Тема работы:

Разработка средств анализа эффективности выполнения атомарных операций в многоядерных вычислительных системах с общей памятью

Состав коллектива:

1. Пазников Алексей Александрович, к.т.н., доцент. СПбГЭТУ «ЛЭТИ», руководитель

2. Гончаренко Евгений Андреевич, студент. СПбГЭТУ «ЛЭТИ», исполнитель

Информация о гранте:

РФФИ, проект № 19-07-00784 «Разработка методов, алгоритмов и программного обеспечения масштабируемой синхронизации для многопроцессорных вычислительных систем», руководитель – Пазников А.А., 2019-2021

Научное содержание работы:

1. **Введение, постановка задачи:**

К многоядерным вычислительным системам (ВС) с общей памятью относятся как автономные десктопные и серверные системы, так и вычислительные узлы в составе распределенных ВС (кластерные ВС, системы с массовым параллелизмом). Такие системы могут включать десятки и сотни процессорных ядер. Например, вычислительный узел суперкомпьютера Summit, занимающего первое место в рейтинге TOP500 (более 2 млн. процессорных ядер, 4608 узлов), включает два 24-ядерных универсальных процессора IBM Power9 и шесть графических ускорителей NVIDIA Tesla V100 (640 ядер). Sunway TaihuLight (более 10 млн. процессорных ядер, третье место в рейтинге TOP500) укомплектован 40 960 процессорами Sunway SW26010, включающими 260 ядер. При реализации кэш-памяти в таких ВС применяются различные политики включения (инклюзивный и эксклюзивный кэш), протоколы когерентности (MESI, MOESI, MESIF, MOWESI, MERSI, Dragon) и топологии межпроцессорных шин в NUMA-системах (Intel QPI, AMD HyperTransport).

Одной из наиболее значимых задач при разработке параллельных программ является создание эффективных средств синхронизации параллельных потоков. Основными методами синхронизации являются средства блокировки потоков (locks, mutexes), неблокируемые (non-blocking, lock-free) потокобезопасные (concurrent, thread-safe) структуры данных и транзакционная память. Атомарные операции (atomic operations, atomics) применяются при реализации всех методов синхронизации. Операция называется атомарной, если она завершается в один неделимый шаг относительно других потоков. Ни один из потоков не может наблюдать операцию «частично завершенной». Если два или более потоков выполняют операции над разделяемой переменной и хотя бы один выполняет операцию записи, то потоки должны использовать атомарные операции для избежания гонки данных (data race).

К наиболее распространенным относятся атомарные операции «сравнение с обменом» (compare-and-swap, CAS), «выборка и сложение» (fetch-and-add, FAA), «обмен» (swap, SWP), «чтение» (load) и «запись» (store). Операция load(m, r) реализует чтение значения переменной

из ячейки памяти m в регистр r. store(m, r) – запись значения переменной в ячейку памяти m из регистра r. FAA(m, r1, r2) увеличивает (уменьшает) на размещенное в регистре r1 значение в ячейке памяти m и возвращает предыдущее значение в регистр r2. SWP(m, r) – обмен значений между ячейкой памяти m и регистром r. CAS(m, r1, r2) реализует сравнение значения ячейки памяти m с регистром r1; если значения равны, то в m устанавливается r2. При разработке параллельных программ необходимо учитывать влияние на эффективность атомарных операций таких аспектов, как выполнение механизма когерентности кэш-памяти, размер буфера, числа потоков, удаленности данных от ядра.

2. Современное состояние проблемы:

Несмотря на широкое применение, эффективность атомарных операций не проанализирована в достаточной степени. Например, считается, что CAS медленнее FAA [1] и ее семантика позволяет ввести понятие «бесполезная работа» (wasted work) [2, 3], поскольку неудачные попытки сравнения данных в памяти и в регистре приводят к дополнительной нагрузке на ядро. В работе [3] анализируется производительность атомарных операций на графических процессорах, однако сегодня остро стоит задача оценки эффективности атомарных операций на универсальных процессорах. В работе [4] рассматривается эффективность выполнения операций CAS, FAA, SWP с целью анализа влияния динамических параметров программы на время выполнения атомарных операций. Результаты исследований демонстрируют недокументированные свойства тестируемых систем и оценки времени выполнения атомарных операций. Однако тесты проводились при фиксированной частоте процессорных ядер, использовании больших страниц памяти и с выключенной предварительной выборкой данных, что искажает результаты моделирования для реальных программ. Кроме того, не исследовались атомарные операции load и store.

