УДК 004.272

Исследование эффективности потокобезопасных ассоциативных массивов на базе программно- и аппаратно-реализуемой транзакционной памяти

Д. М. Берлизов, М. А. Позднышев, Р. В. Терешков, А. А. Пазников[1]

В работе проведен сравнительный анализ существующих реализаций программной и аппаратной транзакционной памяти при разработке потокобезопасных ассоциативных массивов. Алгоритмы реализованы на языке C++ и исследованы на кластерной вычислительной системе с аппаратной поддержкой транзакционной памяти. Построены рекомендации по использованию транзакционной памяти при разработке масштабируемых структур данных.

Ключевые слова: параллельное программирование, транзакционная память, многопоточное программирование, вычислительные системы, STM, HTM.

1. Введение

С развитием многоядерных вычислительных систем (ВС) [1] с общей памятью (SMP и NUMA-системы) особую актуальность приобретает решение проблем, связанных с организацией доступа параллельных потоков к разделяемым данным. В настоящее время наиболее распространённым способом решения данной проблемы является использование блокировок. Разработка параллельных алгоритмов в данной методологии характеризуется незначительными трудозатратами и высокой эффективностью, но имеет такие недостатки, как последовательное выполнение критических секций (serialization), взаимные блокировки (deadlocks), инверсии приоритетов (priority inversion), последовательный захват блокировки (convoying) и голодание (starvation). Наравне с использованием блокировок развивается альтернативное направление: семейство алгоритмов и структур данных, свободных от блокировок (lock-free). Данные методы позволяют достичь большей масштабируемости и обеспечивают гарантированное время выполнения. Перспективным подходом к программированию без использования блокировок является транзакционная память (transactional memory) [2].

Существующие реализации транзакционной памяти включают в себя как программно- [6, 7, 8], так и аппаратно-реализуемые алгоритмы [3, 4, 5]. Также следует отметить реализации, сочетающие в себе аппаратные и программные средства [9, 10, 11]. Данная работа концентрируется на исследовании программной транзакционной памяти, реализованной в компиляторе GCC, а также расширений набора инструкций в процессорах Intel микроархитектуры Haswell (технологии Hardware Lock Elision, Restricted Transactional Memory).

[1] Работа выполнена при поддержке РФФИ (гранты № 15-07-02693, 15-07-00653).

2. Программно-реализуемая транзакционная память

Транзакционная модель памяти предполагает выделение групп инструкций в атомарные транзакции, аналогичные транзакциям в базах данных. Под транзакцией в данном контексте понимается выполняемая потоком конечная последовательность инструкций, реализующая операции транзакционных чтения и записи. Транзакции удовлетворяют свойствам сериализуемости и атомарности. Сериализуемость (serializability) предполагает последовательное выполнение транзакций, не приводящее к состояниям состязания (contention). Условие атомарности (atomicity) гарантирует, что транзакция вступает в силу мгновенно, при этом все изменения в памяти единовременно становятся видимыми для всех потоков. В общем случае, отдельно взятая транзакция выполняется полностью и успешно, либо не выполняется вообще, если возникает конфликт данных (рис. 1). Результаты успешно выполненных транзакций записываются в специальный лог – журнал транзакций. Несомненными достоинствами транзакционной памяти, по сравнению с традиционным подходом на основе операции Compare-and-swap (CAS) являются простота использования. Алгоритмы на основе транзакционной памяти свободны от блокировок и обеспечивают прирост ускорения за счёт одновременного выполнения транзакций. К недостаткам относится сложность отладки программ, ограниченная область применения транзакционной памяти, а также падение производительности при неправильном использовании. Текущие реализации транзакционной памяти характеризуются высокими накладными расходами на создание и управление транзакциями.

