20. *Pavel Pavlukhin, Igor Menshov.* On Implementation High-Scalable CFD Solvers for Hybrid Clusters with Massively-Parallel Architectures, *Lecture Notes in Computer Science,* 2015, Vol. 9251, pp. 436-444.

Статью рекомендовал к опубликованию д.т.н. А.С. Горобцов.

Дордопуло Алексей Игоревич – Южный федеральный университет; e-mail: scorpio@mvs.tsure.ru; 347928, г. Таганрог, 10-й пер. 114/1; тел. 88634361364; НИИ МВС ЮФУ; заведующий лабораторией; к.т.н.

Левин Илья Израилевич – e-mail: levin@superevm.ru; 347928, г. Таганрог, ул. Петровская, д. 15, кв. 143; тел. 88634612111; Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров; директор; д.т.н.; профессор.

Каляев Игорь Анатольевич – e-mail: kaliaev@mvs.sfedu.ru; 347928, г. Таганрог, ул. Р. Люксембург, 8а; тел. 88634360657; НИИ МВС ЮФУ; г.н.с.; чл.-корр. РАН; д.т.н.; профессор.

Гудков Вячеслав Александрович – e-mail: gudkov@mvs.tsure.ru; 347905, г. Таганрог, ул. Дзержинского, 110; тел. 88634612111; Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров; с.н.с.; к.т.н.

Гуленок Андрей Александрович – e-mail: andrei_gulenok@mail.ru; 347905, г. Таганрог, ул. Амвросиевская, 78; НИИ МВС ЮФУ; с.н.с.; к.т.н.

Dordopulo Alexey Igorevitch – Southern Federal University; e-mail: scorpio@mvs.tsure.ru; 114/1, 10th lane, Taganrog, 347928, Russia; phone +78634361364; SRI MCS SFU; head of laboratory; cand. of eng. sc.

Levin Ilya Izrailevitch – e-mail: levin@superevm.ru; 15, Petrovskaya street, ap. 143, Taganrog, 347928, Russia; phone +78634612111; Scientific Research Centre of Supercomputers and Neurocomputers; director; dr. of eng. sc.; professor.

Kalyaev Igor Anatolievitch – e-mail: kaliaev@mvs.sfedu.ru; 8a, Rosa Luxembourg street, Taganrog, 347928, Russia; phone +78634360657; SRI MCS SFU; chief research associate; correspondent member of the RAS; dr. of eng. sc.; professor.

Gudkov Vyacheslav Alexandrovitch – e-mail: gudkov@mvs.tsure.ru; 110, Dzerzhinskogo street, Taganrog, 347905, Russia; phone +78634612111; Scientific Research Centre of Supercomputers and Neurocomputers; senior staff scientist; cand. of eng. sc.

Gulenok Andrei Alexandrovitch – e-mail: andrei_gulenok@mail.ru; 78, Amvrosievskaya street, Taganrog, 347905, Russia; SRI MCS SFU; senior staff scientist; cand. of eng. sc.

УДК 004.272.26 DOI 10.18522/2311-3103-2016-11-5464

И.И. Кулагин, М.Г. Курносов

О СПЕКУЛЯТИВНОМ ВЫПОЛНЕНИИ КРИТИЧЕСКИХ СЕКЦИЙ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С ОБЩЕЙ ПАМЯТЬЮ[*]

Целью работы является исследование метода выполнения критических секций при помощи программной транзакционной памяти, а также описание основных аспектов её реализации. Транзакционная память – это активно развивающийся примитив синхронизации, позволяющий создавать масштабируемые потокобезопасные параллельные программы для вычислительных систем с общей памятью. Для исследования была выбрана реализация транзакционной памяти в компиляторе GCC (библиотека libitm). Библиотека libitm

[*] Работа выполнена при финансовой поддержке РФФИ (гранты № 16-07-00712, 15-07-02693).