1. A. Morrison and Y. Afek. Fast Concurrent Queues for x86 Processors // PPoPP'13. 2013. P. 103–112.
2. T. L. Harris. A Pragmatic Implementation of Non-blocking Linked-Lists // DISC'01. 2001. P. 300–314.
3. Elteir M., Lin H., Feng W. Performance characterization and optimization of atomic operations on amd gpus // IEEE International Conference on Cluster Computing. 2011. P. 234-243.
4. Schweizer H., Besta M., Hoefler T. Evaluating the cost of atomic operations on modern architectures // 2015 International Conference on Parallel Architecture and Compilation (PACT). 2015. P. 445-456.
5. Dey S., Nair M.S. Design and Implementation of a Simple Cache Simulator in Java to Investigate MESI and MOESI Coherency Protocols //International Journal of Computer Applications. 2014. V. 87. N. 11. P. 6-13.
6. Molka D., Hackenberg D., Schone R., Muller M.S. Memory performance and cache coherency effects on an intel nehalem multiprocessor system // 18th International Conference on Parallel Architectures and Compilation Techniques. 2009. P. 261-270.

3. Подробное описание работы, включая используемые алгоритмы:

В качестве показателей эффективности используются латентность l выполнения атомарной операции и пропускная способность b. Пропускная способность $b = n / t$, где n – число выполненных операций за время t при реализации последовательного доступа к ячейкам буфера.

Для точного измерения времени применялся набор инструкция RDTSCP. Для предотвращения переупорядочивания инструкций применялись полные барьеры памяти.

Исследуется влияние состояния кэш-линии в рамках протоколов когерентности (M, E, S, O, I, F) на эффективность выполнения атомарных операций. Определим основные состояния кэш-линий. M (Modified) – кэш-линия модифицирована и содержит актуальные данные. O (Owned) – кэш-линия содержит актуальные данные и является их единственным владельцем. E (Exclusive) – кэш-линия содержит актуальные данные, которые соответствуют состоянию памяти. S (Shared) – кэш-линия содержит актуальные данные, при этом другие процессорные ядра имеют копии этих данных в разделяемом состоянии. F (Forwarded) – кэш-линия содержит наиболее свежие, корректные данные, при этом другие ядра в системе могут иметь копии данных в разделяемом состоянии. I (Invalid) – кэш-линия содержит некорректные данных.

Разработана тестовая программа. Перед началом выполнения организуется буфер q в виде целочисленного массива размера s = 128 Мб. Данные размещаются в кэш-памяти, кэш-линии переводятся в заданное состояние протокола когерентности. Подкачка страниц не применялась. Данные не выравнивались. Перевод кэш-линий в состояние M происходит при записи произвольных значений в ячейки буфера. Для перевода в состояние E выполняется запись в ячейки буфера, затем – инструкция clflush для перевода кэш-линий в состояние I, после этого – чтение ячеек буфера. Переход в состояние S происходит после чтения ячеек буфера из кэш-памяти в состоянии E одного ядра в кэш-память другого ядра. Перевод кэш-линий в состояние O достигается путем чтения из кэш-памяти в состоянии M одного ядра в кэш-память другого ядра, при этом кэш-линии ядра из M переключаются в состояние O.