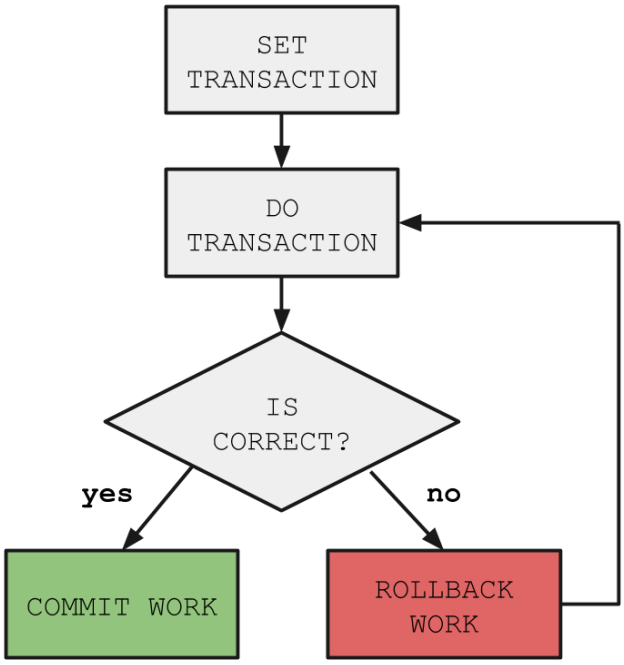


Рис.1. Обобщённый вид выполнения транзакции

Транзакционная память реализована программно (например, в компиляторе GCC, начиная с версии 4.7). Также существует аппаратная поддержка некоторыми семействами процессоров (например, Intel Haswell x86).

Основная идея реализации программной транзакционной памяти в компиляторе GCC состоит в том, что критические секции помещается в блок *__transactional_atomic {...}*, указывающий на то, что данный участок кода является транзакцией (рис. 2).

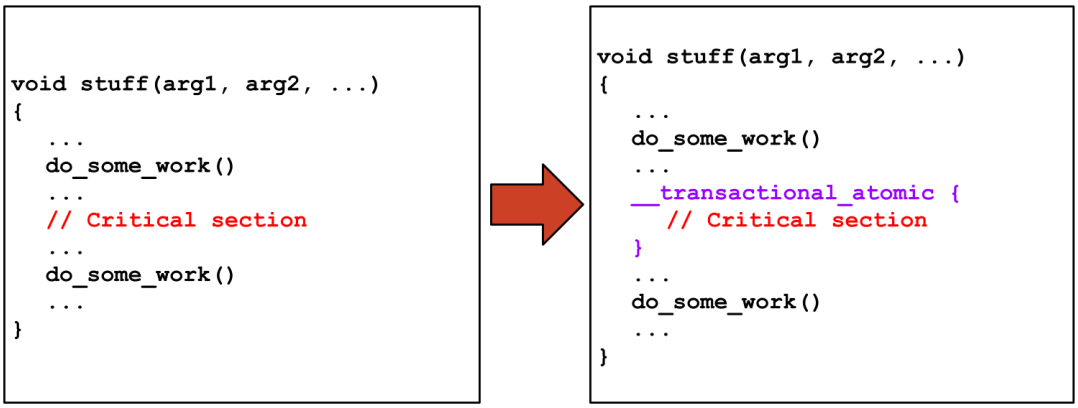


Рис. 2. Реализация транзакций в компиляторе GCC

3. Аппаратно-реализуемая транзакционная память

Аппаратная поддержка транзакционной памяти заключается во введении новых команд в набор инструкций микропроцессора. Например, в процессорах Intel микроархитектуры Haswell введены так называемые расширения транзакционной синхронизации (Transactional Synchronization Extensions, TSX). Одним из таких расширений Restricted Transactional Memory (RTM).

Технология RTM (рис. 3, 4) предполагает выполнение резервного кода (fallback code) в случае, если транзакция не может быть выполнена успешно. Вначале задаётся некоторый лимит на количество попыток запуска транзакции. Если ни одна из этих попыток не заканчивается успешно, алгоритм переходит к выполнению резервного кода, гарантирующего выполнение транзакции, чаще всего – на основе блокировок. Расширение RTM вводит в набор инструкций микропроцессора следующие команды: XBEGIN и XEND, обозначающие начало и конец критической секции, и XABORT, явно отменяющая выполнение транзакции.

В работе предлагаются потокобезопасные ассоциативные массивы (хеш-таблицы и бинарные деревья поиска) на основе программной транзакционной памяти (Software Transactional Memory, STM), реализованной в компиляторе GCC версии 4.9, а также с применением набора инструкций RTM для использования аппаратной поддержки транзакционной памяти в процессорах Intel Core микроархитектуры Haswell. Проведено сравнение разработанных структур данных с аналогичными реализациями на основе блокировок. В докладе представлены результаты анализа эффективности созданных ассоциативных массивов и рекомендации по применению различных реализаций транзакционной памяти. Показано (рис. 5), что использование программной транзакционной памяти позволяет повысить пропускную способность хеш-таблиц и деревьев поиска по сравнению с потокобезопасными структурами данных на основе блокировок.