реализует предложенную компанией Intel спецификацию Intel TM ABI. В ходе исследования были выявлены серьёзные проблемы в транзакционной памяти во время обнаружения конфликтов, приводящие к деградации производительности параллельных программ. Одна из таких проблем в статье обозначена как возникновение ложных конфликтов (конфликты происходящие на уровне runtime-библиотеки, но отсутствующие на самом деле). В работе предложен подход к сокращению ложных конфликтов, возникающих при выполнении параллельных программ на базе транзакционной памяти. Суть подхода заключается в варьировании параметров реализации транзакционной памяти в runtime-библиотеке компилятора GCC по результатам предварительного профилирования программы (profile-guided optimization). Для осуществления профилирования реализован модуль компилятора GCC, выполняющий инструментацию параллельных программ, использующих транзакционную память (STM-программы). Разработанный модуль анализирует промежуточное представление транзакционных секций и встраивает в них вызовы функций профилировщика для регистрации основных событий (транзакционное чтение/запись, начало транзакции, конец транзакции, отмена транзакции). Профилировщик STM-программ также реализован. Эффективность метода сокращения ложных конфликтов исследована на тестовых программах из пакета STAMP. Данный пакет содержит в себе 8 тестов, оперирующих хеш-таблицами, списками и массивами, и является общепринятым для тестирования эффективности транзакционной памяти. Выполнение всех критических секций в тестах организовано при помощи программной транзакционной памяти. Используя предложенный метод, время выполнения тестов удалось сократить приблизительно на 20 %. По результатам профилирования получены субоптимальные значения параметров реализации runtime-библиотеки программной транзакционной памяти.

Программная транзакционная память; компиляторы; параллельное программирование.

I.I. Kulagin, M.G. Kurnosov

TOWARDS CRITICAL SECTIONS SPECULATED EXECUTION OF PARALLEL PROGRAMS ON MULTI-CORE SYSTEMS

Software transactional memory (STM) is an approach to developing the thread-safe programs. In contrast to locking a block of code by mutex, the main idea of this approach is to protect memory areas from concurrent access by threads. In this paper, we investigate efficiency of software transactional memory implementation in GCC compiler. The GCC implementation of software transactional memory implements specification that proposed by Intel company – Intel TM ABI. Software tools for instrumentation and profiling STM-programs are proposed. Profile-guided method for reducing false conflicts in STM-programs is presented. False conflict is a conflict that exists on the level of runtime library but not when the memory accessing occurs. The instrumentation module analyzes the code of transactional sections and puts calls for registration of some events (begin transactions, transactional read/write, commit transactions and abort transactions). The profiling of the instrumented program allows obtaining the dynamic properties of execution transactional code like size of used data, read/write addresses, timestamp of events, etc. The static instrumentation allows optimizing the dynamic properties of execution transactional sections. The method for reducing false conflicts performs the tuning of transactional memory parameters value in GCC implementation (libitm runtime-library) by using the profiling results (profile-guided optimization) like the mean memory size of transactional read/write operations. The efficiency of reducing the false conflicts is investigated on the STAMP benchmarks. These benchmarks contain eight tests which operate with hash tables, lists, arrays protected by software transactional memory. Using the proposed method of tests the time is reduced approximately to 20 %. Suboptimal parameter values of STM runtime-library have been obtained by the proposed profile-guided method.

Software transactional memory; instrumentation; profile-guided optimization; multithreaded programming; compilers.

Введение. При разработке параллельных программ для вычислительных систем с общей памятью неизбежно возникает необходимость защиты разделяемых ресурсов от возникновения состояния гонок за данными (data race). Для синхронизации доступа к разделяемым областям памяти активно используются примитивы

синхронизации, основанные на механизме блокировок (семафоры, мьютексы и др.). Использование данных методов подразумевает создание в коде программы критических секций, выполнение которых возможно только одним потоком в каждый момент времени [1].

Практика показывает, что при одновременном выполнении одной критической секции потоки могут обращаться к непересекающимся областям памяти [2, 3]. В этом случае возникновение состояния гонки за данными не возникает, следовательно, блокировать выполнение потоков не нужно, так как использование блокировок увеличит накладные расходы на выполнение критической секции. На рис. 1 изображен пример данной ситуации. В критической секции содержится код добавления элемента в хеш-таблицу. При добавлении множеством потоков элементов с различными ключами *key* в хеш-таблицу *h* с большой долей вероятности хеш-коды *i* элементов будут отличаться и каждый поток будет выполнять добавление нового узла в отдельный список *h*[*i*]. Блокировка всей функции здесь является избыточной и неэффективной.

```
function hashtable_add(h, key, value)
    lock_acquire()
    i = hash(key)
    list_add_front(h[i], key, value)
    lock_release()
end function
```

Рис. 1. Добавление пары (key, value) в хеш-таблицу h

Таким образом, необходимы эффективные методы выполнения критических секций, позволяющие избежать избыточное блокирование выполнения потоков, которые не требуют глубокой переработки параллельной программы. Кроме того, данные методы должны обеспечить корректность программы – отсутствие состояний гонок за данными, взаимных блокировок (deadlock), ситуаций активных блокировок (livelock) [4–6] и др.

