Lock-free Data Structure w/ HPX

Weile Wei
Ph.D. Student
Louisiana State University
wwei9@lsu.edu

Acknowledgement

This project is funded through:

Google

Google Summer of Code

Thanks mentors and collaborators!

John Biddiscombe
Computational Scientist

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Mikael Simberg
Software Engineer

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Hartmut Kaiser
Research Scientist

LSU Center for Computation & Technology

Max Khizhinsky
Evgeny Kalishenko
Alexander Gaev

LibCDS team

Why lock-free?

concurrent counter increment

```
// ################# Lock Version #################
int counter = 0;
std::mutex counter_mutex;

void increment_with_lock()
{
    std::lock_guard<std::mutex> _(counter_mutex);
    ++counter;
}

// ############## Lock-free Version ###############

std::atomic<int> counter(0);

void increment_lockfree()
{
    ++counter;
}
```

- Lock version:
  - no thread can make progress until the lock-holding thread unlocks the mutex.
- Lock-free version:
  - all threads can make progress.

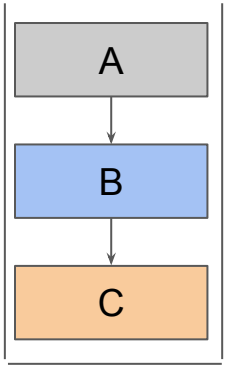
More problems with lock based approaches

- Deadlock
- Priority inversion
- Kill, Crash, Pre-emption
- ...

ABA problem: A is not A

Break lock-free stack in ABA problem

- Suppose there are 2 threads accessing the stack
- The stack is C ← B ← A (head is A). ABC are addresses



stack

Thread 1

1. read A from head
2. read A → next
3.
4.
5.
6.
7. CAS(head, A, B) succeeds
8. Head is B which is already freed!

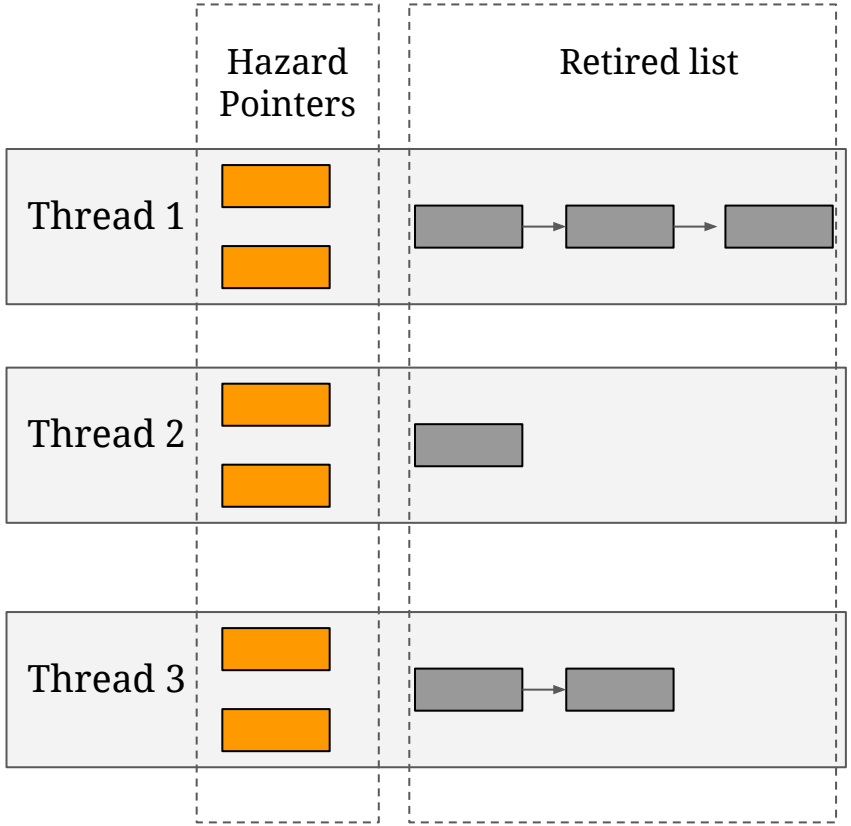
Thread 2

1.
2.
3. pop() A. Stack C ← B
4. pop() B. Stack C
5. push(data), memory rescues address A
6. Stack becomes C ← A
7.
8.

Hazard Pointers

A methodology for memory reclamation for lock-free dynamic objects.

