# ТЕХНОЛОГИИ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ СОВРЕМЕННЫХ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ

УДК 004.272.2

**Александр Валериевич Бакулев,**
к.т.н., доцент, каф. Информационных технологий Рязанского филиала МЭСИ
Тел.: (953) 739-98-45
Эл. почта: contact-bav@mail.ru

**Марина Алексеевна Бакулева,**
к.т.н., доцент, каф. Информационных технологий Рязанского филиала МЭСИ
Тел.: (920) 960-98-47
Эл. почта: marina.bakuleva@gmail.com

**Максим Александрович Козлов,**
аспирант, каф. Информационных технологий Рязанского филиала МЭСИ
Тел.: (920) 634-35-07
Эл. почта: dreamx.max@gmai.com

**Сергей Владимирович Скворцов,**
д.т.н., профессор, каф. Информационных технологий Рязанского филиала МЭСИ
Тел.: (920) 631-49-98
Эл. почта: s.v.skvor@gmail.com

Рассмотрены возможности перспективных технологий программирования для многоядерных процессоров, направленные как на автоматическую адаптацию существующего последовательного программного обеспечения в рамках императивного подхода, так и на разработку многопоточных приложений средствами современных функциональных языков.

***Ключевые слова:*** *многоядерные процессоры, параллельные алгорит-мы, спекулятивная многопоточность, функциональная парадигма.*

**Aleksandr V. Bakulev,**
PhD in Technical Sciences, Associate Professor MESI, Ryazan branch
Tel.: (953) 739-98-45
E-mail: contact-bav@mail.ru

**Marina A. Bakuleva,**
PhD in Technical Sciences, Associate Professor MESI, Ryazan branch
Tel.: (920) 960-98-47
E-mail: marina.bakuleva@gmail.com

**Maksim A. Kozlov,**
Post-graduate student
Tel.: (920) 634-35-07
E-mail: dreamx.max@gmai.com

**Sergei V. Skvortsov,**
Doctorate of Technical Sciences, Professor MESI, Ryazan branch
Tel.: (920) 631-49-98
E-mail: s.v.skvor@gmail.com

**TECHNOLOGY DEVELOPMENT OF PARALLEL PROGRAMS FOR MODERN MULTICORE PROCESSORS**

The possibilities of advanced technologies for programming the multi-core processors, aimed at both the automatic adaptation of existing sequential soft-ware under the mandatory approach, and to develop accurate multipolar applications by means of modern functional languages.

***Keywords:*** *multi-core processors, parallel algorithms are, speculative multi-threading, functional paradigm.*

## 1. Введение

Развитие технологий программирования для персональных компьютеров с точки зрения *повышения производительности программ* традиционно происходило в основном экстенсивными методами. По большей части разработчики программного обеспечения (ПО) надеялись на постоянное совершенствование аппаратных ресурсов компьютера. Очередная замена процессора с увеличенной тактовой частотой позволяла получить ускорение работы программ без малейшего их изменения.

В настоящее время достигнут фактический предел роста тактовой частоты процессора, и дальнейшее увеличение его производительности становится возможным только за счет перехода к многоядерной архитектуре, по существу к параллельной архитектуре. На сегодняшний день уже широко распространены процессоры с четырьмя и более ядрами, а в перспективе количество ядер может измеряться сотнями. Подобная технологическая революция в архитектуре компьютера предоставляет существенный потенциал для роста производительности ПО, однако для реализации этого потенциала необходимы столь же серьезные революционные изменения современных технологий программирования.

Параллельные программы должны обладать необычными динамическими свойствами (масштабируемость на все доступные ресурсы, балансировка загрузки ядер, анализ зависимостей по данным и управлению и др.), без которых обходятся последовательные программы. В результате нынешнее общесистемное ПО и системы программирования оказались плохо приспособленными для создания параллельных программ, трудоёмкость их написания оказалась чрезвычайно велика. Более того, программы с динамическими свойствами, как известно, могут бесконечно долго отлаживаться. Существуют задачи, которые сравнительно просто запрограммировать для исполнения на многоядерных процессорах. Однако как показывает практика, не так много в мире разработано алгоритмов, пригодных для хорошей параллельной реализации. Все это требует разработки новых подходов к параллельному программированию, которые направлены на упрощение и повышение эффективности работы программиста.

Массовое внедрение многоядерных технологий возродило интерес к функциональной парадигме программирования, а также к разработке новых языков и техник для обработки данных. Актуальной становится и задача адаптации огромного объема существующего последовательного ПО для эффективного выполнения в параллельной вычислительной среде.

