Алгоритмы для доработки в libcds

Требования к реализации

К следующему семестру более явно написать формулировки пп. 4 и 9

1. Pull Request должен быть сливаем с текущей master-веткой репозитория
2. Отсутствие закомментированных кусков кода
3. Код написан в стиле и стандарте кодирования библиотеки
4. Написан набор модульных и стресс-тестов (если структура данных стандартная, то тесты используются сущетвующие с регистрацией своей структуры данных)
5. Поддерживается интерфейс сбора статистики по контейнеру, если таковой есть в библиотеке для реализуемого типа контейнеров
6. Сборка и прогон тестов с активированными Address и Undefined Behavior Sanitizers не выдаёт никаких ошибок (“-fsanitize=address,undefined”)
7. Тесты прогнаны с активированным Thread Sanitizer и по каждому предупреждению есть уверенность, что это не ошибка
8. Тесты проанализированы сторонним средством поиска ошибок (например, Intel Parallel Studio) и также по каждому предупреждению есть уверенность, что это не ошибка
9. После выполнения вышеуказанных пунктов ветка добавляется в CI и на всех поддерживаемых платформах все тесты проходят успешно

Нереализованные

[2011] Bar-Nissan, Hendler, Suissa. A dynamic elimination-combining stack algorithm.pdf

*Сложность:*3

Развитие подхода Flat Combining. Алгоритм DECStack интересен тем, что со стороны работы с центральным стеком он является non-blocking и поэтому требует применения safe memory reclamation (SMR) типа Hazard Pointer (HP), а со стороны фазы collide — блокируемый. Требуется решить следующие задачи:

1. Избавиться от предположения, что потоки пронумерованы от 1 до N. У нас есть только thread ID. В этой связи надо как-то изменить структуру массива location. Видимо, вместо него нужно использовать thread-local storage (TLS), тогда в collision хранить указатели на записи в TLS.
2. Распределения динамической памяти (allocation) под структуры multiOp быть не должно. Лучшим вариантом будет, если multiOp создаются на стеке, но в этом случае могут быть серьезные проблемы с временем жизни, - обращение к разрушенным данным и пр.
3. Центральный стек — алгоритм Трейбера. Будет здорово, если в реализации класс стека будет вынесен в опции (traits) класса DECStack.

[2005] Purcell, Harris Non-blocking Hashtables with open addressing.pdf

*Сложность:*5+

Еще одна разновидность open-addressing hash table. Алгоритм не описывает расширение таблицы, предлагается использовать метод из предыдущей работы ([2003] Gao, Groote, Hesselink Efficient almost wait-free parallel accessible dynamic Hashtables.pdf).

[2012] Crain,Gramoli,Raynal A Contention-Friendly, Non-Blocking Skip List.pdf

*Сложность:*6

Ещё одна реализация skip-list. Алгоритм интересен тем, что на нулевом уровне узлы связаны в double-linked list. Авторы не говорят, применим ли HP к их алгоритму, вместо этого они предлагают использовать reference counting или кратко описанный ими epoch-based подход. Требуется понять, можно ли применить HP и/или RCU.

[2011] Brown,Helga Non-blocking k-ary Search Trees.pdf

*Сложность:*7

Развитие подхода [2010] Ellen,Fatourou,Ruppert,vanBreugel Non-blocking Binary Search Trees.pdf на k-арные деревья. Сделать без helping'a, оценить, во что выливается реализация helping'a и стоит ли его вообще делать.

[2012] Afek,Kaplan,Korenfeld,Morrison,Tarjan CBTree - A Practical Concurrent Self-adusting Search Tree.pdf

*Сложность:*7+

[2012] Korenfeld CBTree - A Practical Concurrent Self-Adjusting Search Tree.pdf Разновидность самобалансирующегося дерева, построенного с использованием техники hand-over-hand locking. Проанализировать, какой SMR подходит (HP, RCU, оба), либо же алгоритм не требует никакой SMR схемы.

[2011] Prokopec,Bagwell,Odersky Cache-Aware Lock-Free Concurrent Hash Tries.pdf

*Сложность:*8

[2011] Prokopec,Bagwell,Odersky Lock-Free Resizable Concurrent Tries.pdf [2011] Prokopec,Bronson,Bagwell,Odersky Concurrent Tries with Efficient Non-Blocking Snapshots.pdf Реализация конкурентного trie. No comments.

Требующие доработки

[2007] Herlihy, Lev, Luchangco, Shavit A Provably Correct Scalable Concurrent Skip List .pdf

> *Сложность:*4+
> Pull request

Skip-list, основанный на fine-grained locks (блокировка на уровне элементов — именно для такого класса алгоритмов интересно реализовать различные mutex'ы). Интересная и относительно простая реализация skip-list. Node сделать аналогично тому, как сделано в libcds SkipListSet. Можно подумать над оптимизацией для «невысоких» узлов: для узла высотой 1 не выделяется никакой памяти под башню (у него нет башни), 50% узлов в skip-list имеют высоту 1; хотелось бы для узлов высотой ⇐ 4 не распределять память под башни, а использовать какой-то pre-allocated storage. Тем самым для 50 + 25 + 12,5 + 6,25 = 93.75% узлов не будет аллокаций, что хорошо должно сказаться на масштабируемости. Можно попробовать все узлы распределять высотой 4:

```
struct node {
    …
    uint32_t height;
    node * tower; // только для узлов высотой > 4
    node * next[4];

    node * next( uint32_t level )
        {
        if ( level < 4 ) return next[level];
        return tower[level – 4];
    }
};
```

[2013] Henzinger,Payer,Sezgin Replacing competition with cooperation to achieve scalable lock-free FIFO queues.pdf

> *Сложность:*4
> Pull request

Мне этот алгоритм очень хочется попробовать. Единственный минус — авторы не показали, как использовать технику HP, только намекнули в нескольких предложениях. Требуется адаптировать алгоритм под Hazard Pointer и/или связаться с авторами, запросив пояснений.

