УДК 004.272.34

Аппаратные методы синхронизации потоков в многоядерном вычислительном кластере

И.А. Медведев, Ф.М. Путря

Государственное унитарное предприятие города Москвы Научно-производственный центр "Электронные вычислительно-информационные системы", fputrya@elvees.com

***Аннотация* — В данной работе приведен обзор основных программных и аппаратных методов синхронизации потоков, применяемых в многоядерных системах. Подробно рассмотрен способ синхронизации, на основе небольшого объема общей памяти с аппаратной поддержкой примитивов синхронизации, реализованный в многоядерном кластере DSP, разработанном в ГУП НПЦ «ЭЛВИС». Предложено решение проблемы масштабируемости, характерной для данного подхода, заключающееся в применении распределенного буфера обмена. Проведен анализ влияния на производительность предложенного решения, а также рассмотрен метод повышения эффективности аппаратной синхронизации на основе очередей типа FIFO.**

***Ключевые слова* — многоядерные процессоры, синхронизация, семафоры, очереди.**

I. Введение

Согласно закону Мура количество транзисторов на кристалле удваивается каждые два года. Современные исследования прогнозируют соблюдение закона Мура в ближайшее десятилетие [1]. Возможность размещать на кристалле все большее число транзисторов на протяжении долгого времени использовалась разработчиками процессоров на цели повышения производительности одного вычислительного ядра процессора. Однако несколько лет назад произошел коренной перелом, после которого, в силу ряда технологических ограничений и проблем, основным методом повышения производительности процессора стало наращивание числа вычислительных ядер [2]. Применение многоядерных процессоров приводит к более эффективному выполнению параллельных вычислений, увеличению энергоэффективности за счет использования более простых ядер и сокращению производственного цикла процессора за счет повторного использования разработанной аппаратуры [3].

Останавливаясь на вопросе параллельных вычислений, можно выделить две проблемы: программируемость и синхронизация [4]. Под программируемостью можно понимать наложение распараллеливаемых приложений на многоядерную архитектуру, а под синхронизацией – обеспечение корректности параллельного выполнения потоков за счет исполнения зависимостей между командами. Задача распараллеливания достаточно нетривиальная и для ряда алгоритмов практически неразрешимая, однако на сегодняшний момент уже есть большой опыт создания многопоточных приложений для суперкомпьютеров и многопроцессорных вычислительных комплексов, который, с учетом специфики однокристальных систем, применим и к многоядерным процессорам. Однако потокам, на которые распараллелено некоторое приложение, необходимо взаимодействовать друг с другом, и при отсутствии механизмов синхронизации между такими потоками корректная работа приложения невозможна.

II. Методы синхронизации потоков в многоядерных системах

Проблему синхронизации можно разбить на две задачи: синхронизация данных и синхронизация процессов. Под синхронизацией данных понимается ликвидация различий между двумя копиями данных используемых разными потоками или процессами, она может решаться как программными средствами на уровне операционной системы, так и аппаратно, за счет использования механизмов обеспечения когерентности кэш памятей вычислительных ядер.

Под синхронизацией потоков понимается процесс приведения двух или нескольких потоков к такому их протеканию, когда определённые стадии разных потоков совершаются в определённом порядке, либо одновременно. Классическим способом синхронизации потоков является использование семафоров. Частным случаем семафоров являются мьютексы, схожие задачи выполняют и критические секции.

Обычно, семафор используется для того, чтобы ограничить доступ к некоторому ресурсу, заданным количеством потоков приложения, однако классическая реализация семафора никак не гарантирует порядок, в котором заблокированные потоки будут получать доступ к ресурсу. Для решения этой проблемы могут применяться более сложные семафоры, использующие программно реализованную очередь.