С учетом изложенного *целью* данной работы является обсуждение возможных подходов к разработке многопоточных приложений для современных многоядерных процессоров как в рамках императивного подхода, так и с использованием функциональной парадигмы. В частности, для автоматического распараллеливания существующих последовательных программ предлагается использовать *метод спекулятивной многопоточности*, а при разработке новых параллельных алгоритмов и приложений – *свойства новых языков функционального программирования*, которые совмещают удобство разработки и физический параллелизм исполнения.

## 2. Обзор известных методов и подходов

Вопросы параллельного программирования, а также автоматизации распараллеливания последовательных программ для определенного класса параллельных архитектур имеют уже долгую историю [1–3].

Проблема заключается в том, что, как правило, решались они в привязке к определенным типам архитектур параллельных компьютеров. Подобный подход требовал создания специализированных версий программ для каждой разновидности программно–аппаратной платформы параллельного вычислителя. Со временем появились универсальные библиотеки параллельного программирования, такие как MPI для кластеров и систем с распределённой памятью, а также OpenMP для систем с общей памятью. Однако при написании параллельной программы «вручную» для обеспечения эффективности работы полученной реализации и достижения сбалансированности нагрузки на процессоры приходится учитывать массу деталей, связанных с параметрами конкретной целевой многоядерной архитектуры, что существенно ограничивает переносимость программ. Таким образом, задача эффективного автоматического переноса последовательных программ и организации параллельных вычислений для систем на основе многоядерных процессоров является актуальной и востребованной.

Преобразования распараллеливания обычно осуществляются в две стадии. На первой – *стадии анализа*, в процессе исследования исходной программы выявляется параллелизм алгоритма или задачи. Результат фиксируется в машинно-независимом виде. На второй – *стадии синтеза*, генерируется параллельная программа, эквивалентная исходной, в соответствии с особенностями архитектуры целевой параллельной системы.

На стадии анализа производится выявление скрытого параллелизма в исходной последовательной программе. Для этого используются методы выявления зависимостей между операционными объектами программы (*зависимостей по управлению*) и зависимостей между информационными объектами программы (*зависимостей по данным*).

Выявление потенциального параллелизма последовательной программы основывается на анализе зависимостей составляющих её частей друг от друга [3,4]. В качестве подобных частей, в зависимости от выбранного масштаба рассмотрения – мелкозернистого или крупнозернистого, могут рассматриваться отдельные операторы (инструкции), группы операторов присваивания, блоки, итерации цикла, условные операторы, выполнение процедур после вызова и т.д. Взятая за основу модель крупнозернистого параллелизма, позволяет выбрать в качестве единицы планирования группы операторов, объединённые в регионы [5]. Рассмотрим различные типы отношений, возможные между регионами $g_i, g_j \in G$ с позиции их потенциального параллельного исполнения.

1. *Одновременность* $g_i\ \delta_{par}\ g_j$. Регионы $g_i$, $g_j$ могут выполняться одновременно и обращаться к используемым ячейкам памяти в произвольном порядке.

2. *Упорядоченность* $g_i\ \delta_{ord}\ g_j$. Регион $g_i$ должен выбрать все, что ему требуется, прежде чем регион $g_j$ запишет свои результаты.

3. *Консервативность* $g_i\ \delta_{con}\ g_j$. Регион $g_i$ должен записать свои результаты раньше, чем $g_j$.

4. *Последовательность* $g_i\ \delta_{seq}\ g_j$. Регион $g_i$ должен быть завершен до начала $g_j$.

Выявление отношений между регионами позволит выразить потенциальный параллелизм последовательной программы таким образом, чтобы это не повлияло на корректность полученного при её параллельном исполнении результата. Большинство методов анализа зависимостей основаны на графовом представлении программы [3–8]. Они выполняют построение *графов зависимостей по данным* и *по управлению* и делятся в свою очередь на две большие группы: статические и динамические.

*Статические* методы выполняются на этапе трансляции исходного текста программы [4, 9–11]. Возможности статических методов являются ограниченными, так как не всегда возможно выявить полностью все информационные зависимости между операторами, в связи с тем, что при анализе текста программы никогда не известны значения переменных, используемых в ней. Кроме того, в современных языках программирования существует широкий набор средств, позволяющих осуществлять неявный (косвенный) доступ к информационным объектам. Примером может служить использование указателей и их разыменования, организация доступа к элементам массивов по индексу, использование формальных параметров, процедурных переменных, виртуальных методов классов. Всё это существенно затрудняет задачу анализа потока данных и снижает эффективность выявления параллелизма статическими методами.