Программная транзакционная память. Программная транзакционная память (software transactional memory) – это подход к созданию потокобезопасных программ, основная идея которого заключается в защите области памяти от конкурентного доступа, а не участка кода, как в случае использования блокировок [7, 8]. В рамках программной транзакционной памяти программисту предоставляются языковые конструкции [20] или API для формирования в программе транзакционных секций (transactional section) – участков кода, в которых осуществляется защита совместно используемых областей памяти. Выполнение потоками таких секций осуществляется без их блокирования. На среду выполнения (runtime) ложатся задачи по контролю за корректностью выполнения транзакций. Если во время выполнения транзакции другие потоки одновременно с ней не модифицировали защищенную область памяти, то транзакция считается корректной, и она фиксируется (commit). Если же два или более потока при выполнении транзакций обращаются к одной и той же области памяти и как минимум один из них выполняет операцию записи, то возникает конфликт (аналог состояния гонки данных). Для его разрешения выполнение одной или нескольких транзакций может быть либо приостановлено (до завершения конфликтующей транзакции), либо прервано, а все модифицированные ими (их потоками) области памяти приведены в исходное состояние (на момент старта транзакции) – отмена транзакции и восстановление (cancel and rollback).

На рис. 2 представлен пример создания транзакционной секции, в теле которой выполняется добавление элемента в хэш-таблицу множеством потоков. После выполнения тела транзакционной секции каждый поток приступит к выполнению кода, следующего за ней, в случае отсутствия конфликтов. В противном случае поток повторно будет выполнять транзакцию до тех пор, пока его транзакция не будет успешно зафиксирована.

```
/* Совместно используемая хеш-таблица */
hashtable_t *h;

/* Код потоков */
void *thread_start(void *arg) {
    struct data *d = (struct data *)arg;
    prepareData(d);

    /* Транзакционная секция */
    __transaction_atomic {
        /* Добавление элемента в хеш-таблицу */
        struct data *d = (struct data *)arg;
        hashtable_insert(h, d);
    }

    saveData(d);
    return NULL;
}
```

Рис. 2. Добавление пары (key, value) в хеш-таблицу h

Основными архитектурными решениями, определяющими поведение программной транзакционной памяти, являются политика обновления объектов в памяти, а также стратегия обнаружения конфликтов.

Политика обновления объектов в памяти определяет, когда изменения объектов в рамках транзакции будут записаны в память. Распространение получили две основные политики – ленивая и ранняя. Ленивая политика обновления объектов в памяти (lazy version management) откладывает все операции с объектами до момента фиксации транзакции. Все операции записываются в специальном журнале (redo log), который при фиксации используется для отложенного выполнения операций. Очевидно, что это замедляет операцию фиксации, но существенно упрощает процедуры ее отмены и восстановления. Примером реализаций ТП, использующих данную политику, являются RSTM-LLT [10] и RSTM-RingSW [12, 13].

Ранняя политика обновления (eager version management) предполагает, что все изменения объектов сразу записываются в память. В журнале отката (undo log) фиксируются все выполненные операции с памятью. Он используется для восстановления оригинального состояния модифицируемых участков памяти в случае возникновения конфликта. Эта политика характеризуется быстрым выполнением операции фиксации транзакции, но медленным выполнением процедуры ее отмены. Примерами реализаций, использующих раннюю политику обновления данных, являются GCC (libitm), TinySTM [8], LSA-STM [9], Log-TM [13], RSTM [12] и др.

Момент времени, когда инициируется алгоритм обнаружения конфликта, определяется стратегией обнаружения конфликтов. При отложенной стратегии (lazy conflict detection) алгоритм обнаружения конфликтов запускается на этапе фиксации транзакции [14]. Недостатком этой стратегии является то, что временной интервал между возникновением конфликта и его обнаружением может быть достаточно большим. Эта стратегия используется в RSTM-LLT [12] и RSTM-RingSW [14, 15].