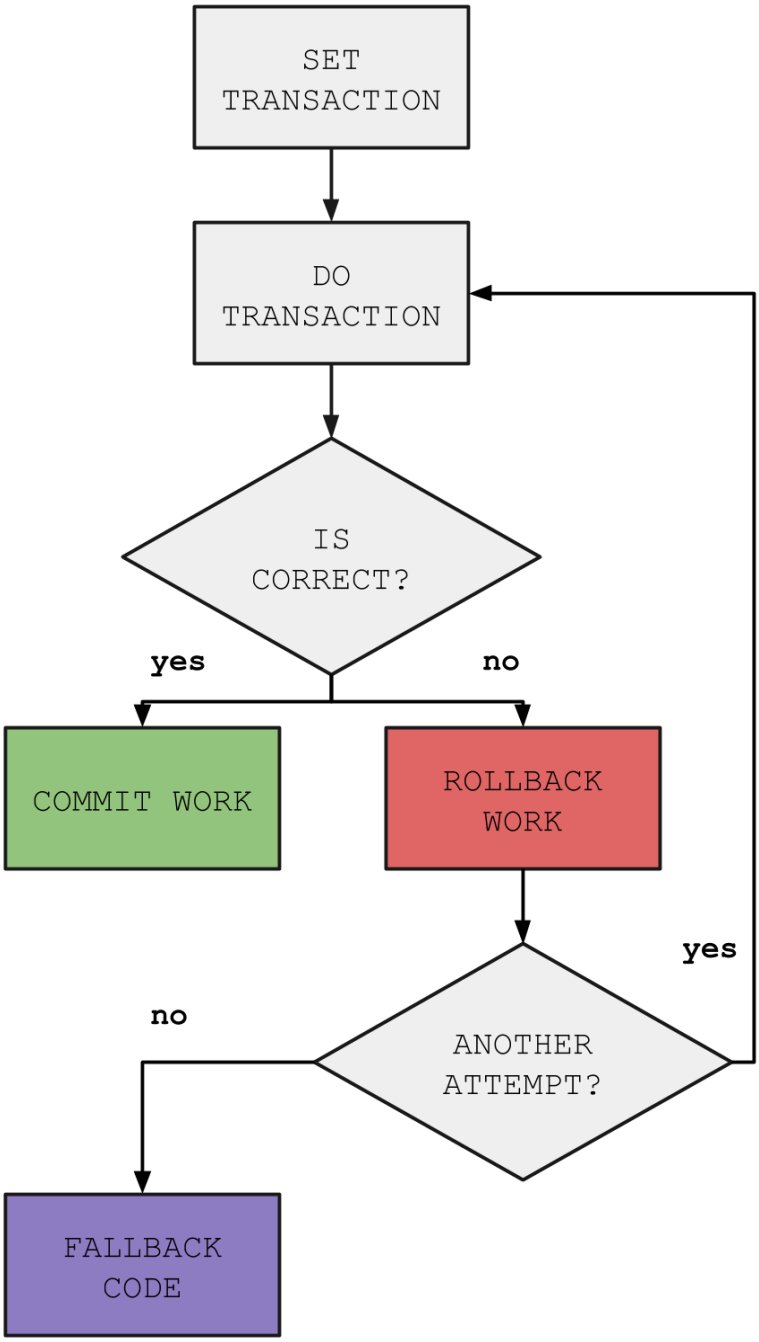
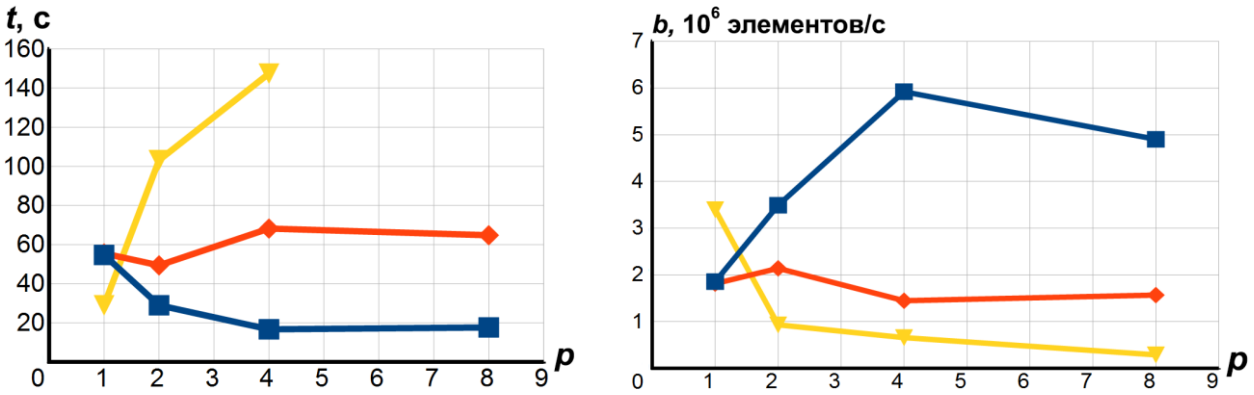


