4. *Четверушкин Б.Н.* К вопросу об ограничении снизу на масштабы в механике сплошной среды // Время, хаос, математические проблемы. – М.: Изд-во МГУ, 2009. – Вып. 4. – С. 75-96.
5. *Morozov D.N., Chetverushkin B.N., Churbanova N.G., Trapeznikova M.A.* An Explicit Algorithm for Porous Media Flow Simulation using GPUs // Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Civil-Comp Press, Stirlingshire, UK, 2011. – P. 19
6. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. – М.: ДМК-Пресс, 2010. – 232 с.

Статью рекомендовал к опубликованию к.ф.-м.н.; доцент Е.В. Шильников.

Морозов Дмитрий Николаевич – Московский физико-технический институт; e-mail: dmitry.morozov@phystech.edu; 141700, Московская область, г. Долгопрудный, Институтский пер., 9; тел.: +74954084554; аспирант.

Четверушкин Борис Николаевич – Институт прикладной математики им. М.В. Келдыша РАН; e-mail: chetver@imamod.ru; 125047, г. Москва, Миусская пл., 4; тел.: +74999781314; академик; директор.

Чурбанова Наталья Геннадиевна – e-mail: nata@imamod.ru; 125047, г. Москва, Миусская пл., 4: тел.: +74999781314; с.н.с.

Трапезникова Марина Александровна – e-mail: marina@imamod.ru; 125047, г. Москва, Миусская пл., 4; тел.: +74999781314; с.н.с.

Morozov Dmitry Nikolaevich – Moscow Institute of Physics and Technology; e-mail: dmitry.morozov@phystech.edu; 9, Institutskii per., Dolgoprudny, 141700, Moscow Region, Russia; phone: +74954084554; postgraduate student.

Chetverushkin Boris Nikolaevich – Keldysh Institute of Applied Mathematics; e-mail: chetver@imamod.ru; 4, Miusskaya sq., Moscow 125047, Russia; phone: +74999781314; academician; director.

Churbanova Natalia Gennadievna – e-mail: nata@imamod.ru; 4, Miusskaya sq., Moscow, 125047, Russia, phone: +74999781314; senior scientist.

Trapeznikova Marina Aleksandrovna – e-mail: marina@imamod.ru chetver@imamod.ru; 4, Miusskaya sq., Moscow, 125047, Russia; phone: +74999781314; senior scientist.

УДК 004.272.2

И.Г. Данилов

ОБ ОДНОМ ПОДХОДЕ К РЕАЛИЗАЦИИ ПРОГРАММНОЙ ТРАНЗАКЦИОННОЙ ПАМЯТИ ДЛЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

Транзакционная память (Transactional Memory, TM) предоставляет неблокирующий синхронизационный управляющий механизм для многопоточных приложений. В настоящее время существует большое количество реализаций (программных, аппаратных и гибридных) данного механизма для систем с общей памятью. Актуальным является исследование возможности применения транзакционной памяти к построению распределённых вычислительных систем. В данной работе описывается система программной транзакционной памяти DSTM_P1, с помощью которой можно запускать многопоточное приложение, написанное на языке C, на многоядерном кластере.

Распределённые вычисления; распределённая транзакционная память.

I.G. Danilov

ON ONE APPROACH TO IMPLEMENT SOFTWARE TRANSACTIONAL MEMORY FOR DISTRIBUTED COMPUTING

Transactional memory provides non-blocking concurrency control mechanism for multithreaded applications. There are a lot of research on this topic for shared memory architectures: as for software TM (STM), as for hardware TM (HTM) and hybrid TM (HyTM). Transactional Memory seems to be promising synchronization technique for distributed computing systems. In this work, distributed software transactional memory system called DSTM_P1 was presented. DSTM_P1 provides a possibility to execute multithreaded C-application on high perfomance cluster.

Distributed computing; distributed transactional memory.

В настоящее время транзакционная память является объектом активных научных исследований в области синхронизации параллельных вычислительных процессов. Связано это прежде всего с желанием упростить существующие модели параллельного и распределённого программирования без потери или с возможным увеличением производительности приложений. Ведущие производители процессорных устройств уже заявили поддержку механизмов транзакционной памяти в ряде моделей процессоров будущих поколений [1, 2]. Отечественных работ в данной области практически нет. Большая часть зарубежных работ посвящена системам с общей памятью [3]; для систем с распределённой памятью существует всего несколько рабочих программных реализаций [4, 5].