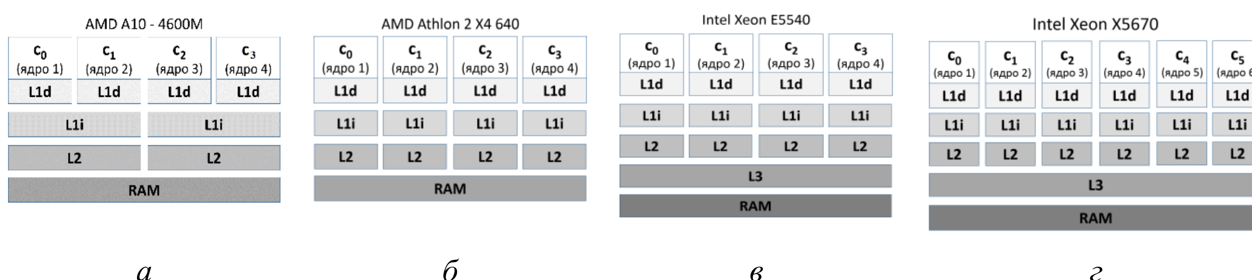

Рис. 1. Исследуемые микроархитектуры (*а* – Piledriver, *б* – K10, *в* – Nehalem-EP, *г* – Westmere-EP)
Fig. 1. Target microarchitectures (*а* – Piledriver, *б* – K10, *в* – Nehalem-EP,
г – Westmere-EP)

Для операции CAS выполняется два эксперимента: для успешной и неудачной операции. Неудачным считается CAS, при выполнении которого $m \neq r_1$. Заведомо неудачное выполнение

CAS достигается путем сравнения адреса указателя с данными по этому указателю. В успешном CAS $m = r_1$.

Эксперименты проводились на процессорах AMD Athlon II X4 640 (микроархитектура K10), AMD A10-4600M (микроархитектура Piledriver) (протокол когерентности MOESI [10]), Intel Xeon X5670 (микроархитектура Westmere-EP) и Intel Xeon E5540 (микроархитектура Nehalem-EP) (MESIF [11]) (рис. 1). Размер кэш-линии 64 байта. В качестве компилятора использовался компилятор GCC 4.8.5, операционная система SUSE Linux Enterprise Server 12 SP3 (версия ядра 4.4.180).

Опишем основные шаги алгоритма измерения времени выполнения атомарных операций для состояния E (рис. 2*a*). Вспомогательная функция DoOper реализует выполнение заданной атомарной операции для всех элементов буфера (строки 1-4). Основной алгоритм выполняется до тех пор, пока размер буфера не достигнет максимального (строка 5). Задействованный диапазон тестового массива *testSizeBuffer* в экспериментах варьируется от 6Кб (минимальный размер кэша первого уровня) до 128 Мб. Каждый эксперимент запускается *nruns* раз (строка 6). Выполняется запись произвольных данных в ячейки буфера для перевода кэш-линий ядра c_0 в состояние M (для остальных ядер состояние кэш-линий переводится в I) (строка 7). Далее выполняется инвалидация кэш-линий (строка 12) с последующим чтением, что переводит кэш-линий ядра c_0 в состояние E (строка 9). На следующем шаге над каждой переменной выполняется атомарная операция (строка 11). Вычисляется время выполнения операции и суммарное время выполнения (строка 13). Рассчитываются латентность (строка 15), пропускная способность (строка 16) и увеличивается текущий размер буфера на $step = L_x / 8$ (где L_x – размер кэш-памяти уровня $x = 1, 2, 3$)) (строка 17).

Алгоритмы выполнения атомарных операций для состояний M и I аналогичны описанному, при этом для M строки 8, 9 не выполняется, для I не выполняются строки 7, 9.

1	**Function** DoOper(oper)
2	**for** $i = 0$ **to** d **do**
3	*buffer*.oper()
4	**end for**
5	**while** $d \le testSizeBuffer$ **do**
6	**for** $j = 0$ **to** *nruns* **do**
7	DoOper(STORE(1))
8	CLFLUSH(*buffer*, *d*);
9	DoOper(LOAD)
10	*start* = GET_TIME();
11	DoOper(AtomicOp(*buffer*[*i*]))
12	*end* = GET_TIME();
13	$sumTime = sumTime + (end - start)$;
14	**end for**
15	$latency = sumTime / nruns / d$;
16	$bandwidth = (d / sumTime / nruns) \times 10^9$
17	$/ 2^{20}$;
18	$d = d + step$;