Все перечисленные выше методы характеризуются тем, что они реализуются программными методами. В данных методах для синхронизации, как правило, ис-

пользуется область памяти, к которой имеют доступ все вычислительные ядра, на которых исполняется многопоточное приложение. В результате каждая процедура синхронизации, приводит к вызову системных функций синхронизации и к обращению к разделяемой между потоками области памяти. Таким образом, каждая процедура синхронизации является довольно накладной с точки зрения затрачиваемого на нее времени, поскольку даже в случае обращения к разделяемой ячейке памяти, как минимум необходимо время на процедуры обращения к памяти и, в случае использования кэш памяти, на процедуру передачи строки кэш памяти с используемой для синхронизации ячейкой памяти всем ядрам, участвующим в процедуре синхронизации в целях обеспечения когерентности кэш памятей.

С увеличением числа ядер в процессоре для более эффективного использования системы необходимо большее число параллельно исполняемых потоков, и соответственно более частый вызов процедур синхронизации. В результате накладные расходы на процедуры синхронизации могут привести к тому, что эффективность исполнения приложения с ростом числа ядер не будет увеличиваться, а в ряде случаев может даже и уменьшаться.

Таким образом, аппаратная поддержка стандартных механизмов синхронизации потоков является необходимым элементом в современных многоядерных системах, и от эффективности работы механизма синхронизации, в конечном счете, зависит и производительность всего программно-аппаратного комплекса, реализующего определенную функцию на многоядерном процессоре.

На сегодняшний день существует несколько аппаратных решений, поддерживающих различные механизмы синхронизации. Один из механизмов направленных на оптимизацию процесса синхронизации через общее адресное пространство – это введение в систему инструкций ядра специализированных команд доступа к памяти, позволяющих заданному ядру заблокировать доступ к определенной ячейке памяти со стороны других ядер. Недостатки данного подхода уже упоминались выше – это потеря времени на процедуру обращения к общей памяти и невозможность контролировать последовательность разблокирования потоков. Другими аппаратными методами синхронизации является аппаратная поддержка механизма передачи сообщений за счет реализации почтовых ящиков, представляющих собой аппаратно реализованные очереди типа FIFO, как это, например, сделано в процессоре CELL [5]. Еще одним способом синхронизации является организация небольшого объема общей для всех ядер памяти, с аппаратной поддержкой примитивов синхронизации. Преимущество такого подхода – скорость, достигаемая за счет быстрого доступа к общему участку памяти, и синхронизации между потоками, выполняемой всего одной инструкцией обращения к такой памяти. Данный подход использован в многоядерных DSP-процессорах серии «МУЛЬТИКОР» фирмы ГУП НПЦ "ЭЛВИС" [6], более подробно мы его рассмотрим в следующем параграфе. Существуют также и более экзотические подходы к проблеме аппаратной поддержки синхронизации, такие как транзакционная память. Данный подход подробно описан в [7]. Там же приводится классификация механизмов синхронизации применимая и к перечисленным выше подходам. Так синхронизацию между потоками можно разделить на сильную и слабую. В случае сильной синхронизации аппаратурой гарантируется определенная последовательность выполнения команд различных потоков. При слабой синхронизации не отслеживается последовательность обращения разных потоков к разделяемому ресурсу, в этом случае программист должен быть уверен в том, что последовательность таких обращений не критична, либо дополнительно использовать программные примитивы синхронизации.

Стоит отметить, что эффективность разных механизмов синхронизации зависит и от типа распараллеливания. Так, в случае разбиения алгоритма на параллельные потоки и в случае программной конвейеризации эффект от использования разных примитивов синхронизации может быть разным.

III. Централизованный буфер обмена, проблема масштабируемости

Архитектура многоядерных DSP-процессоров серии «МУЛЬТИКОР» представляет собой гетерогенную систему на кристалле, состоящую из одного управляющего ядра, которое при необходимости, обеспечивает работу операционной системы и специализированного многоядерного DSP-кластера, выполняющего в основном вычислительные задачи. Однако, вычислительные задачи, выполняемые на многоядерном кластере могут представлять собой весьма сложные многопоточные программы, предполагающие активный обмен данными между ядрами и использование механизмов синхронизации потоков.

