УДК 004.272
DOI: 10.17223/19988605/38/8

А.А. Пазников

ОПТИМИЗАЦИЯ ДЕЛЕГИРОВАНИЯ ВЫПОЛНЕНИЯ КРИТИЧЕСКИХ СЕКЦИЙ НА ВЫДЕЛЕННЫХ ПРОЦЕССОРНЫХ ЯДРАХ

Работа выполнена при поддержке РФФИ (*гранты № 15-07-02693, 15-37-20113, 15-07-00653, 16-07-00712, 15-07-00048*).

В работе рассмотрена задача оптимизации метода делегирования выполнения критических секций на выделенных процессорных ядрах (Remote Core Locking – RCL). Данный метод позволяет повысить масштабируемость многопоточных программ за счёт снижения конкурентности доступа к критическим секциям и пространственной локализации обращений к памяти. Предложен алгоритм оптимизации выделения памяти в программах на основе RCL в многоядерных системах с неоднородным доступом к памяти. Выполнено исследование на кластерной вычислительной системе. Показано, что оптимизация достигается благодаря выделению памяти на NUMA-узлах, на которых функционирует RCL-сервер.

Ключевые слова: RCL; многопоточное программирование; критические секции; масштабируемость.

В настоящее время имеет место значительное увеличение количества процессорных ядер в вычислительных системах (ВС) с общей памятью [1], относящихся к классам SMP и NUMA. В связи с этим возрастает потребность в создании масштабируемых параллельных программ, эффективных при большом количестве параллельных потоков.

Среди перспективных подходов, позволяющих обеспечить высокую масштабируемость параллельных программ, следует в первую очередь выделить алгоритмы и структуры данных, свободные от блокировок (lock-free) [2], и программную транзакционную память (software transactional memory) [3]. Недостатками неблокируемых алгоритмов и структур данных являются ограниченная область их применения и существенные трудозатраты, связанные с разработкой параллельных программ. Кроме того, пропускная способность таких структур часто сопоставима с аналогичными структурами данных, основанными на блокировках. Программная транзакционная память на сегодняшний день не обеспечивает достаточный уровень производительности многопоточных программ и не получила широкого применения в реальных приложениях.

Традиционный подход организации критических секций в многопоточных программах с помощью блокировок сегодня остаётся наиболее распространённым в многопоточном программировании. Блокировки проще в использовании по сравнению с неблокируемыми алгоритмами и структурами данных и обеспечивают высокий уровень производительности. Кроме того, основная часть существующих многопоточных программ разработана с использованием блокировок. Поэтому востребованным является создание масштабируемых алгоритмов реализации взаимного исключения.

Масштабируемость блокировок существенно зависит от решения проблем, связанных с конкурентным доступом (access contention) параллельных потоков в разделяемых областях памяти и пространственной локализацией обращений к кэш-памяти (cache locality). Конкурентный доступ имеет место при одновременном обращении нескольких потоков к критической секции, защищённой одним объектом синхронизации, что приводит к увеличению загруженности кэш-памяти. Локальность кэша имеет существенное значение в том случае, когда внутри критической секции выполняется обращение к разделяемым данным, которые перед этим использовались на другом ядре. Данное обстоятельство приводит к промахам по кэшу (cache misses) и значительному увеличению времени выполнения критических секций.

Среди наиболее эффективных подходов к реализации масштабируемых блокировок можно выделить CAS spinlocks, MCS-locks [4], Flat combining [5], CC-Synch [6], DSM-Synch [6], Oyama lock [7]. От-

дельно необходимо выделить ряд методов, предполагающих делегирование выполнения критических секций выделенному процессорному ядру для повышения локальности кэша [5, 8–10]. Работы [9, 10] посвящены разработке потокобезопасных структур данных (списков и хеш-таблиц) на основе делегирования выполнения критических секций. В статье [8] предлагается универсальное аппаратное решение, включающее набор процессорных инструкций для передачи управления выделенному ядру процессора. Flat Combining [5] относится к программно-реализуемым методам. В роли сервера, выполняющего критические секции, выступают все потоки. Однако выполнение процедуры передачи управления критической секцией между потоками приводит к снижению производительности даже при незначительном конкурентном доступе. Кроме того, данные алгоритмы не поддерживают блокировки потоков внутри критических секций как вследствие активного ожидания, так и из-за блокировки потока на уровне ядра операционной системы.