1	Первый поток на c_0:
2	**while** $d \le testSizeBuffer$ **do**
3	**for** $j = 0$ **to** *nruns* **do**
4	DoOper(STORE(1))
5	CLFLUSH(*buffer*,*d*);
6	DoOper(LOAD)
7	Второй поток на c_2:
8	DoOper(LOAD)
9	Первый поток на c_0:
10	*start* = GET_TIME();
11	DoOper(AtomicOp(*buffer*[*i*]))
12	*end* = GET_TIME();
13	$sumTime = sumTime + (end - start)$;
14	**end for**
15	$latency = sumTime / nruns / d$;
16	*bandwidth*
17	$= (d / sumTime / nruns) \times 10^9 / 2^{20}$;

	end while

а

18	$d = d +$ step; **end while**

б

Рис. 2. Алгоритм измерения латентности и пропускной способности выполнения атомарных операций SWP/FAA/CAS/Load/Store. (*а* – Exclusive c_0, *б* – Shared c_0)
Fig. 2. Algorithm for measuring the latency and throughput of atomic operations SWP/FAA/CAS/Load/Store and throughput (*а* – Exclusive c_0, *б* – Shared c_0)

Основные шаги алгоритма измерения времени выполнения атомарных операций для c_0 в состоянии кэш-линий S (рис. 2*б*). Алгоритм выполняется в двух потоках, привязанных к ядрам c_0 и c_2. Синхронизация реализуется посредством атомарных флагов. В первом потоке на ядре c_0 выполняется запись произвольных данных в буфер, перевод кэш-линий в состояние M, при этом для остальных ядер состояние изменяется на I (строка 4). Далее выполняется очистка кэш-линий (сброс в I) (строка 5) и чтение данных для изменения состояния кэш-линий c_0 в состояние E (строка 6). Во втором потоке (ядро c_2) выполняется чтение данных, что изменяет состояние кэш-линий ядер c_0 и c_2 на S (строка 8). Далее первый поток на ядре c_0 реализует атомарную операцию над элементами тестового буфера (строка 11). Вычисляется время выполнения операции и показатели эффективности (строки 15-17).

На рис. 3, 4 приведены результаты экспериментов для операции SWP.

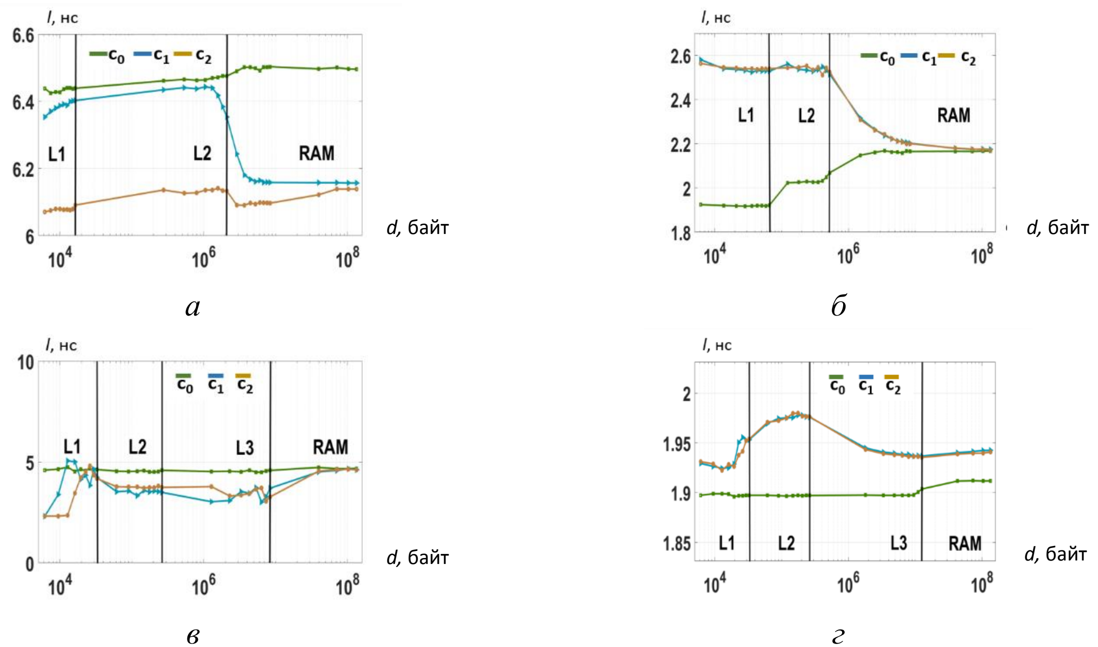

Рис. 3. Латентность выполнения атомарной операции SWP для кэш-линий
в состоянии Exclusive (*а* – Piledriver, *б* – K10, *в* – Nehalem-EP, *г* – Westmere-EP)