В кластере применяется метод аппаратной поддержки синхронизации, основанный на использовании небольшого объема доступной всем ядрам памяти. Буфер обмена, входящий в состав кластера, представляет собой многопортовый регистровый файл, каждая ячейка которого снабжена дополнительным битом состояния для сохранения информации о типе последней операции (рис. 1б). Запись и чтение в регистровый файл производится согласно управляющей модели, которая, в зависимости от режима работы буфера обмена, обеспечивает сильную или слабую

зависимость последовательности выполнения операций. При сильной зависимости запись в ячейку буфера возможна только после чтения и наоборот, чтение из ячейки возможно только после записи. Невозможная операция (например, "чтение после чтения") приводит к блокировке ядра, которое отправило запрос, до тех пор, пока действиями другого ядра данный конфликт не разрешится. При слабой зависимости ограничений на последовательность выполнения операций нет. Оба режима работы буфера позволяют реализовать синхронизацию ядер, но при этом режим слабой зависимости требует более детального анализа программы со стороны программиста.

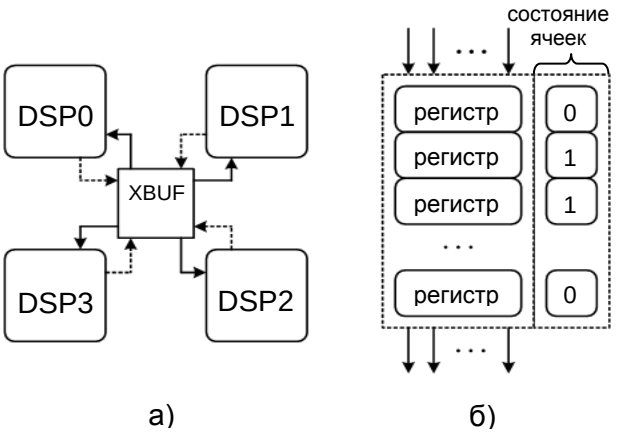

Рис. 1. Централизованный буфер обмена
а). Организация коммутации б). Буфер обмена

Буфер обмена (XBUF) топологически расположен в центре DSP-кластера, и все ядра имеют к нему равноправный доступ (рис. 1а). Однако, с ростом числа ядер в DSP-кластере, существенно усложняется коммутационная логика в централизованном буфере, усложняются пути линий связи на топологии кристалла и увеличиваются задержки на них, что ограничивает масштабируемость всей системы. Так, реализация буфера обмена, обращения к которому могут выполняться всеми ядрами без потери рабочих тактов, сильно усложняется даже для случая 4 ядер. Для решения этой проблемы предлагается разработать распределенный буфер обмена. Основными требованиями, выдвигаемыми к распределенному буферу, являются:

1). Сохранение полной функциональности механизма синхронизации.

2). Решение проблемы масштабируемости системы.

Для определенности дальнейший анализ мы будем проводить для случая кластера, содержащего 4 ядра.

IV. Распределенный буфер обмена

Для решения проблемы масштабируемости предлагается использовать распределённый буфер обмена, который разделяется на равные части (почтовые ящики Xmail), каждая из которых помещается рядом с соответствующим ядром и является для него ближней, остальные же части будут для данного ядра дальними и доступ к ним может потребовать несколько тактов. Можно выделить три основных варианта коммутации между ядрами: коммутация типа "каждый с каждым" (рис. 3а), однонаправленное кольцо (рис. 2) и двунаправленное кольцо (рис. 3б). Первые два варианта были реализованы в RTL и для них проведен анализ производительности методом моделирования RTL моделей для случаев реальных и синтезированных задач.