Пессимистичная стратегия обнаружения конфликтов (eager conflict detection) запускает алгоритм их обнаружения при каждой операции обращения к памяти. Такой подход позволяет избежать недостатков отложенной стратегии, но может привести к значительным накладным расходам, а также, в некоторых случаях, может привести к увеличению числа откатов транзакций. Стратегия реализована в TinySTM [8], LSA-STM [9] и TL2 [16]. В компиляторе GCC (libitm) реализован комбинированный подход к обнаружению конфликтов – отложенная стратегия используется совместно с пессимистической.

Для обнаружения конфликтных операций требуется отслеживать изменения состояния используемых областей памяти. Информация о состоянии может соответствовать областям памяти различной степени гранулярности. Выбор гранулярности обнаружения конфликтов – один из ключевых моментов при реализации программной транзакционной памяти.

На сегодняшний день используются два уровня гранулярности: уровень программных объектов (object-based STM) и уровень слов памяти (word-based STM). Уровень программных объектов подразумевает отображение объектов модели памяти языка (объекты C++, Java, Scala) на метаданные runtime-библиотеки. При использовании уровня слов памяти осуществляется отображение блоков линейного адресного пространства процесса на метаданные. Метаданные хранятся в таблице, каждая строка которой соответствует объекту программы или области линейного адресного пространства процесса. В строке содержатся номер транзакции, выполняющей операцию чтения/записи памяти; номер версии отображаемых данных; их состояние и др. Модификация метаданных выполняется runtime-системой с помощью атомарных операций процессора.

В данной работе рассматривается реализация программной транзакционной памяти в компиляторе GCC, использующий уровень слов памяти (в версиях GCC 4.8+ размер блока – 16 байт).

На рис. 3 представлен пример организации метаданных транзакционной памяти с использованием уровня слов памяти (GCC 4.8+). Линейное адресное пространство процесса фиксированными блоками циклически отображается на строки таблицы, подобно кэшу прямого отображения. Выполнение операции записи приведет к изменению поля «состояние» соответствующей строки таблицы на «заблокировано». Доступ к области линейного адресного пространства, у которой соответствующая строка таблицы помечена как «заблокировано», приводит к конфликту.

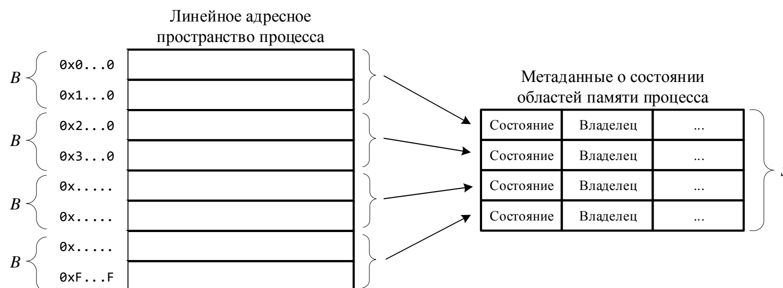

Рис. 3. Таблица с метаданными транзакционной памяти GCC 4.8+ (word-based STM): $B = 16$, $S = 2^{19}$

Основными параметрами транзакционной памяти с использованием уровня слов памяти являются число S строк таблицы и количество B адресов линейного адресного пространства, отображаемых на одну строку таблицы. От выбора этих

параметров зависит число ложных конфликтов – ситуаций аналогичных ситуации ложного разделения данных при работе кэша процессора. В текущей реализации GCC (4.8-6.2) эти параметры фиксированы.

Ложные конфликты. При отображении блоков линейного адресного пространства процесса на метаданные runtime-библиотеки возникают коллизии. Это неизбежно, так как размер таблицы метаданных гораздо меньше размера линейного адресного пространства процесса. Коллизии приводят к возникновению ложных конфликтов. *Ложный конфликт* – ситуация, при которой два или более потока во время выполнения транзакции обращаются к разным участкам линейного адресного пространства, но сопровождаемые одними и теме же метаданными о состоянии, и как минимум один поток выполняет операцию записи. Таким образом, ложный конфликт – это конфликт, который происходит не на уровне данных программы, а на уровне метаданных runtime-библиотеки.