Fig. 3. Latency an atomic operation SWP for cache lines in the Exclusive state
(*а* – Piledriver, *б* – K10, *в* – Nehalem-EP, *г* – Westmere-EP)

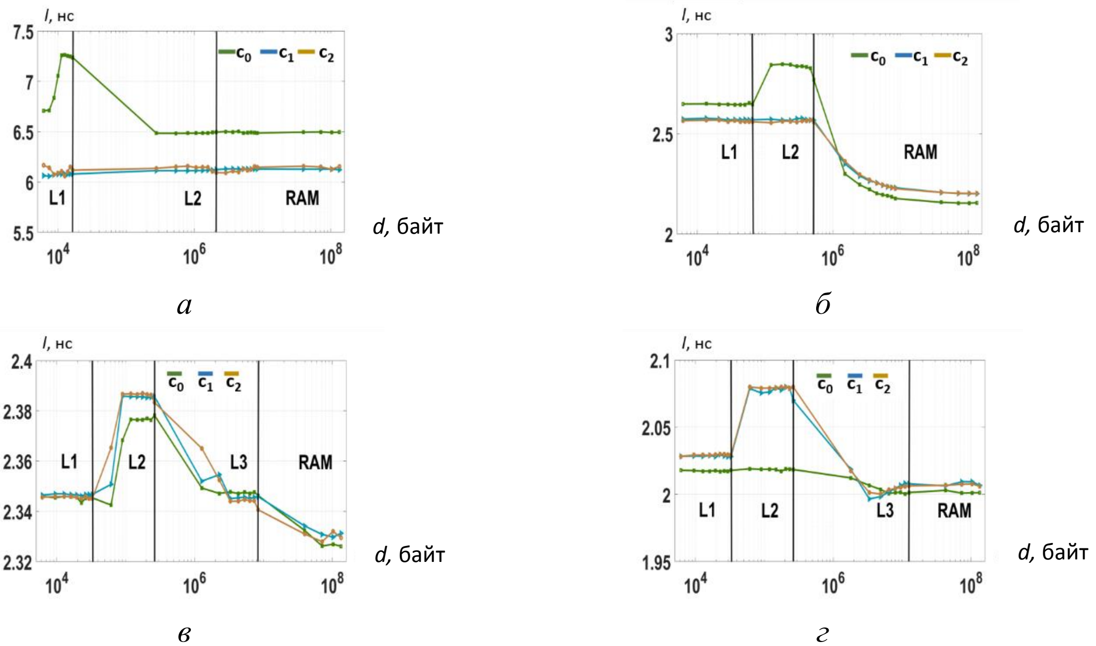

Рис. 4. Латентность выполнения атомарной операции SWP для кэш-линий
в состоянии Shared (*а* – Piledriver, *б* – K10, *в* – Nehalem-EP*, г* – Westmere-EP)
Fig. 4. Latency an atomic operation SWP for cache lines in the Shared state
(*а* – Piledriver, *б* – K10, *в* – Nehalem-EP, *г* – Westmere-EP)

Опишем основные шаги алгоритма измерения времени выполнения атомарных операций для ядер c_1 и c_2 в состоянии S кэш-линий ядра c_0 (рис. 5*а*). Алгоритм выполняется в трех потоках, привязанных к ядрам c_0, c_1, c_2. Первый поток (ядро c_0) реализует запись в буфер, что переводит состояние кэш-линий c_0 в M (для остальных ядер состояние изменяется на I) (строка 4). Далее выполняется сброс состояния в I (строка 5) и чтение данных, что переводит кэш-линии c_0 в состояние E (строка 6). Во втором потоке (ядро c_1) выполняется чтение данных (состояние кэш-линии c_0 и c_1 изменяется на S) (строка 8). На следующем шаге в третьем потоке (ядро c_2) над каждым элементом буфера выполняется атомарная операция (строка 11). Вычисляется показатели эффективности (строки 15-17). Для измерения латентности выполнения атомарных операций на ядре c_1 используется аналогичный алгоритм, в котором шаги для ядер c_1 и c_2 меняются местами.

```
1   Первый поток на c0:
2   while d ≤ testSizeBuffer do
3      for j = 0 to nruns do
4         DoOper(STORE(1))
5         CLFLUSH(buffer,d);
6         DoOper(LOAD)
7   Второй поток на c1:
8         DoOper(LOAD)
9   Третий поток на c2:
10        start = GET_TIME();
11        DoOper(AtomicOp(buffer[i]))
12        end = GET_TIME();
13        sumTime = sumTime + (end – start);
14     end for
15     latency = sumTime / nruns / d;
16     bandwidth
17  = (d / sumTime / nruns) × 10^9 / 2^20;
18     d = d + step;
    end while
```

