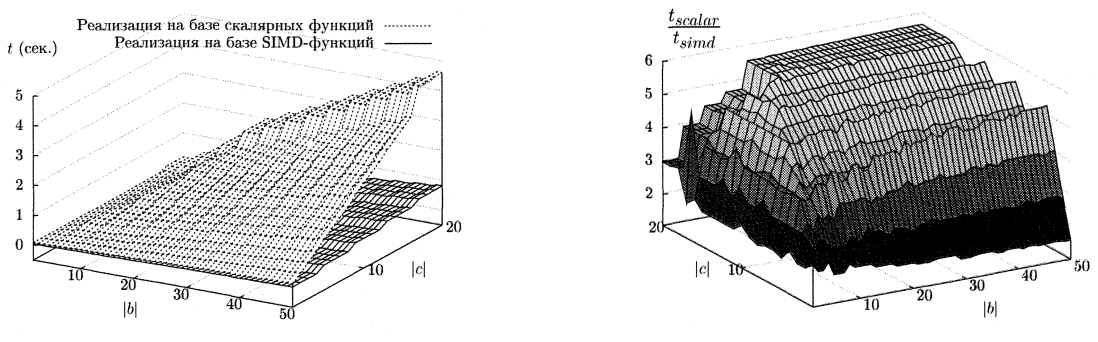


Рис. 8. Сравнение масштабируемости DataDistribution и CountDistribution

Во второй серии экспериментов мы исследовали выигрыш от применения векторных операций вместо скалярных при проверке вхождения канндидата в корзину в зависимости от длины кандидата и корзины. Результаты данной серии экспериментов представлены на

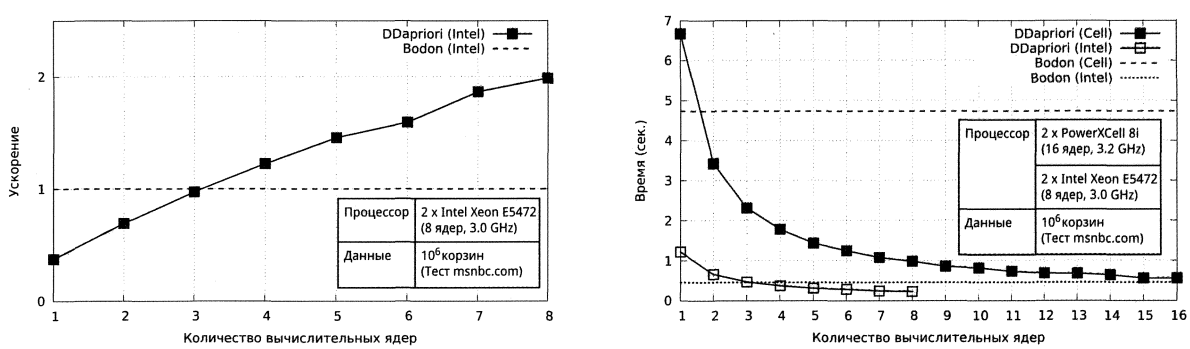
рис. 9. Результаты экспериментов показывают, что выигрыш от использования векторных операций прямо пропорционален длинам кандидата и корзины.



а) Время проверки б) Выгода от использования векторных функций

Рис. 9. Использование векторных функций при проверке вхождения кандидата в корзину

Третья серия экспериментов была направлена на сравнение производительности разработанного алгоритма на платформах Cell и Intel. Для проведения этих экспериментов нами была разработана реализация алгоритма DDCapriori, ориентированная на процессоры Intel. В данной реализации вместо SPE-нитей используются POSIX-нити и не используются векторные функции. Результаты экспериментов предствлены на рис. 10.



а) Ускорение на Intel б) Время

Рис. 10. Сравнение быстродействия на процессорах Cell и Intel

Эксперименты показывают, что на процессорах Cell алгоритм демонстрирует несколько лучшее ускорение, чем на процессорах Intel. Однако процессоры Intel обеспечивают существенно более высокое быстродействие, чем процессоры Cell.

5. Заключение

В работе представлен параллельный алгоритм решения задачи анализа рыночной корзины, адаптированный для вычислительных систем на базе процессоров Cell. Параллелизм достигается путем разделения множества кандидатов на подмножества и распределения этих подмножеств по вычислительным ядрам. При этом множество корзин передается целиком на каждое вычислительное ядро.

В реализации использована модель «мастер-рабочие». Нить-мастер запускается на управляющем ядре PPE и выполняет управление рабочими и формирование кандидатов и множеств часто встречающихся наборов на каждом шаге алгоритма. Нити-рабочие запускаются на вычислительных ядрах SPE и выполняют вычисление опорных чисел кандидатов.

Реализация выполнена на языке программирования C с использованием векторных функций библиотеки IBM Cell Broadband Engine SDK, которые позволяют эффективно реализовать наиболее затратную операцию вычисления опорных чисел.

Представлены результаты вычислительных экспериментов, показывающие эффективность предложенного алгоритма.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 09-0700241-а).

Литература

1. Agrawal, R. Mining Association Rules between Sets of Items in Large Databases / R. Agrawal, T. Imielinski, A.N. Swami // Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. R 207-216.

2. IBM Corporation. Cell Broadband Engine technology. URL: http://www.alphaworks.ibm.com/topics/cell (дата обращения: 01.06.2009).

3. Buehrer, G. Data Mining on Cell Broadband Engine / G. Buehrer, S. Parthasarathy, M. Goyder // Proceedings of the 22nd International Conference on Supercomputing. 2008. P. 26-35.

4. Zaki, M.J. Parallel and Distributed Association Mining: A Survey // IEEE Concurrency. October 1999. Vol. 7. No. 4. P. 14-25.

5. Zaki, M.J. Parallel data mining for association rules on shared-memory multi-processors / M.J. Zaki, M. Ogihara, S. Parthasarathy, W. Li // Proceedings of the 1996 ACM/IEEE conference on Supercomputing. 1996. Article No. 43.

6. Duan, R. Data Mining Algorithms on the Cell Broadband Engine / R. Duan, A. Strey // Proceedings of the 14th International Euro-Par Conference. 2008. P. 665-675.

7. Han, S. Scalable Parallel Data Mining for Association Rules / S. Han, G. Karypis, V. Kumar // IEEE Transactions on Knowledge and Data Engineering. Vol. 12. Issue 3. P. 337-352.

8. IBM Cell Broadband Engine SDK, Version 3.0 documentation. URL: http://www-01.ibm.com/chips/techlib/techlib.nsf/products/IBM_SDK_for_Multicore_Acceleration (дата обращения: 01.06.2009).

9. Bodon, F. A fast APRIORI implementation // Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMP03). Vol. 90. Melbourne. 2003.

10. Visualization of Navigation Patterns on a Web Site Using Model Based Clustering. Technical Report MSR-TR-00-18. Microsoft Research. 2000. / I. Cadez, D. Heckerman, C. Meek, at al. URL: http://research.microsoft.com/pubs/69752/tr-2000-18.pdf (дата обращения: 13.12.2009).

Кафедра системного программирования,
Южно-Уральский государственный университет
kvapen@gmail.com

Поступила в редакцию 22 марта 2010 г.