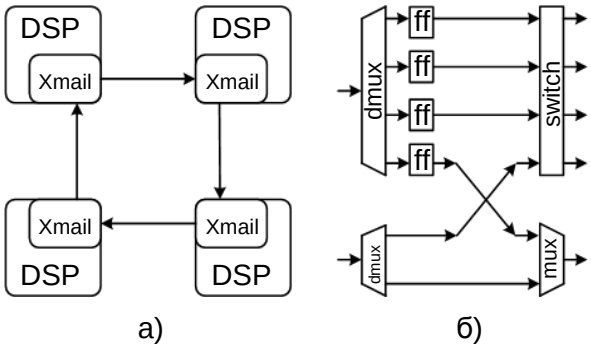

Рис. 2. Буфер обмена с коммутацией однонаправленного кольца
а). Общая архитектура б). Схема маршрутизации

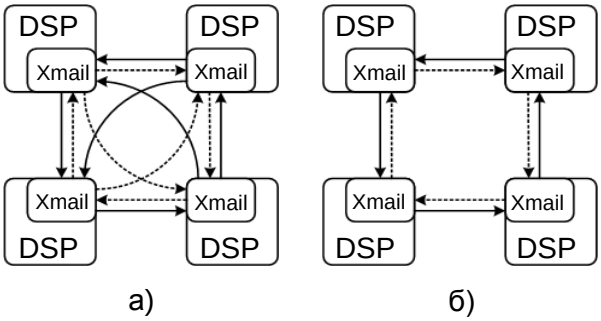

Рис. 3. Буфер обмена с коммутацией
а). «Каждый с каждым» б). Двунаправленного кольца

Реализация буфера обмена с коммутацией типа "каждый с каждым" не решает проблему масштабируемости системы из-за сложности организации межсоединений и используется в работе для сравнительного анализа производительности, т.к. из всех распределенных систем она обладает минимальными расстояниями между элементами с точки зрения времени доступа.

Для буфера обмена с кольцевой коммутацией предложены схемы маршрутизации. Для коммутации однонаправленного кольца схема маршрутизации представлена на рис. 2б, для коммутации двунаправленного кольца – на рис. 4. Для последней схемы необходимо предусмотреть протокол маршрутизации, который будет отвечать за распределение запросов по разным кольцам. Также следует заметить, что для

эффективного использования таких вариантов коммутации необходимо, чтобы при блокировке определенного запроса не блокировались все порты на пути, который он проходит. В этом случае заблокированный запрос должен сохраняться в том ядре, в котором он был заблокирован. Для этого в коммутационной логике введены специальные буферные элементы, содержащие регистры (flip-flop, на рисунках обозначены как ff).

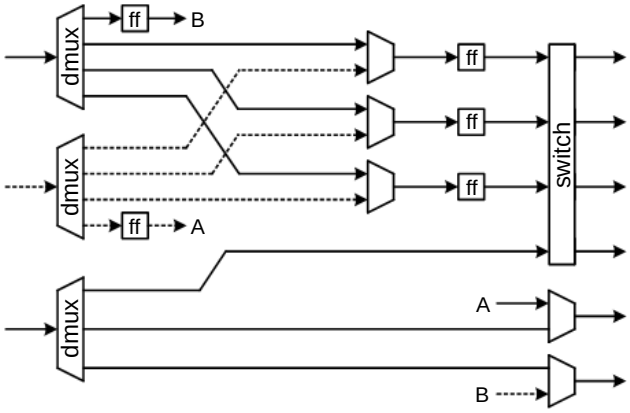

Рис. 4. Схема маршрутизации буфера обмена с коммутацией двунаправленного кольцо

V. Анализ производительности

Для анализа производительности использовались RTL-модели для следующих проектов: централизованный буфер обмена, распределенный буфер с коммутацией каждый с каждым, распределенный буфер с коммутацией типа однонаправленное кольцо. Набор тестов состоит из теста БПФ (быстрое преобразование Фурье) и тестов на передачу управления. Результаты выполнения тестов были получены путем моделирования вышеописанных RTL-моделей. Первый тест представляет собой вариант распараллеливания путем разбиения на параллельные потоки, второй – программная конвейеризация.