а

```
1   Первый поток на c0:
2   while d ≤ testSizeBuffer do
3      for j = 0 to nruns do
4         DoOper(STORE(1))
5   Второй поток на c2:
6         DoOper(LOAD)
7   Первый поток на c0:
8         start = GET_TIME();
9         DoOper(AtomicOp(buffer[i]))
10        end = GET_TIME();
11        sumTime = sumTime + (end – start);
12     end for
13     latency = sumTime / nruns / d;
14     bandwidth
15  = (d / sumTime / nruns) × 10^9 / 2^20;
16     d = d + step;
    end while
```

б

Рис. 5. Алгоритм измерения латентности и пропускной способности выполнения атомарных операций. (*а* – кэш-линии ядра c_0 в состоянии Shared, измерения для ядер c_1 и c_2, *б* – состояние Owned)
Fig. 5. Algorithm for measuring the latency and throughput of atomic operations
(*a* – Shared c_0, measurements for c_1 and c_2, *б* – Owned state)

Опишем основные шаги алгоритма измерения времени выполнения атомарных операций в состоянии кэш-линий O на ядре c_0 (рис. 5*б*). Алгоритм выполняется в двух потоках, привязанных к ядрам c_0 и c_2. В первом поток (ядре c_0) произвольные данные записываются в буфер для изменения состояния кэш-линий ядра c_0 в M (для остальных ядер состояние изменяется I) (строка 4). Во втором потоке (ядро c_2) выполняется чтение данных, что изменяет состояние кэш-линий локального ядра c_0 на O, а ядра c_2 – на S (строка 8). На следующем шаге в первом потоке (ядро c_0) над каждой переменной буфера выполняется атомарная операция (строка 9). Вычисляется время выполнения операции и показатели эффективности (строки 13-15).

На рис. 6 приведены результаты для операции SWP, состояние O.

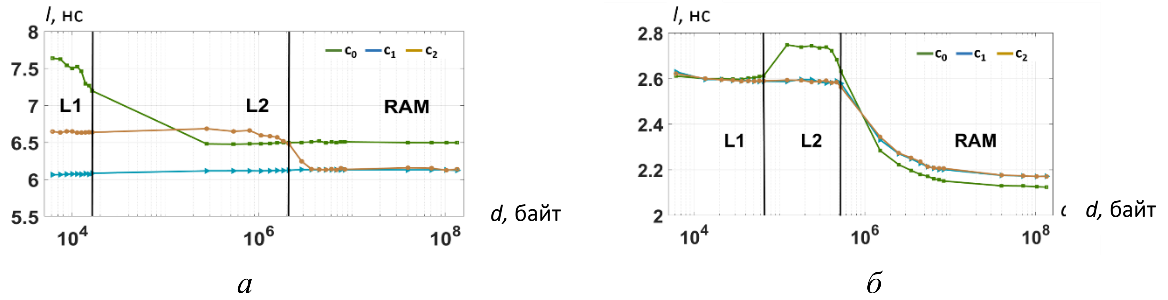

Рис. 6. Латентность выполнения атомарной операции SWP кэш-линий
в состоянии Owned (*а* – архитектура Piledriver, *б* – архитектура K10)
Fig. 6. Latency of atomic operation SWP for cache lines in the Owned state (*a* – Piledriver, *б* – K10)

В табл. 1 представлены субоптимальные параметры выполнения атомарных операций на процессоре с микроархитектурой Westmere-EP. Аналогичные результаты получены и для других процессоров. Так, например, для процессоров микроархитектур K10, Nehalem-EP, Westmere-EP наименьшая латентность выполнения операции load получена в состоянии S на ядре c_0 (от 1,14 до 1,81 нс), в то время как для процессора с микроархитектурой Piledriver данная операция имеет наименьшую латентность на ядрах c_0 и c_2 (от 1,8 до 2,4 нс). В табл. 2 даны отношения максимальной и минимальной пропускной способности.