1. Метод делегирования выполнения критических секций на выделенных процессорных ядрах

Время выполнение критических секций в многопоточных программах складывается из времени передачи права выполнения блокировкой и времени выполнения инструкций критической секции (рис. 1). При передаче права владения блокировкой возникают накладные расходы вследствие переключения контекста, загрузки кэш-линии, содержащей глобальную переменную синхронизации, из оперативной памяти, активизации потока, выполняющего критическую секцию. На время выполнения инструкций критической секции существенно влияет пространственная локализация обращений к глобальным переменным. Традиционные алгоритмы взаимного исключения предполагают частое переключение контекста, которое приводит к вытеснению разделяемых переменных из кэш-памяти.

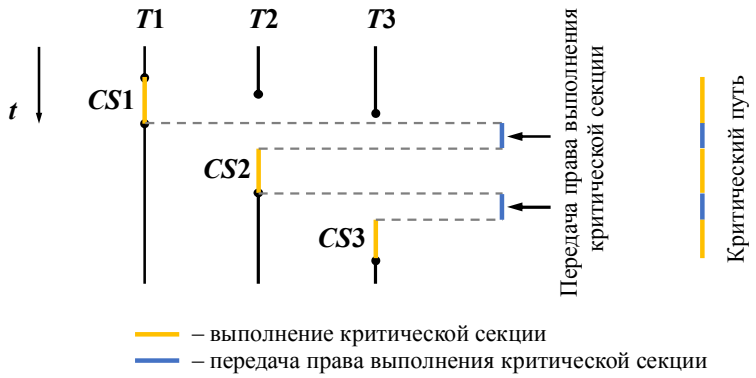
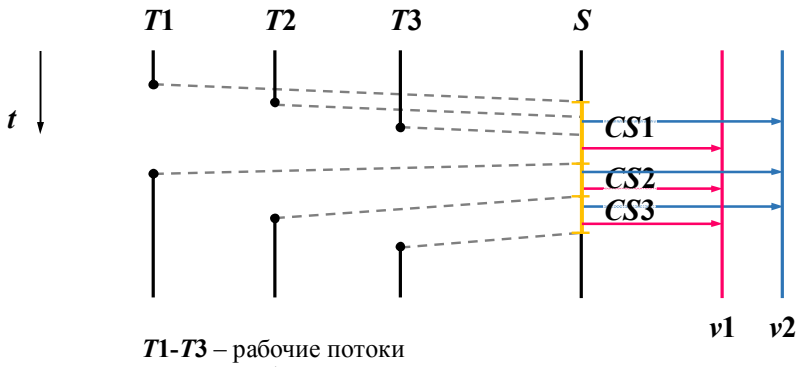


Рис. 1. Критический путь выполнения критических секции

В работе рассматривается метод выполнения критических секций на выделенных узлах процессора (Remote Core Locking – RCL) [11], позволяющий минимизировать время выполнения существующих многопоточных программ за счет уменьшения критического пути выполнения критических секций.

Данный метод предполагает замену в существующих программах высоконагруженных критических секций на удалённый вызов процедур для выполнения на выделенных процессорных ядрах (рис. 2). Все критические секции, защищённые одной блокировкой, выполняются потоком-сервером, работающим на выделенном ядре. Поскольку критические секции выполняются в отдельном потоке, отсутствуют накладные расходы на передачу права владения критической секцией.

RCL также позволяет сократить время выполнения инструкций критических секций. Это достигается тем, что разделяемые данные, защищённые блокировкой, с большой вероятностью находятся в локальной кэш-памяти ядра, на котором функционирует RCL-сервер, что обеспечивает снижение числа промахов по кэшу. Конкурентность доступа снижается за счёт использования отдельной выделенной кэш-линии для каждого потока-клиента, которая отвечает за хранение информации о критической секции и активного ожидания потока-клиента (рис. 3).



***T*1-*T*3** – рабочие потоки
S – сервер, функционирующий на выделенном ядре
***CS*1-*CS*3** – выполнение критических секций
***v*1, *v*2** – разделяемые переменные

Рис. 2. Делегирование выполнения критических секций на выделенном процессорном ядре

Рис. 3. Таблица запросов на выполнение критических секций в RCL

В существующей реализации RCL не учитывается неоднородный доступ к памяти в NUMA-системах. В данной работе предлагаются алгоритмы, позволяющие минимизировать время доступа к памяти в системах с неоднородным доступом к памяти при выполнении многопоточных программ на базе RCL.