Исходный тест БПФ показал логичное падение производительности для распределенных вариантов буфера обмена, вызванное простоями ядер при обращениях к частям обменного буфера, расположенных в других ядрах. Поэтому код теста был оптимизирован. Оптимизация проводилась под однонаправленную кольцевую архитектуру распределенного буфера обмена таким образом, чтобы большая часть обращений к регистрам буфера обмена приходилась на регистры ближней для данного ядра части буфера и, таким образом, минимизировалось время простоя ядра при обращениях к дальним регистрам буфера. Результаты представлены для двух вариантов теста: неоптимизированного и оптимизированного. Как видно из рис. 5, время выполнения теста БПФ при кольцевой архитектуре буфера обмена существенно сокращается после оптимизации теста.

В распределенном варианте реализации буфера обмена доступ к "дальним" регистрам осуществляется дольше, чем к "ближним". Поэтому для буфера с распределенной архитектурой есть смысл говорить о двух типах блокировок: блокировки ядра, при передаче запроса к "дальним" регистрам и стандартные блокировки ядер для режима сильной зависимости. Причем здесь следует учитывать их взаимосвязь и влияние друг на друга. На рис. 6 и рис. 7 представлено соотношение блокировок двух типов, возникающих при синхронизации в распределенной системе для оптимизированного и неоптимизированного варианта теста БПФ.

Теперь рассмотрим тесты на передачу управления. Общий принцип данных тестов заключается в конвейеризации: когда каждое ядро, выполнив свой этап обработки данных, передает управление и данные следующему ядру. При анализе производительности будем менять число операций (T), требующихся каждому ядру на выполнение очередного этапа перед передачей управления следующему ядру.

Рассмотрим два варианта данного теста (для определенности назовем их соответственно А и Б). Алгоритм передачи управления в тесте А можно описать следующей схемой: DSP0 –> DSP1 –> DSP2 –> DSP3. В данном тесте интервалы, через которые ядра предают управление, одинаковы. Алгоритм передачи управления в тесте Б идет по схеме: DSP0 –> DSP1 –> DSP3; DSP0 –> DSP2 –> DSP3; В данном случае, если ядра DSP0 и DSP3 выполняют T операций перед передачей управления, то ядра DSP1 и DSP2 выполняют один и тот же этап, но на него необходимо в два раза больше операций - 2T. Для каждого теста конвейерный алгоритм выполняется для большого числа входных блоков данных, достаточного, для того, чтобы время простоя ядер при ожидании первого блока данных (загрузка конвейера) не влияло на результаты эксперимента.

Результаты выполнения тестов А и Б показаны на рис. 8. По вертикальной оси откладывается отношение производительности теста при определенной архитектуре буфера обмена к производительности теста при централизованной архитектуре.

Анализируя полученные результаты, можно сделать вывод, что падение производительности при распределенной архитектуре буфера обмена ощутимо сказывается на приложениях, которые достаточно часто используют механизм синхронизации.

В случае редкого использования механизма синхронизации (синхронизация выполняется не чаще чем раз в 50 тактов) и оптимизации приложений под распределенную архитектуру буфера обмена падение производительности значительно меньше.