A hazard pointer is a single-writer multi-reader pointer that can be owned by at most one thread at any time. Only the owner of the hazard pointer can set its value, while any number of threads may read its value.



LibCDS: A C++ library of Concurrent Data

This library is a collection of lock-free (i.e. Hazard Pointers) and lock-based fine-grained algorithms of data structures like maps, queues, list etc.

- Written in C++ 11
- Run across multi-platforms
- GitHub: https://github.com/khizmax/libcds

LibCDS Overview

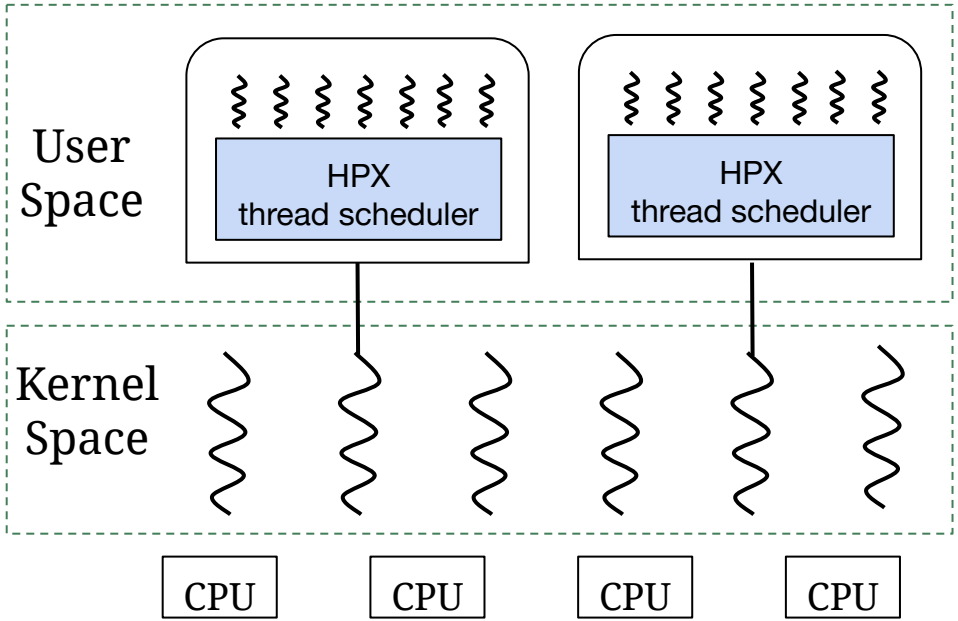
```cpp
#include <cds/init.h>       // for cds::Initialize and cds::Terminate
#include <cds/gc/hp.h>      // for cds::HP (Hazard Pointer) SMR
int main(int argc, char** argv)
{
    // Initialize libcds
    cds::Initialize();
    {
        // Initialize Hazard Pointer singleton
        cds::gc::HP hpGC;

        // If main thread uses lock-free containers
        // the main thread should be attached to libcds infrastructure
        cds::threading::Manager::attachThread();

        // Now you can use HP-based containers in the main thread
        //...
    }

    // Terminate libcds
    cds::Terminate();
}
```

init thread manager: each thread has its ThreadData

allocates Hazard Pointer SMR (Safe Memory Reclamation) thread-private thread_data

LibCDS uses thread_local for storing thread-private data, that is bound to kernel-level thread.

HPX user-level threading



To note, HPX threads can migrate from one kernel thread to another. We make them to be HPX user-level thread-private data.