Целью работы является разработка и исследование методов и алгоритмов запуска, выполнения и неблокирующей синхронизации потоков на вычислительных кластерных системах с использованием механизмов транзакционной памяти и протокола удаленного доступа к памяти.

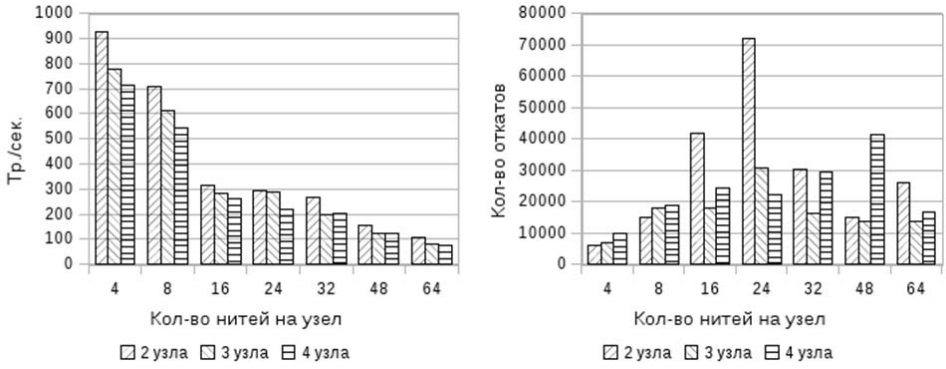
Транзакционная память. Транзакционная память – это альтернативный традиционным методам синхронизации механизм, позволяющий частям программы выполняться в изоляции, независимо от других параллельно выполняемых задач [6]. В отличие от традиционных методов синхронизации, в данном случае конкурирующие вычислительные потоки никогда не блокируются в критической секции (атомарной секции – для транзакционной памяти), а продолжают вычисления *изолированно* и изменяют общую память напрямую (ведется лог «отмены» изменений) либо все изменения в памяти записываются в промежуточный буфер. В момент *фиксации/коммита* изменений (в момент выхода из атомарной секции) производится проверка на наличие ситуации конфликта между двумя или несколькими транзакциями. Конфликт возникает, если транзакции получают доступ к одним и тем же данным и одна из конфликтующих транзакций производит запись в эти данные. В зависимости от определенных условий менеджер разрешения конфликтов *откатывает* одну из транзакций и отменяет все изменения, сделанные этой транзакцией в общей памяти.

Система DSTM_P1. Автором разработана система программной распределённой транзакционной памяти DSTM_P1 [7], с помощью которой можно запускать написанное на языке C многопоточное приложение на многоядерном кластере с высокоскоростным межсистемным соединением (Infiniband и др.). В отличие от традиционного подхода к синхронизации (использование мьютексов, семафоров) участки доступа к разделяемым данным в приложении помечаются как атомарные (конструкция __tm_atomic{}). Разработанный алгоритм распределённой транзакционной памяти является модификацией алгоритма TFA [5] и использует независимые локальные часы на каждом узле кластера, значение которых используется для версионирования данных. Вся доступная приложению динамическая память разбивается на блоки CDU (англ. *Conflict Detection Unit*) размером с определенное на этапе компиляции количество байт (степень 2-ки). Каждый такой блок ассоциируется с

8-байтовым целым числом – «блокировкой», в котором хранится либо версия блока, если «блокировка» не занята, либо указатель на метаданные транзакции, владеющей «блокировкой» (флагом занятости является старший бит числа). Для предотвращения потери обновления и чтения «грязных» данных используется промежуточный буфер, в который записываются все изменения, производимые транзакцией над блоками данных. Новые версии становятся видны другим транзакциям в момент *фиксации* транзакции, только после успешной последовательности запросов на блокировку для всех измененных блоков памяти. Для предотвращения неповторяемого считывания в каждой транзакции ведется список начальных версий (в момент доступа) всех считанных блоков. Далее на стадии фиксации этот список проверяется на наличие изменений в версии: если версия блока из списка изменилась, значит, другая транзакция успела перезаписать прочитанный ранее блок данных и необходимо *откатить* транзакцию на начало выполнения (начало конструкции __tm_atomic{}). Так как в алгоритме для версионирования используются независимые на каждом узле локальные часы, то для возможности корректной проверки версий считанных блоков необходимо построить соотношение частичной упорядоченности между значимыми событиями в системе. В разработанном алгоритме таким событием является считывание блока данных. Транзакция, которая читает блок данных, находящийся на удалённом узле, увеличивает значение локальных часов, если оно меньше, чем значение версии считываемого блока. Считывание производится блоками CDU с помощью односторонних RDMA-операций (англ. *Remote Direct Memory Access*). Блоки данных с удалённых узлов кэшируются.