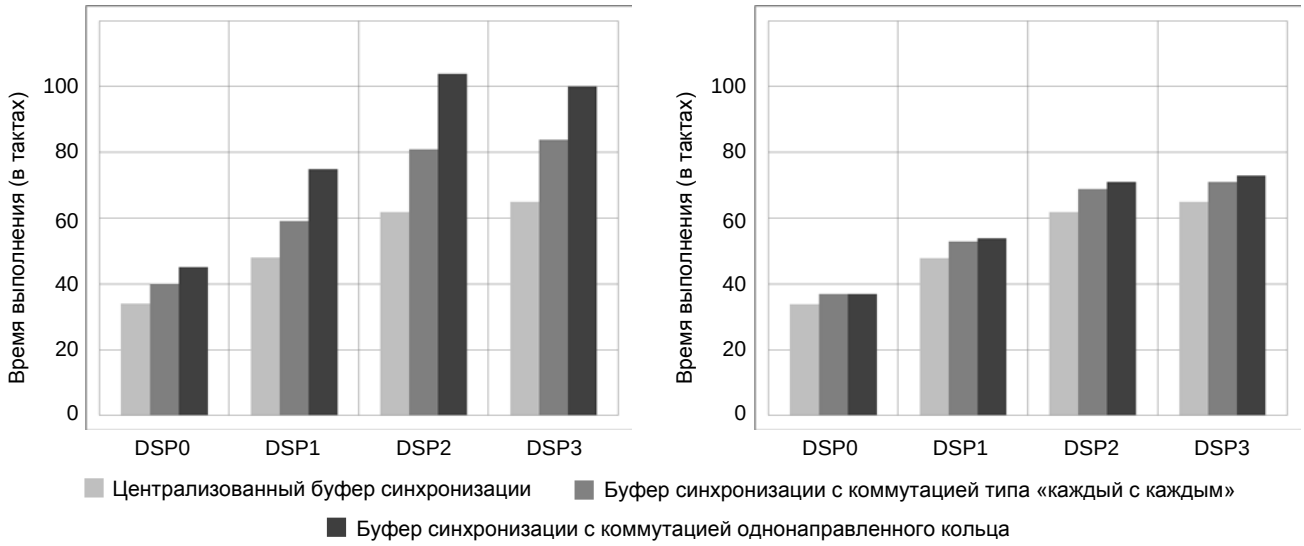

Рис. 5. Время выполнения основной части БПФ теста, слева - неоптимизированная программа, справа - оптимизированная

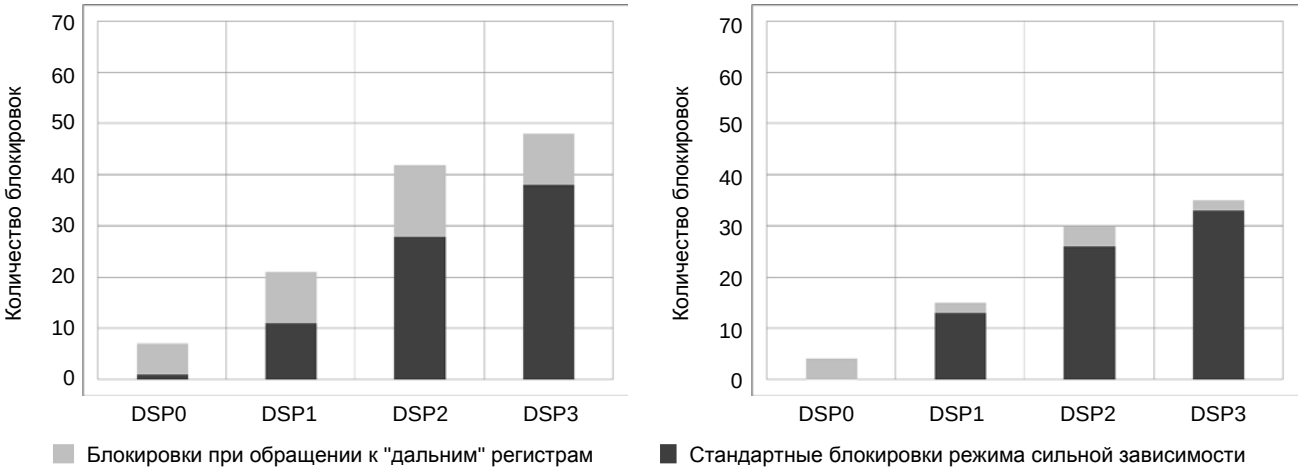

Рис. 6. Соотношение блокировок, возникающих при выполнении основной части БПФ теста при коммутации буфера обмена "каждый с каждым", слева - неоптимизированная программа, справа - оптимизированная