Рис. 3. Выполнение транзакции в RTM

```
void RTM() {
  while (1){
    start_transaction()
    if (TRANSACTION_SUCCESS)
      return
    else
      attempts_counter++
    if (attempts_counter > limit)
      break
  }
  fallback_code()
}
```

Рис. 4. Псевдокод алгоритма RTM



а *б*

Рис. 5. Результаты сравнения структур данных, разработанных на базе программно- и аппаратно-реализуемой транзакционной памяти, с реализациями на основе блокировок:
а – зависимость времени вставки (t) 10^8 элементов в бинарное дерево поиска от числа используемых потоков (p);
б – зависимость пропускной способности (b) при вставке 10^8 элементов в бинарное дерево поиска от числа используемых потоков (p).

■ – реализация на основе STM в GCC;

▼ – реализация на основе алгоритма RTM;

♦ – реализация на основе блокировок (мьютексов).

Литература

1. *Хорошевский В. Г.* Распределённые вычислительные системы с программируемой структурой // Вестник СибГУТИ. 2010. № 2 (10). С. 3−41.
2. *Herlihy M., Shavit N.* Transactional memory: beyond the first two decades // ACM SIGACT News. – 2012. V. 43. № 4. P. 101-103.
3. *Ananian C. S. et al.* Unbounded transactional memory // High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on. IEEE, 2005. P. 316-327.
4. *Rajwar R., Herlihy M., Lai K.* Virtualizing transactional memory // Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on. – IEEE, 2005. – С. 494-505.
5. *Yen L. et al.* LogTM-SE: Decoupling hardware transactional memory from caches // High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on. IEEE, 2007. P. 261-272.
6. *Herlihy M. et al.* Software transactional memory for dynamic-sized data structures // Proceedings of the twenty-second annual symposium on Principles of distributed computing. ACM, 2003. P. 92-101.
7. *Dice D., Shalev O., Shavit N.* Transactional locking II // Distributed Computing. – Springer Berlin Heidelberg, 2006. P. 194-208.
8. *Marathe V. J., Scherer III W. N., Scott M. L.* Adaptive software transactional memory // Distributed Computing. – Springer Berlin Heidelberg, 2005. P. 354-368.
9. *Damron P. et al.* Hybrid transactional memory // ACM Sigplan Notices. ACM, 2006. V. 41. – №. 11. P. 336-346.
10. *Lev Y., Maessen J. W.* Split hardware transactions: true nesting of transactions using best-effort hardware transactional memory // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. ACM, 2008. P. 197-206.

11. *Baugh L., Neelakantam N., Zilles C.* Using hardware memory protection to build a high-performance, strongly-atomic hybrid transactional memory // ACM SIGARCH Computer Architecture News. 2008. V. 36. №. 3. P. 115-126.

Статья поступила в редакцию 28.02.2015;
переработанный вариант — 13.03.2015

Берлизов Даниил Михайлович
студент 3 курса факультета информатики и вычислительной техники СибГУТИ (630102, Новосибирск, ул. Кирова, 86), тел. +7 (913) 466-95-47, e-mail: sillyhat34@gmail.com

Позднышев Максим Андреевич
студент 3 курса факультета информатики и вычислительной техники СибГУТИ (630102, Новосибирск, ул. Кирова, 86), тел. +7 (952) 915-85-00, e-mail: maxpozdnishev@gmail.com

Терешков Роман Вадимович
студент 3 курса факультета информатики и вычислительной техники СибГУТИ (630102, Новосибирск, ул. Кирова, 86), тел. +7 (923) 150-97-79, e-mail: rtereshkov@gmail.com

Пазников Алексей Александрович
к.т.н., доцент кафедры вычислительных систем СибГУТИ (630102, Новосибирск, ул. Кирова, 86) тел. (383) 2-698-286, e-mail: apaznikov@gmail.com.

Efficiency analysis of thread-safe associative arrays based on software and hardware transactional memory

D. Berlizov, M. Pozdnyshev, R. Tereshkov, A. Paznikov

This paper presents a comparison study of currently existing implementations of software and hardware transactional memory for scalable data structures development. Proposed algorithms are implemented in C++ and evaluated on a computer cluster with hardware transactional memory support. Finally, recommendations are provided for using transactional memory for scalable data structures development.

Keywords: parallel programming, transactional memory, multithreading, computer systems, STM, HTM