[1995] Valois Lock-Free Linked Lists Using Compare-and_Swap.pdf

> *Сложность:*3+
> Pull request

Эта работа, несмотря на свою «старость», интересна тем, что позволяет сделать полноценные итераторы по lock-free структуре. На основе описанного алгоритма списка можно построить

hash-map (MichaelHashSet/Map, SplitListSet/Map – эти структуры в libcds как параметр принимают реализацию списка) или даже SkipList с возможностью безопасной итерации, что очень ценно. Не забывать, что в данном алгоритме есть ошибка — см. [1995] Michael, Scott Correction of a Memory Management Method for Lock-Free Data Structures.pdf.

[2008] Herlihy,Shavit,Tzafrir Hopscotch Hashing.pdf

> *Сложность:*5+
> Pull request

Интересный алгоритм open-addressing hash table. Похож на Cuckoo hashing, реализация которого есть в libcds.

[2010] Ellen,Fatourou,Ruppert,vanBreugel Non-blocking Binary Search Trees.pdf

> *Сложность:*2+
> Pull request

В libcds уже есть HP- и RCU-based реализации (без helping'а), требуется сделать append-only версию (без GC – cds::gc::nogc), то есть версию без возможности удаления элементов. Попробовать реализовать helping (будет хорошо, если helping будет включаться отдельным флагом в traits).

A.Williams “C++ Concurrency in Action” - lock-free очередь на shared_ptr

> *Сложность:*2
> Pull request

Алгоритм и его подробный анализ есть в книге. Требуется создать очередь на C++11. Оценить, можно ли создать intrusive-версию.

A.Williams “C++ Concurrency in Action” - lock-free стек на shared_ptr

> *Сложность:*1+
> Pull request

Алгоритм и его подробный анализ есть в книге. Требуется создать стек на C++11, используя оба алгоритма, описанных в книге: на shared_ptr и split reference counting. Оценить, можно ли создать intrusive-версию.

[2014] Dodds,Haas,Kirsch Fast Concurrent Data-Structures Through Explicit Timestamping.pdf

> *Сложность:*4
>
> Pull request

Перспективный алгоритм построения lock-free стека, очереди, deque. Должен обеспечивать очень хорошую производительность для push-операций, так как push производится в локальный для потока storage. Псевдокод основан на tagged pointers, требуется применить какой-либо другой SMR – Hazard Pointers, reference counting.

[2003] Gao, Groote, Hesselink Efficient almost wait-free parallel accessible dynamic Hashtables.pdf

> *Сложность:*5
>
> Pull request

Интересный алгоритм open-addressing hash table. Следует сделать intrusive-вариант: храним указатели на данные, считаем, что распределять память под данные не нужно. Алгоритм описан достаточно хорошо, но непривычно. Замечания:

1. Понять, где следует применять атомарные операции. В псевдокоде это отмечено скобками < >:
   - < a := b > ⇒ atomic load, atomic store
   - < if a == 0 then a := b > ⇒ a.CAS(0, b)
2. Псевдокод описан в предположении, что имеется P потоков (процессов), работающих с hash set. Попробовать избавиться от P. Если не получится — P должен быть параметром конструктора. Для thread-local data использовать TLS. Реализация должна учитывать, что P может быть достигнуто — в этом случае либо выбрасывать исключение (плохо), либо каким-то образом дожидаться освобождения слота (лучше).
3. К данным, хранимым в hash table, следует применять Hazard Pointer. Для управления внутренними структурами алгоритм использует разновидность refCount.

Развитие: если можно выделить алгоритм расширения hash-таблицы в отдельную программную сущность (класс), то его в принципе можно применить к любым алгоритмам open-addressing, например, к cuckoo hashing.

[2010] Gidenstam,Sundell,Tsigas Cache-Aware Lock-free Queues for Multiple Producers-Consumers and Weak Memory Consistency.pdf

> *Сложность:*4
>
> Pull request

Также: А также: [2010] Gidenstam,Sundell,Tsigas Efficient Lock-Free Queues that Mind the Cache.pdf. Алгоритм интересен тем, что память выделяется под блок элементов, как в std::deque, что может значительно увеличить производительность stl-like очереди. Оценить, можно ли реализовать этот алгоритм на Hazard Pointer. Алгоритм существенно полагается на разработанную авторами SMR схему — помесь HP и reference counting – в части управления списком блоков. Попробовать использовать для блоков shared_ptr + HP. Создать intrusive и stl-like версии.

[2014] Henzinger, Kirsch, Payer, Sezgin, Sokolova Quantitative Relaxation of Concurrent Data Structures.pdf

> *Сложность:*3
>
> Pull request

Сегментированный стек. В статье приводится псевдокод для tagged pointers (указатель + версия). Следует реализовать его на Hazard Pointer, intrusive и stl-like версию. Размер сегмента — степень двойки, задается в конструкторе. Оценить, возможно ли сделать оптимизацию — не удалять сегмент, а вести некоторый free-list нескольких удаленных сегментов.

Реализованные

[2003] Michael CAS-Based Lock-Free Algorithm for Shared Deques.pdf

> *Сложность:*3
>
> Pull request

Реализовать описанный в статье алгоритм bounded deque. SMR здесь не требуется, так как deque основана на массиве.