Возникновение ложных конфликтов приводит к откату транзакций, так же, как и возникновение обычных конфликтов, несмотря на то, что состояние гонки за данными не возникает, что влечет за собой увеличение времени выполнения STM-программ. Сократив число ложных конфликтов можно существенно уменьшить время выполнения программы.

На рис. 4 показан пример возникновения ложного конфликта в результате коллизии отображения линейного адресного пространства на строку таблицы. Поток 1 при выполнение операции записи над областью памяти с адресом A1 захватывает соответствующую строку таблицы. Выполнение операции чтения над областью памяти с адресом A2 потоком 2 приводит к возникновению конфликта, несмотря на то что операции чтения и записи выполняются над различными адресами. Последнее обусловлено тем, что 1 и 2 отображены на одну строку таблицы метаданных.

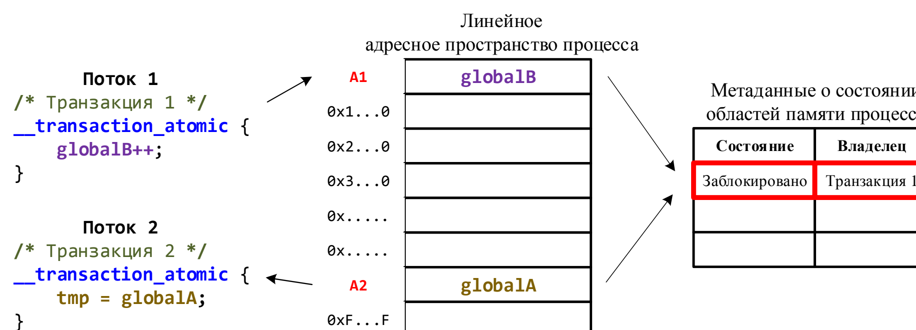

Рис. 4. Пример возникновения ложного конфликта при выполнении двух транзакций (GCC 4.8+)

Сокращение числа ложных конфликтов. В работе [18] для минимизации числа ложных конфликтов предлагается использовать вместо таблицы с прямой адресацией (как в GCC 4.8+), в которой индексом является часть линейного адреса, хеш-таблицу, коллизии в которой разрешаются методом цепочек. В случае отображения нескольких адресов на одну запись таблицы каждый адрес добавляется в список и помечается тэгом для идентификации (рис. 5). Такой подход позволяет избежать ложных конфликтов, однако накладные расходы на синхронизацию доступа к метаданным существенно возрастают, так как значительно увеличивается количество атомарных операций «сравнение с обменом» (compare and swap – CAS).

Рис. 5. Хеш-таблица для хранения метаданных

Авторами предложен метод, позволяющий сократить число ложных конфликтов в STM-программах. Предполагается, что метаданные организованы в виде таблицы с прямой адресацией. Суть метода заключается в автоматической настройке параметров S и B таблицы под динамические характеристики конкретной STM-программы. Метод включает три этапа.

Этап 1. Инструментация транзакционных секций с целью профилирования**.** На первом этапе выполняется компиляция C/C++ STM-программы с использованием разработанного модуля инструментации транзакционных секций (модуль расширения GCC). В ходе статического анализа транзакционных секций STM-программ выполняется внедрение кода для регистрации обращений к функциям runtime-библиотеки.

Этап 2. Профилирование программы. На данном этапе выполняется запуск STM-программы в режиме профилирования. Профилировщик регистрирует все операции чтения/записи памяти в транзакциях. В результате формируется протокол (trace), содержащий информацию о ходе выполнения транзакционных секций:

- адрес и размер области памяти, над которой выполняется операция;
- временная метка (timestamp) начала выполнения операции.

Этап 3. Настройка параметров таблицы. По протоколу определяются средний размер W читаемой/записываемой области памяти во время выполнения транзакций. По значению W подбираются субоптимальные параметры B и S таблицы, с которыми STM-программа компилируется.

Эксперименты с тестовыми STM-программами из пакета STAMP (8 типов STM-программ), позволили сформулировать эвристические правила для подбора параметров B и S по значению W.

Значение параметра S целесообразно выбирать из множества $\{2^{18}, 2^{19}, 2^{20}, 2^{21}\}$. Значение параметра B выбирается следующим образом:

- если $W = 1$ байт, то $B = 2^4$ байт;
- если $W = 4$ байт, то $B = 2^6$ байт;
- если $W = 8$ байт, то $B = 2^7$ байт;
- если $W >= 64$ байт, то $B = 2^8$ байт.