Т а б л и ц а 1

Субоптимальные параметры для архитектуры Westmere-EP

Операция	Субоптимальные параметры			Общий уровень памяти	*l*, нс	*b*, Мб/с
	Состояние кэш-линий	Размер буфера	Ядро			
Load	Exclusive	L2	c_0, c_1, c_2	L3	1,1 – 1,15	823 – 866
Store	Modified	RAM	c_0, c_1, c_2	L3	4,1 – 4,21	226 – 230
FAA	Invalid	L1	c_0, c_1, c_2	L3	1,89 – 1,91	490 – 499
SWP	Invalid, Modified	RAM	c_0, c_1, c_2	L3	1,93	493
unCAS	Exclusive	L1, L2	c_0, c_1, c_2	L3	2,55 – 2,59	367 – 372
CAS	Invalid	L2, L3	c_0, c_1, c_2	L3	4,9 – 19,3	49 – 194

Таблица 2

Отношения максимальной и минимальной пропускной способности при выполнении атомарных операций

Операция	Piledriver b_{max} / b_{min}	K10 b_{max} / b_{min}	Nehalem-EP b_{max} / b_{min}	Westmere-EP b_{max} / b_{min}
Load	1,4	1,1	2,1	2
FAA	1,1	1,2	2,2	1,1
SWP	1,1	1,2	2,1	1,1
unCAS	1,1	1,2	2,4	2,0
Store	3,9	1,1	2,1	1,1
CAS	1,1	1,6	6,1	7,2

4. Полученные результаты:

Состояние кэш-линий M. Значения показателей эффективности для атомарных операций SWP, FAA, unCAS на ядре c_0 архитектуры Piledriver незначительно варьируются при изменении размера буфера, в то время как для ядер c_1 и c_2 увеличение размера буфера более L2 приводит к сокращению латентности до минимального значения для обоих ядер. Для K10 при размере буфера, превышающем L2, латентность операций SWP, FAA, unCAS различается в пределах погрешности для всех ядер. При выполнении load, SWP, CAS для Nehalem-EP латентность сопоставима для всех ядер при размере буфера более L3. При выполнении операций load, SWP для Westmere-EP наблюдается аналогичная ситуация, для остальных операций (store, CAS, FAA, unCAS) латентность является наименьшей для ядра c_0 и не зависит от размера буфера.

Состояние кэш-линий E. При выполнении операций SWP, FAA, unCAS на архитектуре Piledriver и размере буфера более L2 для ядер c_1 и c_2 латентность сопоставима, при этом на ядре c_0 получены максимальные значения. Для архитектуры K10 латентность операций SWP, FAA, unCAS аналогична для всех ядер при размере буфера, превышающем L2. Для Nehalem-EP латентность SWP, load сопоставима на всех ядрах при размере буфера более L3. Для Westmere-EP минимальная латентность для load, SWP, FAA, CAS получена на ядре c_0, для ядер c_1 и c_2 при размере буфера более L3 латентность различается незначительно.

Состояние кэш-линий S. При выполнении SWP, FAA на архитектуре Piledriver при размере буфера L1 латентность максимальна на ядре c_0. Для K10 при размере буфера L2 наибольшая латентность получена на ядре c_0 для SWP, FAA, store. Для Nehalem-EP и Westmere-EP при размере буфера L2 латентность выполнения SWP, FAA максимальна на ядрах c_1, c_2.

Состояние кэш-линий I. При выполнении SWP, FAA, unCAS на архитектуре Piledriver размер буфера не значительно влияет на время выполнения для всех ядер, наименьшая латентность получена на ядрах c_1 и c_2. При выполнении load, SWP на K10 при размере буфера не более L2 латентность выполнения минимальна для ядра c_0. Для Nehalem-EP латентность SWP, load сопоставима на всех ядрах при размере буфера более L3. При выполнении load, SWP, FAA, CAS на Westmere-EP наименьшая латентность получена на ядре c_0.