*Динамические* методы исследуют программу на этапе её выполнения [5]. Динамический анализ программ основан на внедрении в исходную программу дополнительных операторов, проводящих анализ. Полученная программа выполняется на некотором тестовом наборе входных данных (или нескольких наборах), и во время выполнения собирается информация о фактических зависимостях, присутствующих в программе на данном конкретном наборе данных. Такой подход позволяет производить выявление зависимостей во многих ситуациях, когда статический анализ невозможен. Поскольку анализ происходит во время выполнения программы, анализатору доступны значения всех переменных, присутствующих в программе. Поэтому появляется возможность проанализировать любые сложные и запутанные виды зависимостей и снять большинство ограничений на структуру управляющих и информационных связей последовательной. При этом под *динамическим распараллеливанием* понимается способ выявления параллельных регионов и планирование их выполнения непосредственно во время исполнения (run-time) программы.

### 3. Метод спекулятивной многопоточности

Наиболее подходящим методом динамического распараллеливания последовательных программ для многоядерных вычислительных систем с общей памятью, с учётом

обозначенных выше недостатков, является метод ***спекулятивной многопоточности*** [5, 12].

Суть метода состоит в следующем. Среди множества всех регионов $V$ последовательной программы выявляются регионы, имеющие зависимости, характер которых не может быть установлен на этапе трансляции программы из-за неоднозначности. Метод спекулятивной многопоточности предписывает параллельное многопоточное выполнение подобных регионов, в расчёте на то, что информационные зависимости на стадии исполнения не проявятся, прибегая к позиции крайнего оптимизма (спекулируя на удаче). В случае если подобные надежды оправдают себя, будет получен выигрыш в производительности, в противном случае, результаты вычислений региона должны быть аннулированы, и он будет выполнен повторно, что приведет к накладным расходам. Регионы программы, к которым применяется подобный метод, будем называть ***спекулятивными*** регионами.

Для того, чтобы организовать многопоточное выполнение спекулятивных регионов, формируется динамическая последовательность стадий их исполнения, называемых ***эпохами*** $\forall e_i \in EP \mid i = \overline{1, N},\ N = \mid EP \mid$. Например, для циклического региона эпохами являются отдельные итерации цикла. Каждая эпоха снабжается локальным буфером памяти для сохранения критических к возможным нарушениям зависимости данных.

Подобный подход позволяет фактически производить обработку отдельных итераций цикла в параллельном конвейерном режиме, заполняя ступени конвейера (ядра процессора) новыми эпохами по мере их освобождения. Такой подход минимизирует возможные потери, даже в случае повторного выполнения неудачных эпох.

### 4. Функциональный подход к параллельному программированию

Языки параллельного программирования, использующие явное управление вычислительным процессом, позволяют описать максимальный параллелизм задачи, но обеспечивают это не самым удобным способом. Это объясняется следующими причинами:

– программист сам должен формировать все параллельные фрагменты и следить за корректной синхронизацией данных;

– использование в языках такого типа «ручного» управления памятью может привести к конфликтам между процессами в борьбе за общий ресурс (программисту приходится тщательно следить за распределением памяти или явно соблюдать принцип единственного присваивания);

– при явном распараллеливании в ходе программирования не всегда адекватно можно отразить в создаваемой программе параллелизм данных, присущий задаче, что, в свою очередь может привести к дальнейшим искажениям при переносе написанной программы на другую архитектуру.

При распараллеливании последовательных программ очень редко обеспечивается достижение приемлемого уровня параллелизма из-за ограничений вычислительного метода, выбранного программистом, что часто обуславливается стереотипами последовательного мышления. В реальной ситуации учесть особенности различных параллельных систем оказалось гораздо труднее, чем это изначально предполагалось.

Создание прикладных параллельных программ, ориентированных на обработку информационных потоков, удобнее осуществлять с применением функциональных языков параллельного программирования, в которых выполнение каждого оператора осуществляется по готовности его данных. Они не требуют явного описания параллелизма задачи, который в этом случае определяется в соответствии с информационными связями. Использование такого языка позволяет:

– создавать мобильные программы с параллелизмом на уровне операторов, ограниченным лишь методом решения задачи;

– обеспечить перенос программы на конкретную архитектуру без ее изменения независимо от числа доступных ядер;

– проводить оптимизацию программы по множеству параметров с учетом специфики архитектуры ВС, для которой осуществляется трансляция без учета управляющих связей программы.