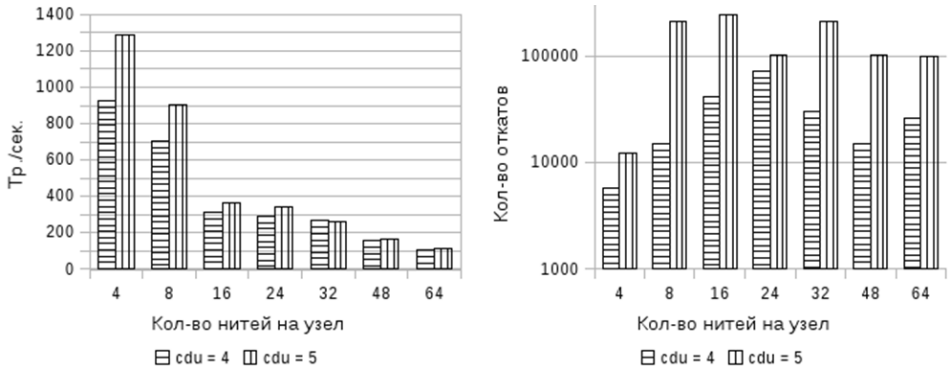
Общая схема работы DSTM_P1 может быть описана следующим образом: 1) пользователь системы загружает исходные коды многопоточного приложения, написанного на языке C с использованием библиотеки Pthread и применением атомарных конструкций (__tm_atomic{}) в местах доступа к разделяемым данным; 2) приложение компилируется с помощью DTMC [8] в язык промежуточного представления; 3) далее полученное представление трансформируется в представление, содержащее соответствующие вызовы функций библиотеки транзакционной памяти для всех инструкций доступа к памяти атомарного блока; 4) представление, полученное на предыдущем шаге, линкуется с необходимыми библиотеками (транзакционной памяти, «обертками» системных библиотек pthread и malloc); 5) на лету компилируется и исполняется main-функция приложения; 6) все вызовы функций библиотеки pthread и malloc переадресуются в соответствующие вызовы функций библиотек «обёрток». Во время исполнения main-функции при вызове pthread_create определяется код функции потока, который распределяется в системе с помощью модуля балансировки нагрузки и исполняется в отдельном потоке на менее загруженном узле кластера.

Результаты экспериментальных исследований. Для проведения экспериментов было адаптировано существующее многопоточное Pthread-приложение, в функции потока которого в течение заданного промежутка времени выполняются различные операции над списком целочисленных значений: поиск заданного значения и обновление списка (добавление/удаление элементов), при начальном размере списка 256 и количестве операций обновления равном 20 % от общего числа. Эксперименты проводились на кластере HP (на одном узле два 4-ядерных процессора Intel Xeon, узлы соединены Infiniband 4x DDR). На рис. 1 представлены результаты в зависимости от количества узлов в системе при изначальном распределении списка равными частями по двум узлам. На рис. 2, 3 представлены результаты, в зависимости от размера блока данных CDU (2^cdu), для двух (рис. 2) и четырех (рис. 3) узлов, при этом список распределён равными частями – по двум узлам, рис. 2; по четырем узлам, рис. 3.