2. Оптимизация алгоритмов делегирования выполнения критических секций в NUMA-системах

В настоящее время вычислительные системы с неоднородным доступом к общей памяти (Non-uniform memory access – NUMA) (рис. 4) получили широкое распространение. Такие системы представляют собой композиции многоядерных процессоров, каждый из которых связан с локальной областью глобальной памяти. Процессор вместе с его локальной памятью формирует NUMA-узел (NUMA-node). Взаимодействие между процессорами и обращение к памяти осуществляются через шину (AMD Hyper-Transport, Intel Quick Path Interconnect). Обращение к «локальной» области памяти выполняется непосредственно, обращение к «удалённой» области памяти является более трудоёмким, поскольку выполняется через транзитные узлы.

В многопоточных программах при выполнении критических секций на выделенных процессорных ядрах существенную роль играет латентность при обращении RCL-сервера к участкам памяти внутри критических секций. Поэтому в таких программах важно выделять память в области памяти, локальной узлу, на котором функционирует сервер RCL. Это позволяет сократить время обращений к памяти и минимизировать время выполнения многопоточной программы.

В настоящей статье исследуется влияние политики выделения памяти в NUMA-системах на эффективность метода делегирования выполнения критических секций. Предложен алгоритм RCLAwareMalloc, позволяющий минимизировать время выполнения параллельных программ за счет оптими-

зации выделения памяти в NUMA-системах. Суть алгоритма заключается в следующем. При выполнении многопоточной программы память выделяется на том NUMA-узле, на котором функционирует RCL-сервер. Основные шаги алгоритма RCLAwareMalloc:

1. При выполнении функции выделения памяти проверяется, инициализированы ли в программе RCL-блокировки (запущены ли RCL-серверы).

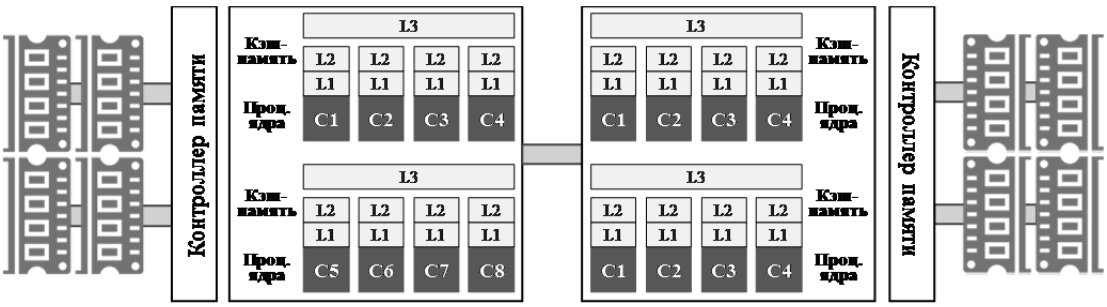


Рис. 4. Функциональная схема многопроцессорной NUMA-системы

2. Если в программе не инициализированы RCL-блокировки, то используется функция DefaultMalloc выделения памяти по умолчанию и указатель на выделенный сегмент памяти добавляется в список AllocList.

3. Если в программе инициализированы RCL-блокировки, то проверяется, выполняются ли все RCL-серверы на одном NUMA-узле.

4. Если все RCL-серверы выполняются на одном NUMA-узле, то устанавливается политика выделения памяти на данном NUMA-узле. В противном случае используется функция DefaultMalloc.

5. Если после очередного запуска RCL-блокировки обнаружено, что не все RCL-серверы выполняются на одном узле, то осуществляется привязка выделения памяти к NUMA-узлу.

6. При инициализации первой RCL-блокировки проверяется, пустой ли список AllocList. Если список не пустой, то выполняется миграция блоков выделенной памяти на NUMA-узел, на котором функционирует RCL-сервер.

3. Результаты экспериментов

Эксперименты проводились на узле вычислительного кластера Oak Центра параллельных вычислительных технологий Федерального государственного бюджетного образовательного учреждения высшего образования «Сибирский государственный университет телекоммуникаций и информатики». Узел укомплектован двумя четырёхъядерными процессорами Intel Xeon E5420, 24 Гб памяти. Соотношение скорости доступа к локальному и удалённому сегментам памяти 2:1.