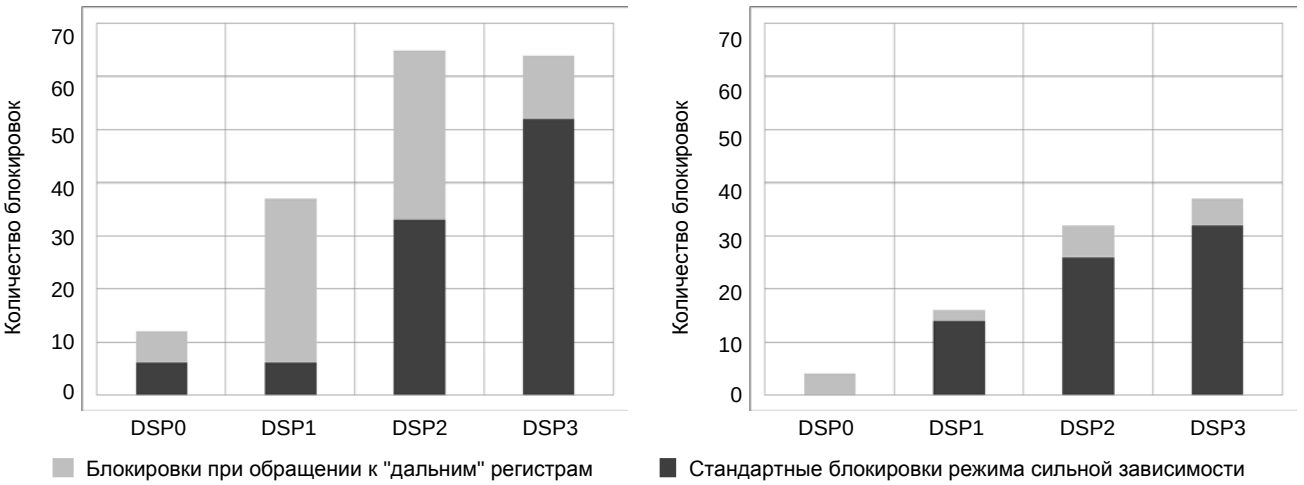

Рис. 7. Соотношение блокировок, возникающих при выполнении основной части БПФ теста при коммутации буфера обмена в виде однонаправленного кольца, слева - неоптимизированная программа, справа - оптимизированная