Состояние кэш-линий O. При выполнении атомарных операций SWP, FAA на архитектуре Piledriver при размере буфера L1 наибольшая латентность получена на ядре c_0. Для ядра c_1 размер буфера не влияет на время выполнения операций. Наибольшая латентность операций SWP, FAA, store на архитектуре K10 получена при размере буфера L2 на ядре c_0.

Наибольшей латентностью, по сравнению с другими операциями, характеризуется операция CAS (успешный). Операция load выполняется с наименьшей латентностью. Например, для процессоров K10, Westmere-EP минимальная латентность CAS получена для состояния S на ядре c_0 и составляет от 18 до 24 нс; для процессора Piledriver CAS имеет наименьшую латентность на ядрах c_0 и c_1 (от 42 до 44 нс); на процессоре Nehalem-EP в состоянии S CAS выполняется с наименьшей латентностью на ядре c_0 (от 22 до 46 нс), при этом наибольшая латентность (46 нс) получена при размере буфера L2. Для архитектуры Piledriver минимальная латентность операции load (1,76 нс) превышает минимальную латентность операции CAS (12,39 нс) в 7 раз. Для архитектуры K10 минимальная латентность load (1,72 нс) превышает минимальную латентность CAS (22,38 нс) в 13 раз. Для Nehalem-EP отношение минимальной латентности load (1,3 нс) к минимальной латентности CAS (9,86 нс) – 7,5. Для архитектуры Westmere-EP отношение минимальной латентности load (1,1 нс) к минимальной латентности CAS (4,9 нс) – 4,5. Сравнивая микроархитектуры, отметим, что в среднем наименьшая латентность получена на процессоре Westmere-EP (MESIF), наибольшая – на Piledriver (MOESI).

5. Эффект от использования кластера в достижении целей работы:

Экспериментальные исследования проводились на вычислительном узле кластерной вычислительной системы, укомплетованном двумя 6-ядерными процессорами Intel Xeon X5670 с тактовой частотой 2932 МГц и 24 Гб ОЗУ, и узле, укомплектованном двумя 4-ядерными процессорами Intel Xeon E5540 с тактовой частотой 2530 МГц. Применение данных систем с общей памятью позволило исследовать влияние архитектурных свойств многоядерных систем на эффективность выполнения атомарных операций.

Перечень публикаций, содержащих результаты работы:

1. Goncharenko E.A., Paznikov A.A. Analysis of the efficiency of atomic operations in multi-core shared-memory computer systems // Tomsk State University Journal of Control and Computer Science, 2020. – N 51. – pp. 102–110

2. Goncharenko E., Paznikov A. Evaluation of the Impact of Cache Coherence Protocol and Data Locality on the Efficiency of Atomic Operations on Multicore Processors // Proceedings of the 11th Majorov International Conference on Software Engineering and Computer Systems (MICSECS 2019), 2019. – pp. 1-12.

3. Goncharenko E. A., Paznikov A. A., Tabakov A. V. Performance Modeling of Atomic Operations in Control Systems Based on Multicore Computer Systems // 2019 III International Conference on Control in Technical Systems (CTS), 2019. – pp. 140-143.

4. Goncharenko E.A., Paznikov A.A., Tabakov A.V. Evaluating the performance of atomic operations on modern multicore systems // Journal of Physics: Conference Series, 2019 – V. 1399. – N. 3 – pp. 033107. DOI: 10.1088/1742-6596/1399/3/033107 Scopus: 2-s2.0-85077466469
5. Goncharenko E. A., Paznikov A. A. Analysis of the Influence of Cache Coherence Mechanism Performance on Atomic Operations in Multicore Computing Systems // 2019 XXII International Conference on Soft Computing and Measurements (SCM). – IEEE, 2019. – P. 111-114. DOI: 10.1109/SCM.2019.8903720 Scopus: 2-s2.0-85075919838
6. Свид. 2019660469 Российская Федерация. Свидетельство о государственной регистрации программы для ЭВМ. Программа для анализа эффективности выполнения атомарных операций в многоядерных вычислительных системах с общей памятью / Пазников А.А., Гончаренко Е.А. Заявитель и патентообладатель Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И. Ульянова (Ленина)» (СПбГЭТУ «ЛЭТИ»). Заявл. 23.06.2019, опубл. 06.08.2019.