Моделирование алгоритмов выполнялось на основе синтетического теста. Тест реализует итеративный доступ к элементам массива длины $b = 5 \times 10^8$. Использовалось три шаблона доступа к элементам:

– случайный: на каждой итерации выбирается случайный элемент массива;

– последовательный: на каждой новой итерации выбирается элемент, следующий за предыдущим;

– с интервалом: на каждой новой итерации выбирается элемент, расположенный на расстоянии $s = 1000$ элементов от предыдущего.

Число p параллельных потоков варьировалось от 1 до 8 (количество процессорных ядер на вычислительном узле). Каждый поток выполнял $n = 10^8 / p$ операций инкремента соответствующего элемента тестового массива. Индекс элемента массива определялся шаблоном доступа к массиву.

Проводилось сравнение двух алгоритмов выделения памяти:

1. Выделение памяти по умолчанию (DefaultMalloc).

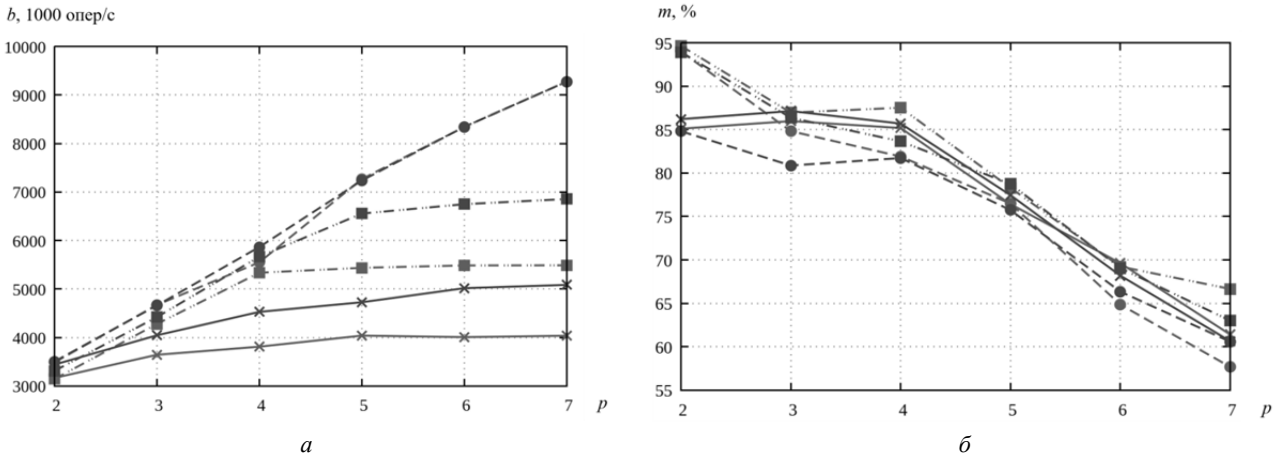
2. Выделение памяти с учётом расположения RCL-сервера (RCLAwareMalloc).

Выполнялась привязка потоков к процессорным ядрам вычислительного узла:

– поток, выполняющий выделение памяти: NUMA-узел 0, процессорное ядро 0;

– RCL-сервер: NUMA-узел 1, процессорное ядро 0;
– тестовый поток 0: NUMA-узел 0, процессорное ядро 0;
– тестовый поток 1: NUMA-узел 0, процессорное ядро 1;
– тестовый поток 2: NUMA-узел 0, процессорное ядро 2;
– тестовый поток 3: NUMA-узел 0, процессорное ядро 3;
– тестовый поток 4: NUMA-узел 1, процессорное ядро 1;
– тестовый поток 5: NUMA-узел 1, процессорное ядро 2;
– тестовый поток 6: NUMA-узел 1, процессорное ядро 3.

В качестве показателей эффективности использовалась пропускная способность $b = n / t$ (t – время выполнения экспериментов) и m – процент промахов по кэшу (cache misses).