Эксперименты. Экспериментальное исследование проводилось на вычислительной системе, оснащенной двумя четырехъядерными процессорами Intel Xeon E5420. В данных процессорах отсутствует поддержка аппаратной транзакционной памяти (Intel TSX).

В качестве тестовых программ использовались многопоточные STM-программы из пакета STAMP [12, 14, 15]. Данный пакет является общепринятым для тестирования эффективности транзакционной памяти. Число потоков варьировалось от 1 до 8. Тесты собирались компилятором GCC 5.1.1. Операционная система GNU/Linux Fedora 21 x86_64.

В рамках экспериментов измерялись значения двух показателей:

- время t выполнения STM-программы;
- количество C ложных конфликтов в программе.

На рис. 6 и 7 показана зависимость количества C ложных конфликтов и времени t выполнения теста от числа потоков при различных значениях параметров B и S. Результаты приведены для программы genome из пакета STAMP. В ней порядка 10 транзакционных секций, реализующих операции над хеш-таблицей и связными списками. Функции, в которых возникает состояние гонки за данными, вызываются внутри транзакционных секций. Видно, что увеличение значений параметров S и B приводит к уменьшению числа возможных коллизий (ложных конфликтов), возникающих при отображении адресов линейного адресного пространства процесса на записи таблицы.

При размере таблицы 2^{21} записей, на каждую из которых отображается 2^6 адресов линейного адресного пространства, достигается минимум времени выполнения теста genome, а также минимум числа ложных конфликтов.

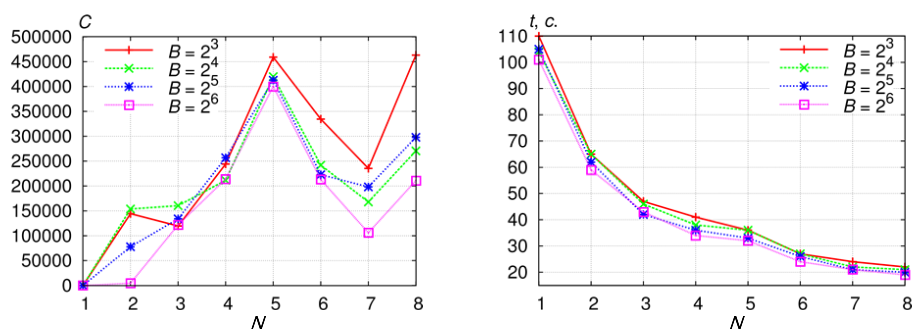

Рис. 6. Зависимость числа C ложных конфликтов (слева) и времени t выполнения теста (справа) от числа N потоков: $S = 2^{19}$

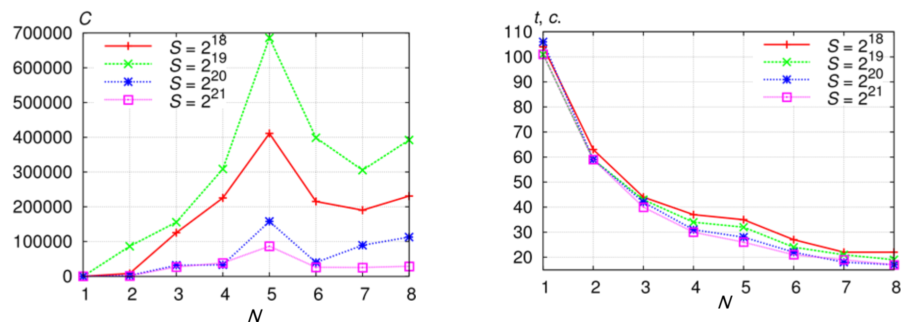

Рис. 7. Зависимость числа C ложных конфликтов (слева) и времени t выполнения теста (справа) от числа N потоков: $B = 2^6$

Время выполнения теста genome удалось сократить в среднем на 20 % за счет минимизации числа ложных конфликтов.