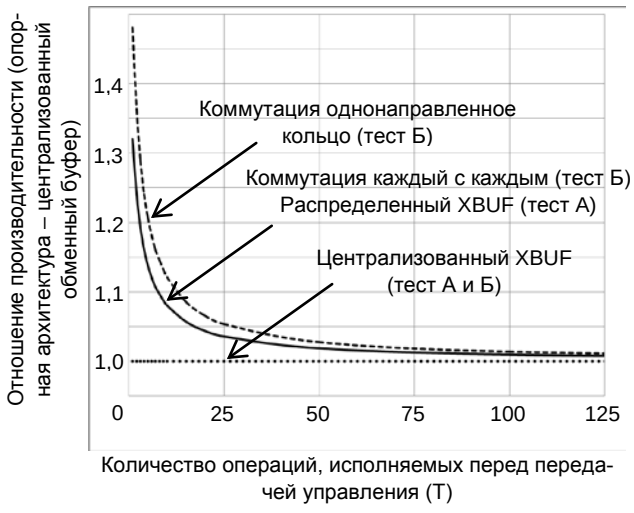

Рис. 8. Сравнительный анализ производительности тестами на передачу управления

VI. Почтовый ящик на основе очереди

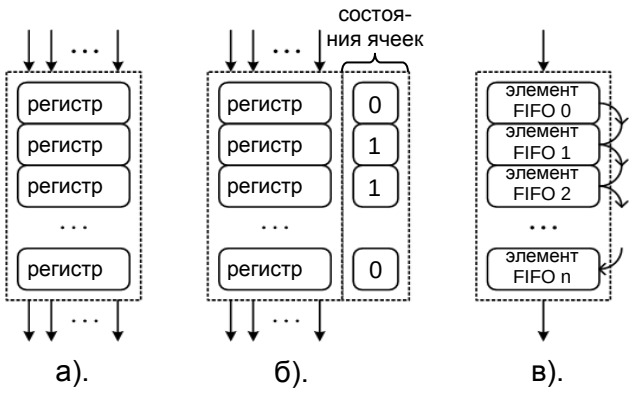

Рис. 9. Использование регистрового файла в буфере обмена
а). Режим слабой синхронизации б). Режим сильной синхронизации в). Режим очереди

Для повышения эффективности механизма синхронизации предлагается новый режим работы распределенного буфера, который реализует еще один механизм синхронизации с сильной зависимостью. Основная идея такой реализации заключается, в том, чтобы использовать каждую часть распределенного буфера, принадлежащую конкретному ядру как очередь типа FIFO (рис. 9в). В этом случае обращение будет производиться не к определенному регистру буфера обмена, а к определенному почтовому ящику, содержащему очередь запросов. При этом путем введения ограничения на чтение только из "ближнего" почтового ящика и применения механизма отложенной записи в почтовые ящики, можно добиться устранения блокировок ядер, обусловленных обращениями к дальним почтовым ящикам. В этом случае можно добиться существенного сокращения простоя ядер при синхронизации по сравнению с обычной распределенной реализацией буфера обмена, поскольку при такой организации буфера ядра будут блокироваться только при попытках чтения из пустой очереди, либо при записи в полную очередь.

Применение почтовых ящиков на основе очереди предоставляет возможность для создания более гибких и более эффективных примитивов синхронизации. На рис. 9 приведены схемы почтового ящика, для каждого из возможных режимов синхронизации.

VII. Заключение

С ростом числа вычислительных ядер реализация аппаратной поддержки синхронизации на основе централизованных решений становится все менее эффективной. Применение распределенных решений, когда обменный буфер с аппаратной поддержкой синхронизации распределяется между ядрами, дает возможность создавать быстродействующие программы, использующие стандартные примитивы синхронизации на основе механизмов передачи сообщений, для которых производительность системы будет сопоставима с идеальным централизованным вариантом. Однако, при программировании такой распределенной системы необходима оптимизация программы с учетом особенностей архитектуры. Более того, при частом использовании примитивов синхронизации, наблюдается падение эффективности, вызванное простоями ядер, при обращениях к дальним регистрам буфера обмена. Для решения данной проблемы предлагается дополнить буфер обмена режимом почтовых ящиков на основе очередей. Такие очереди дополнительно обеспечивают аппаратную поддержку каналов передачи сообщений, что позволит ускорить приложения, использующие механизм передачи сообщений.

Литература

1. International Technology Roadmap for Semiconductors. URL: http://public.itrs.net/reports.html (дата обращения: 21.01.10).
2. Путря Ф. М. Архитектурные особенности процессоров с большим числом вычислительных ядер // Информационные технологии. 2009. № 4. С. 2–7.
3. Held J., Bautista J., Koehl S. From a few cores to many: A tera-scale computing research overview // Intel White Paper. 2006.
4. Shaoshan L., Gaudiot J.-L. Synchronization Mechanisms on Modern Multi-core Architectures // Advances in Computer Systems Architecture / Lynn C., Yunheung P., Sangyeun C. – 2007.
5. Kistler M., Perrone M., Petrini F. Cell Multiprocessor Communication Network: Built for Speed // IEEE Micro 2006. № 26. P. 10–23.
6. ГУП НПЦ "ЭЛВИС". URL: http://www.multicore.ru (дата обращения 21.01.10).
7. Drepper U. Parallel Programming with Transactional Memory // ACM QUEUE. 2008. P. 40–45.