а *б*

Рис. 5. Эффективность выполнения тестовой параллельной программы: *а* – пропускная способность, *б* – промахи по кэшу.
– DefaultMalloc, случайный доступ; – DefaultMalloc, последовательный доступ;
– DefaultMalloc, доступ с интервалом; – RCLAwareMalloc, случайный доступ;
– RCLAwareMalloc, последовательный доступ; – RCLAwareMalloc, доступ с интервалом

Эксперименты показали (рис. 5), что алгоритм RCLAwareMalloc выделения памяти, учитывающий размещение RCL-сервера в NUMA-системе, позволяет повысить (по сравнению с алгоритмом DefaultMalloc) пропускную способность критической секции за счёт сокращения времени выполнения обращений RCL-сервера к памяти. Предложенный подход наиболее эффективен при случайном доступе к элементам массива и в случае доступа к элементам массива с интервалом. Эффективность RCLAwareMalloc увеличивается с ростом числа потоков.

Заключение

Разработан алгоритм выделения памяти в вычислительных системах с неоднородным доступом к памяти при выполнении критических секций на выделенных процессорных ядрах (RCL). Алгоритм позволяет минимизировать время выполнения критических секций по сравнению с выделением памяти по умолчанию за счёт минимизации количества обращений к удалённым NUMA-сегментам памяти. Алгоритм реализован в виде библиотеки и может быть использован с целью минимизации существующих многопоточных программ.

ЛИТЕРАТУРА

1. Хорошевский В.Г. Распределённые вычислительные системы с программируемой структурой // Вестник СибГУТИ. 2010. № 2 (10). С. 3−41.
2. Herlihy M., Shavit N. The Art of Multiprocessor Programming, Revised Reprint. Elsevier, 2012. 528 с.
3. Herlihy M., Moss J. E.B. Transactional memory: Architectural support for lock-free data structures // Proceedings of the 20th annual international symposium on computer architecture ACM. ACM, 1993. V. 21. No. 2. P. 289–300.

4. Mellor-Crummey J.M., Scott M.L. Algorithms for scalable synchronization on shared-memory multiprocessors // ACM Transactions on Computer Systems (TOCS). 1991. V. 9. No. 1. P. 21–65.
5. Hendler D. et al. Flat combining and the synchronization-parallelism tradeoff // Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures. ACM, 2010. P. 355–364.
6. Fatourou P., Kallimanis N.D. Revisiting the combining synchronization technique // ACM SIGPLAN Notices. ACM, 2012. V. 47. No. 8. P. 257–266.
7. Oyama Y., Taura K., Yonezawa A. Executing parallel programs with synchronization bottlenecks efficiently // Proceedings of the International Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications, PDSIA '99. 1999. P. 1–24.
8. Suleman M.A. et al. Accelerating critical section execution with asymmetric multi-core architectures // ACM SIGARCH Computer Architecture News. ACM, 2009. V. 37. No. 1. P. 253–264.
9. Metreveli Z., Zeldovich N., Kaashoek M.F. Cphash: A cache-partitioned hash table // ACM SIGPLAN Notices. ACM, 2012. V. 47, No. 8. P. 319–320.
10. Calciu I., Gottschlich J.E., Herlihy M. Using elimination and delegation to implement a scalable NUMA-friendly stack // Proc. Usenix Workshop on Hot Topics in Parallelism (HotPar). 2013. P. 1–7.
11. 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.

Пазников Алексей Александрович, канд. техн. наук. E-mail: apaznikov@gmail.com
Институт физики полупроводников им. А.В. Ржанова СО РАН (г. Новосибирск)

Поступила в редакцию 20 марта 2016 г.

Paznikov Alexey A. (Rzhanov Institute of Semiconductor Physics Siberian Branch of Russian Academy of Sciences. Novosibirsk, Russian Federation).

Optimization method of remote core locking.

Keywords: RCL; multithreading programming; critical sections; scalability.

DOI: 10.17223/19988605/38/8

The number of processor cores in shared memory computer systems is increasing nowadays. Therefore, the need of scalable parallel programs rises up. These programs must be efficient on large number of parallel threads.

Among the perspective scalable approaches of multithreading programming, one can emphasize lock-free algorithms and data structures and software transactional memory. The main drawbacks of lock-free algorithms and data structures are their restrained area of applying and essential complexity of parallel programs development. Besides the throughput of those structures is comparable with lock-based structures. Current implementations of software transactional memory do not provide high performance of multithreading programs and have not common in real applications yet.