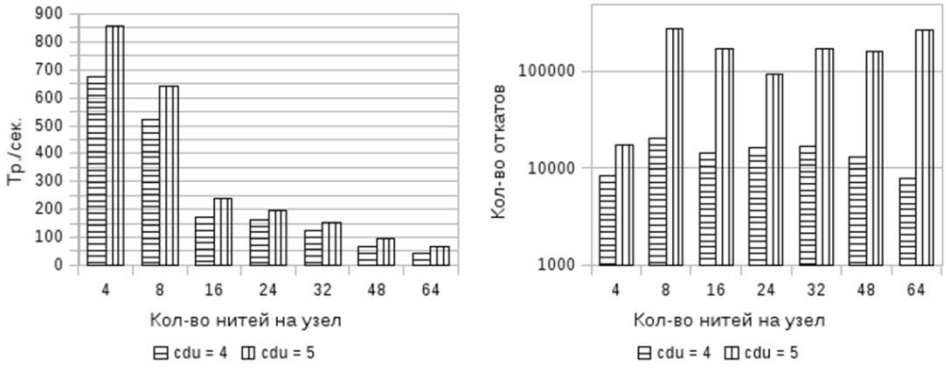
а б

Рис. 1. Для списка, изначально распределённого по двум узлам: а – транзакций в секунду; б – общее количество откатов



а б

Рис. 2. Для двух узлов при разном размере CDU (16,32 байт): а – транзакций в секунду; б – общее количество откатов (лог. масштаб)



а б

Рис. 3. Для четырех узлов при разном размере CDU (16,32 байт): а – транзакций в секунду; б – общее количество откатов (лог. масштаб)

Увеличение производительности при увеличении размера блока данных CDU (рис. 2,а, 3,а) можно объяснить тем, что уменьшается количество RDMA-операций при доступе к списку, так как за один раз считывается большее количество элементов списка (размер элемента в данном случае равен 16 байтам). Большее количество элементов списка в блоке CDU приводит и к увеличению откатов транзакций (рис. 2,б, 3,б).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Reinders J.* Transactional Synchronization in Haswell // Intel Software Network. URL: http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell/ (дата обращения: 9.05.2012).
2. *Merritt R.* IBM plants transactional memory in CPU // Intel Software Network. URL: http://www.eetimes.com/electronics-news/4218914/IBM-plants-transactional-memory-in-CPU (дата обращения: 9.05.2012).
3. *Grahn H.* Transactional Memory // In: J. Parallel Distrib. Comput. – 2010. – Vol. 70 (10). – P. 993-1008.
4. *Bocchino R.L., Adve V.S., Chamberlain B.L.* Software transactional memory for large scale clusters // In Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Prog. (PPOPP 2008), 20–23 February 2008, Salt Lake City, UT, USA. – P. 247-258.
5. *Saad M.M., Ravindran B.* Transactional Forwarding Algorithm: Technical Report // ECE Dept., Virginia Tech, January 2011.
6. *Larus J., Kozyrakis C.* Transactional Memory // In: Communications of the ACM. – 2008. – Vol. 51 (7). – P. 80-88.
7. *Данилов И.Г.* Прототип распределённой программной транзакционной памяти DSTM_P1 // Высокопроизводительные параллельные вычисления на кластерных системах: Материалы XI Всероссийской конференции (Н. Новгород, 2–3 ноября 2011 г.) / Под ред. проф. В.П. Гергеля. – Н. Новгород: Изд-во Нижегородского госуниверситета, 2011. – С. 102-107.
8. Velox Project. Dresden TM Compiler // URL: http://www.velox-project.eu/software/dtmc (дата обращения: 9.05.2012).

Статью рекомендовал к опубликованию д.т.н. Н.И. Витиска.

Данилов Игорь Геннадьевич – Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге; e-mail: vainamon@gmail.com; 347928, г. Таганрог, пер. Некрасовский, 44, ГСП 17А; тел.: 88634371773; кафедра математического обеспечения и применения ЭВМ; аспирант.

Danilov Igor Gennad'evich – Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education “Southern Federal University”; GSP 17A, 44, Nekrasovskiy, Taganrog, 347928, Russia; e-mail: vainamon@gmail.com; phone: +78634371773; the department of software engineering; postgraduate student.

УДК 004.724.4

Д.В. Кутузов, А.В. Осовский, Е.А. Моторина

ПАРАЛЛЕЛЬНАЯ КОММУТАЦИЯ В МНОГОЗВЕННЫХ СХЕМАХ

Рассматриваются проблемы построения параллельных пространственных коммутационных систем. Проводится анализ недостатков кроссовых схем и возможности реализации параллельной коммутации в многозвенных схемах. В статье описана структура параллельного кроссового коммутатора, который является базовым при построении многозвенных схем. Описывается структура коммутационного элемента параллель-