Заключение. В работе предложено использовать программную транзакционную память для организации спекулятивного выполнения критических секций параллельных программ. Выполнен анализ основных архитектурных аспектов реализации программной транзакционной памяти. Использование программной транзакционной памяти не требует существенных изменений параллельной програм-

мы, однако эффективное её применение ограничено. В частности, спекулятивное выполнение критических секций при помощи программной транзакционной памяти целесообразно, когда риск возникновения ситуации гонки за данными в защищаемом участке кода невелик.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Herlihy M., Shavit N.* The Art of Multiprocessor Programming. Morgan Kaufmann PublishersInc., San Francisco, CA, USA, 2008.
2. *Hendler D., Shavit N., Yerushalmi L.* A scalable lock-free stack algorithm // Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures. SPAA ’04. – 2004. – P. 206-215.
3. *Moir M. et al.* Using elimination to implement scalable and lock-free FIFO queues // Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures. ACM, 2005. – P. 253-262.
4. *Shavit N., Touitou D.* Elimination trees and the construction of pools and stacks: preliminary version // Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures. ACM, 1995. – P. 54-63.
5. *Lozi J.P. et al.* Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications // USENIX Annual Technical Conference. 2012. – P. 65-76.
6. *Afek Y., Korland G., Natanzon M., Shavit N.* Scalable Producer-Consumer Pools based on Elimination-Diffraction Trees // European Conference on Parallel Processing, 2010. – P. 151-162.
7. *Kuznetsov S.D.* Transaktsionnaya pamat. [Transactional memory]. http://citforum.ru/ programming/digest/transactional_memory/. (in Russian).
8. *Shavit N., Touitou D.* Software Transactional Memory // In PODC’95: Proceedings of thefourteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, Aug. 1995. ACM. – P. 204-213.
9. *Pascal Felber, Christof Fetzer, Patrick Marlier, and Torvald Riegel.* Time-based SoftwareTransactional Memory // IEEE Transactions on Parallel and Distributed Systems. – December 2010. – Volume 21, Issue 12. – P. 1793-1807.
10. *Torvald Riegel, Pascal Felber, and Christof Fetzer*. A Lazy Snapshot Algorithm with EagerValidation // 20th International Symposium on Distributed Computing (DISC), 2006.
11. *Victor Luchango, Jens Maurer, Mark Moir*. Transactional memory for C++ [PDF]. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3718.pdf.
12. Rochester Software Transactional Memory Runtime. Project web site [HTML]. www.cs.rochester.edu/research/synchronization/rstm/.
13. *Michael F. Spear, Luke Dalessandro, Virendra J. Marathe, and Michael L. Scott.* A comprehensive strategy for contention management in software transactional memory // In PPoPP ’09: Proc. 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, February 2009. – P. 141-150.
14. *Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, and David A.* Wood. LogTM: Log-based transactional memory. In HPCA ’06: Proc. 12th International Symposium on High-Performance Computer Architecture, February 2006. – P. 254-265.
15. *Michael F. Spear, Maged M. Michael, and Christoph von Praun*. RingSTM: scalable transactions with a single atomic instruction. In SPAA ’08: Proc. 20th Annual Symposium on Parallelism in Algorithms and Architectures, June 2008. – P. 275-284.
16. *Dave Dice, Ori Shalev, and Nir Shavit.* Transactional locking II // In DISC ’06: Proc. 20th International Symposium on Distributed Computing, September 2006. Springer Verlag Lecture Notes in Computer Science. – Vol. 4167. – P. 194-208.
17. *Pascal Felber, Christof Fetzer, Torvald Riegel*. Dynamic performance tuning of word-based software transactional memory. PPOPP 2008. – P. 237-246.
18. *Craig Zilles and Ravi Rajwar*. Implications of false conflict rate trends for robust software transactional memory // In IISWC ’07: Proc. 2007 IEEE.
19. *Olszewski M., Cutler J., Steffan J.G.* JudoSTM: A Dynamic Binary-Rewriting Approach to Software Transactional Memory // Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. PACT ’07. 2007. – P. 365-375.
20. Intel Corporation. Intel Transactional Memory Compiler and Runtime Application Binary Interface. Revision: 1.0.1, November 2008.