Classical approach of lock-based critical sections in multithreading programs remains the most prevalent. Locks are easier than lock-free algorithms and data structures and ensures high level of productivity. Besides that, the most of present software is lock-based. Thus, development of scalable locks is substantial now.

Locks scalability depends on access contention while accessing of parallel threads to share memory and space cache locality. Access contention takes place while simultaneous access of multiple thread to one critical section, protected by one synchronization object. This leads to cache become inefficiency. Cache locality is significant when the critical section contains the access to shared data, which was accessed previously on the other core. This circumstance leads to cache misses and severe critical section execution time increasing.

The most efficient approaches of scalable locks implementation are CAS spinlocks, MCS-locks, Flat combining CC-Synch, DSM-Synch, Oyama lock. However, these methods cannot guarantee the minimum of critical section execution.

The critical section execution time includes time of lock ownership transfer and critical section execution time. In this work, we consider the method of critical section execution on remote processor cores (Remote Core Locking, RCL). RCL minimizes the execution time of legacy multithreading programs by reduction critical section execution critical path. This method replaces critical sections in legacy programs by procedures remote calls. All critical sections protected by one lock are performed by server thread launched on remote core. Therefore, by execution of the critical section by the particular thread minimizes the overheads of lock ownership transfer.

In the current implementation RCL, they do not consider the non-uniform access to memory in NUMA-systems. These systems are the compositions multicore processors connected with local area of global memory. Time of critical section execution in RCL depends on RCL-server latency while accessing to memory blocks within critical section. Thus, in these programs memory should be allocated on NUMA node with the RCL-server. This reduces time of memory accesses and minimizes of programs execution time.

In this work, we propose the algorithm minimizing the memory access time in NUMA systems while RCL-programs execution. The algorithm optimizes the memory allocation on NUMA-systems. The essential of the algorithms is allocating on the NUMA-node, on which RCL-server is executing. The results of experiments show that the memory allocation algorithm increases the throughput of critical sections execution by reducing the time of RCL-server access to memory.

REFERENCES

1. Khoroshevsky, V.G. (2010) Distributed programmable structure computer systems. *Vestnik SibGUTI*, 2(10). pp. 3−41. (In Russian).
2. Herlihy, M. & Shavit, N. (2012) *The Art of Multiprocessor Programming*. Revised Reprint. Elsevier.
3. Herlihy, M. & Moss, J.E.B. (1993) Transactional memory: Architectural support for lock-free data structures. *Proceedings of the 20th annual international symposium on computer architecture ACM*. 21(2). pp. 289–300. DOI: 10.1145/173682.165164
4. Mellor-Crummey, J.M. & Scott, M.L. (1991) Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems (TOCS)*. 9(1). pp. 21–65. DOI: 10.1145/103727.103729

5. Hendler, D. et al. (2010) Flat combining and the synchronization-parallelism tradeoff. *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*. ACM. pp. 355–364. DOI: 10.1145/1810479.1810540
6. Fatourou, P. & Kallimanis, N.D. (2012) Revisiting the combining synchronization technique. *ACM SIGPLAN Notices*. ACM. 47(8). pp. 257–266. DOI: 10.1145/2370036.2145849
7. Oyama, Y., Taura, K. & Yonezawa, A. (1999) Executing parallel programs with synchronization bottlenecks efficiently. *Proceedings of the International Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications, PDSIA '99*. pp. 1–24.
8. Suleman, M.A. et al. (2009) Accelerating critical section execution with asymmetric multi-core architectures. *ACM SIGARCH Computer Architecture News*. ACM. 37(1). pp. 253–264. DOI: 10.1145/2528521.1508274
9. Metreveli, Z., Zeldovich, N. & Kaashoek, M.F. (2012) Cphash: A cache-partitioned hash table. *ACM SIGPLAN Notices*. ACM. 47(8). pp. 319–320. DOI: 10.1145/2370036.2145874
10. Calciu, I., Gottschlich, J.E. & Herlihy, M. (2013) Using elimination and delegation to implement a scalable NUMA-friendly stack. *Proc. Usenix Workshop on Hot Topics in Parallelism (HotPar)*. pp. 1–7.
11. Lozi, J.P. et al. (2012) Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications. *USENIX Annual Technical Conference*. pp. 65–76.