В качестве инструмента для решения возникающих проблем при разработке многопоточных приложений предлагается использовать функциональный язык параллельного программирования Clojure. Этот язык представляет собой современный диалект Лиспа и одновременно является языком общего назначения, который поддерживает разработку в интерактивном режиме и упрощает параллельное программирование. В частности, Clojure использует API JVM для работы с многопоточными приложениями, но в отличии от Java он реализует его более эффективно за счет функциональной парадигмы. При этом Clojure обеспечивает неизменность (immutable) данных, что исключает проблемы блокировки ресурсов.

Данный язык обладает следующими свойствами [13]:

– обеспечивает параллелизм на уровне данных;

– имеет архитектурную независимость (за счёт использования JVM);

– поддерживает параллельную модель акторов;

– обеспечивает поддержку функций в виде объектов первого класса;

– имеет конструкцию future, которая обеспечивает выполнение долго работающего кода одновременно с решением других задач;

– поддерживает три механизма синхронного и один асинхронного обновления данных.

Примером реализации функционального подхода может служить разработка алгоритма и программного приложения для решения задачи сортировки методом параллельного слияния [14], который демонстрирует идею декомпозиции данных в рамках известного подхода «разделяй и властвуй» [15]. Раз-

работанный алгоритм включает два этапа: деление списка на подсписки и параллельное слияние полученных подсписков.

Первый этап не сильно отличается от классического варианта, где деление списка на подсписки выполняется рекурсивно. В предлагаемой параллельной реализации первого этапа при рекурсивном спуске каждая итерация слияния выделяется в отдельную нить и ожидает выполнения. Как только в подсписках останутся единственные элементы, начинается их слияние, т.е. второй этап алгоритма.

На втором этапе происходит возврат из рекурсии, и каждая пара подсписков сливается в отдельном потоке (thread). Тем самым загружаются все свободные ядра процессора и минимизируется время их простоя.

В программе на языке Clojure сортируемые данные представляются в виде связанного списка. Для его обработки используется функция map, которая представляет собой функцию высшего порядка. Она применяет некоторую функцию к каждому элементу списка и возвращает список результатов. В функциональных языках функция map часто называется «применить ко всем», что достаточно точно определяет ее сущность.

Практическая реализация алгоритмов параллельного и последовательного слияния подтверждает рассмотренные преимущества функционального подхода. Экспериментальные данные для процессора с 4 ядрами показывают, что для размера списка от $10^3$ до $10^4$ среднее время многопоточной сортировки резко уменьшается (примерно в 6 раз), а для размеров списка $10^5$, $10^6$ и более ускорение составляет от 20 до 25%.

**5. Заключение**

В статье рассмотрены перспективные подходы и технологии адаптации последовательных программ и алгоритмов с целью эффективного использования аппаратных средств современных многоядерных процессоров при разработке многопоточных приложений.

Представлены возможности использования метода спекулятивной многопоточности для автоматического распараллеливания последовательных программ, разработанных на императивных языках программирования. Данный метод динамического распараллеливания позволяет существенно упростить анализ зависимостей по данным и управлению между последовательными регионами таких программ, повысить за счёт этого возможности их параллельного исполнения.

Функциональная парадигма создает благоприятную среду для работы программиста и упрощает разработку параллельных приложений за счет использования новых свойств языков программирования (на примере языка Clojure), которые имеют такие преимущества перед императивной реализацией, как единый поток управления программой и неявная синхронизация обрабатываемых данных.

**Литература**

1. Корячко В.П., Скворцов С.В., Телков И.А. Архитектуры многопроцессорных систем и параллельные вычисления. М.: Высшая школа, 1999. 235 с.

2. Корячко В.П., Скворцов С.В., Таганов А.И., Шибанов А.П. Эволюция автоматизированного проектирования электронно-вычислительных средств // Радиотехника. 2012. № 3. С. 97–103.

3. Корячко В.П., Скворцов С.В., Телков И.А. Модель планирования параллельных процессов в суперскалярных процессорах // Информационные технологии. 1997. № 1. С. 8–12.

4. Скворцов С.В. Оптимизация кода для суперскалярных процессоров с использованием дизъюнктивных графов // Программирование. 1996. № 2. С. 41–52.

5. Бакулев А.В. Модели и алгоритмы организации мобильных параллельных вычислений в среде многоядерных процессоров. Диссертация на соискание ученой степени кандидата технических наук. Рязань: РГРТУ, 2010. 177 с.

6. Рудаков В.Е., Скворцов С.В. Построение базового множества независимых путей потокового графа для тестирования программных модулей // Системы управления и информационные технологии. 2012. Т. 50. № 4. С. 67–70.

7. Корячко В.П., Гостин А.М., Бакулев А.В., Бакулева М.А. Дискретная математика: учебное пособие. Рязань: РГРТУ, 2011. 178 с.