REFERENCES

1. *Herlihy M., Shavit N.* The Art of Multiprocessor Programming. Morgan Kaufmann PublishersInc., San Francisco, CA, USA, 2008.
2. *Hendler D., Shavit N., Yerushalmi L.* A scalable lock-free stack algorithm, *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures. SPAA '04,* 2004, pp. 206-215.
3. *Moir M. et al.* Using elimination to implement scalable and lock-free FIFO queues, *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2005, pp. 253-262.
4. *Shavit N., Touitou D.* Elimination trees and the construction of pools and stacks: preliminary version, *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*. ACM, 1995, pp. 54-63.
5. *Lozi J.P. et al.* Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications, *USENIX Annual Technical Conference,* 2012, pp. 65-76.
6. *Afek Y., Korland G., Natanzon M., Shavit N.* Scalable Producer-Consumer Pools based on Elimination-Diffraction Trees, *European Conference on Parallel Processing, 2010,* pp. 151-162.
7. *Kuznetsov S.D.* Transaktsionnaya pamat. [Transactional memory]. Available at: http://citforum.ru/programming/digest/transactional_memory/. (in Russian).
8. *Shavit N., Touitou D*. Software Transactional Memory, *In PODC'95: Proceedings of thefourteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, Aug. 1995. ACM,* pp. 204-213.
9. *Pascal Felber, Christof Fetzer, Patrick Marlier, and Torvald Riegel.* Time-based SoftwareTransactional Memory. *IEEE Transactions on Parallel and Distributed Systems,* December 2010, Vol. 21, Issue 12. pp. 1793-1807.
10. *Torvald Riegel, Pascal Felber, and Christof Fetzer*. A Lazy Snapshot Algorithm with EagerValidation, *20th International Symposium on Distributed Computing (DISC), 2006*.
11. *Victor Luchango, Jens Maurer, Mark Moir*. Transactional memory for C++ [PDF]. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3718.pdf.
12. Rochester Software Transactional Memory Runtime. Project web site [HTML]. www.cs.rochester.edu/research/synchronization/rstm/.
13. *Michael F. Spear, Luke Dalessandro, Virendra J. Marathe, and Michael L. Scott.* A comprehensive strategy for contention management in software transactional memory, *In PPoPP '09: Proc. 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, February 2009,* pp. 141-150.
14. *Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, and David A.* Wood. LogTM: Log-based transactional memory, *In HPCA '06: Proc. 12th International Symposium on High-Performance Computer Architecture, February 2006,* pp. 254-265.
15. *Michael F. Spear, Maged M. Michael, and Christoph von Praun*. RingSTM: scalable transactions with a single atomic instruction, *In SPAA '08: Proc. 20th Annual Symposium on Parallelism in Algorithms and Architectures, June 2008,* pp. 275-284.
16. *Dave Dice, Ori Shalev, and Nir Shavit.* Transactional locking II, *In DISC '06: Proc. 20th International Symposium on Distributed Computing, September 2006*. Springer Verlag Lecture Notes in Computer Science, Vol. 4167, pp. 194-208.
17. *Pascal Felber, Christof Fetzer, Torvald Riegel*. Dynamic performance tuning of word-based software transactional memory. PPOPP 2008, pp. 237-246.
18. *Craig Zilles and Ravi Rajwar*. Implications of false conflict rate trends for robust software transactional memory, *In IISWC '07: Proc. 2007 IEEE*.
19. *Olszewski M., Cutler J., Steffan J.G.* JudoSTM: A Dynamic Binary-Rewriting Approach to Software Transactional Memory, *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. PACT '07. 2007,* pp. 365-375.
20. Intel Corporation. Intel Transactional Memory Compiler and Runtime Application Binary Interface. Revision: 1.0.1, November 2008.

Статью рекомендовал к опубликованию д.т.н. Э.В. Мельник.

Кулагин Иван Иванович – Федеральное государственное бюджетное учреждение науки Институт физики полупроводников им. А.В. Ржанова Сибирского отделения Российской академии наук; e-mail: ivan.i.kulagin@gmail.com; 630090, г. Новосибирск, проспект Академика Лаврентьева, 13; тел.: +79137720919; инженер программист лаборатории Вычислительных систем.

Курносов Михаил Георгиевич – e-mail: mkurnosov@gmail.com; к.т.н.; научный сотрудник лаборатории Вычислительных систем.

Kulagin Ivan Ivanovich – Rzhanov Institute of Semiconductor Physics Siberian Branch of Russian Academy of Sciences; e-mail: ivan.i.kulagin@gmail.com; 13, Lavrentev ave., Novosibirsk, 630090, Russia; phone: +79137720919; software engineer.

Kurnosov Mikhail Georgievich – e-mail: mkurnosov@gmail.com; cand. of eng. sc.; research scientist