8. Бакулев А.В., Бакулева М.А. Построение ассоциативных правил на основе дифференцирования графовой модели анализируемой выборки // Вестник Рязанского государственного радиотехнического университета. 2013. № 46–2. С. 86–89.

9. Бакулев А.В. Алгоритм синтеза параллельной реализации последовательной программы для вычислительных систем, построенных на базе многоядерных процессоров // Вестник Рязанского государственного радиотехнического университета. 2009. № 30. С. 43–49.

10. Скворцов С.В. Целочисленные модели оптимизации кода по критерию времени // Информационные технологии. 1997. № 10. С. 2–7.

11. Першин А.С., Скворцов С.В. Распределение регистровой памяти в системах параллельной обработки данных // Системы управления и информационные технологии. 2007. № 1 (27). С. 65–70.

12. Bakulev A.B., Bakuleva M.A., Avilkina S.B. Mathematical methods and algorithms of mobile parallel computing on the base of multi-core processors // European researcher. 2012. V. 33. № 11–1. P. 1826–1834.

13. Fogus M., Houser C. The Joy of Clojure: Thinking the Clojure Way. – Manning Publications, 2011. 300 P.

14. Козлов М.А., Скворцов С.В. Алгоритмы параллельной сортировки данных и их реализация на языке Clojure // Вестник Рязанского государственного радиотехнического университета. 2013. № 4–1 (46). С. 92–96.

15. Миллер Р. Последовательные и параллельные алгоритмы: Общий подход. М.: БИНОМ. Лаборатория знаний, 2006. 406 с.

**References**

1. Koryachko V.P., Skvortsov S.V., Telkov I.A. Architecture multi-proces-

sor systems and parallel computing. M.: Vysshaya shkola, 1999. 235 s.

2. Koryachko V.P., Skvortsov S.V., Taganov A.I., Shibanov A.P. The evolution of computer-aided design of electronic computing equipment // Radiotehnika. 2012. № 3. S. 97–103.

3. Koryachko V.P., Skvortsov S.V., Telkov I.A. Planning model of parallel processes in superscalar processors // Informacionnye tehnologii. 1997. № 1. S. 8–12.

4. Skvortsov S.V. Code optimization for superscalar processors using moat disjunctive graph // Programmirovanie. 1996. № 2. S. 41–52.

5. Bakoulev A.V. Models and algorithms for organizing mobile parallel computing environment for multi-core processors. Dissertation for the degree of candidate of technical sciences. Ryazan RSREU, 2010. 177 s.

6. Rudakov V.E., Skvortsov S.V. Building a basic set of independent paths flow graph for testing software modules // Sistemy upravleniya i informacionnye tehnologii. 2012. T. 50. № 4. S. 67–70.

7. Koryachko V.P., Gostin A.M., Bakoulev A.V., Bakuleva M.A. Discrete Mathematics: a tutorial. Ryazan RSREU, 2011. 178 s.

8. Bakoulev A.V., Bakuleva M.A. Construction of association rules based on the graph model differentiation analyzed sample // Vestnyk Ryazanskogo gosudarstvennogo radiotehnicheskogo universiteta. 2013. № 46–2. S. 86–89.

9. Bakoulev A.V. Synthesis algorithm for parallel implementation of a sequence of programs for computing systems based on multi-core processors // Vestnyk Ryazanskogo gosudarstvennogo radiotehnicheskogo universiteta. 2009. № 30. S. 43–49.

10. Skvortsov S.V. Integer Optimization Model code on the criterion of time // Informacionnye tehnologii. 1997. № 10. S. 2–7.

11. Pershin A.S., Skvortsov S.V. Distribution of registered memory systems in parallel processing // Sistemy upravleniya i informacionnye tehnologii. 2007. № 1 (27). S. 65–70.

12. Bakulev A.V., Bakuleva M.A., Avilkina S.B. Mathematical methods and algorithms of mobile parallel computing on the base of multi-core proces-sors // European researcher. 2012. V. 33. № 11–1. P. 1826–1834.

13. Fogus M., Houser C. The Joy of Clojure: Thinking the Clojure Way. – Manning Publications, 2011. 300 P.

14. Kozlov M.A., Skvortsov S.V. Parallel algorithms sorting matches the data and their implementation in Clojure // Vestnyk Ryazanskogo gosudarstvennogo radiotehnicheskogo universiteta. 2013. № 4–1 (46). S. 92–96.

15. Miller R. Serial and parallel algorithms: All approach. M. BINOM. Laboratoriya znaniĭ, 2006. 406 s.