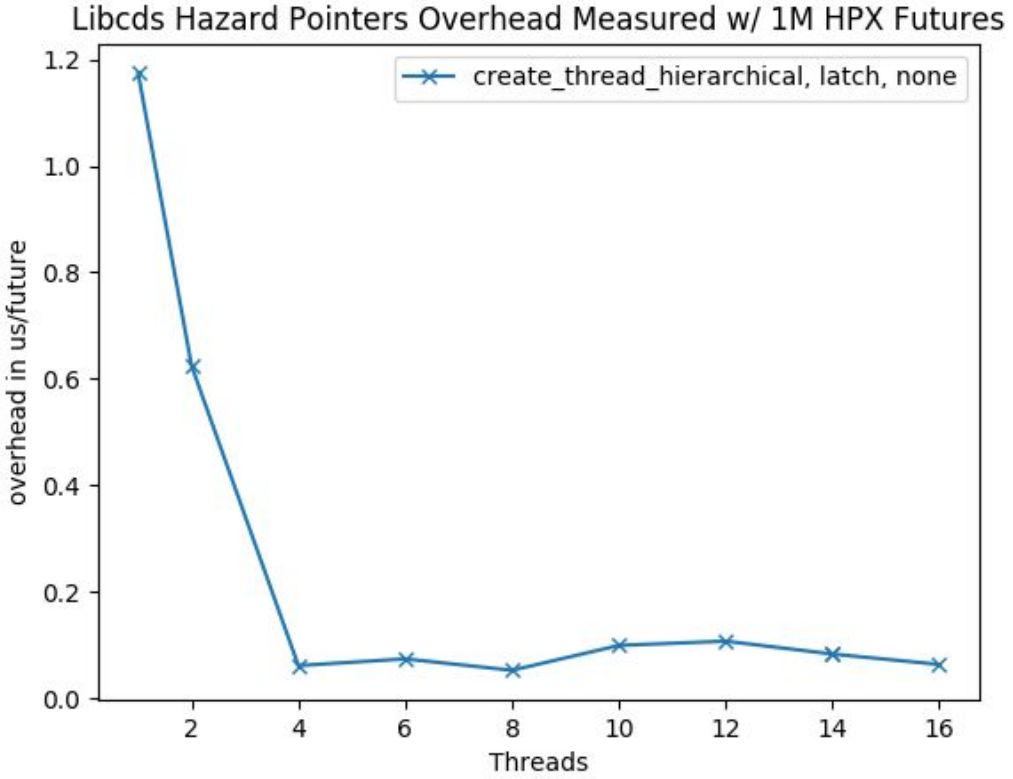
HP_overhead.cpp | hpx_manager.h

```
43          /// Checks whether current thread is attached to \p libcds feature or not.
44          static bool isThreadAttached()
45          {
46              std::array<size_t, 3> hpx_thread_data = hpx::threads::get_libcds_data(hpx::threads::get_self_id());
47              ThreadData * pData = reinterpret_cast<ThreadData*> (hpx_thread_data[thread_manager_index]);
48              return pData != nullptr;
49          }
50
51          /// This method must be called in beginning of thread execution
52          static void attachThread()
53          {
54              std::array<size_t, 3> hpx_thread_data = hpx::threads::get_libcds_data(hpx::threads::get_self_id());
55              ThreadData * pData = reinterpret_cast<ThreadData*> (hpx_thread_data[thread_manager_index]);
56              if(pData == nullptr)
57              {
58                  pData = new ThreadData;
59                  hpx_thread_data[thread_manager_index] = reinterpret_cast<std::size_t>(pData);
60                  hpx::threads::set_libcds_data(hpx::threads::get_self_id(), hpx_thread_data);
61              }
62              assert( pData );
63              pData->init();
64          }
65
66          /// This method must be called in end of thread execution
67          static void detachThread()
68          {
69              std::array<size_t, 3> hpx_thread_data = hpx::threads::get_libcds_data(hpx::threads::get_self_id());
70              ThreadData * pData = reinterpret_cast<ThreadData*> (hpx_thread_data[thread_manager_index]);
71              assert( pData );
72
73              if ( pData->fini())
74              {
75                  hpx_thread_data = hpx::threads::get_libcds_data(hpx::threads::get_self_id());
76                  hpx_thread_data[thread_manager_index] = reinterpret_cast<std::size_t>(nullptr);
77                  hpx::threads::set_libcds_data(hpx::threads::get_self_id(), hpx_thread_data);
78              }
79
80          }
```

cds › threading › hpxthread › Manager › attachThread

Hazard Pointer Overhead

```
/////////////// measure libcds overhead ////////////////////
void null_function(bool uselibcds) noexcept
{
#ifdef CDS_THREADING_HPX
    if (uselibcds) cds::threading::Manager::attachThread();
#endif
    // dummy compuation
#ifdef CDS_THREADING_HPX
    if (uselibcds) cds::threading::Manager::detachThread();
#endif
}
```



Reference

- Michael, M. M. (2004). Hazard pointers: Safe memory reclamation for lock-free objects. IEEE Transactions on Parallel and Distributed Systems, 15(6), 491-504.

- P0233r0. Maged M. Michael, Michael Wong. Hazard Pointers, Safe Resource Reclamation for Optimistic Concurrency. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0233r0.pdf

- Work in progress
  - https://github.com/STEllAR-GROUP/hpx/tree/libcds
  - https://github.com/weilewei/libcds/tree/hpx-thread

Lock-free Data Structure w/ HPX

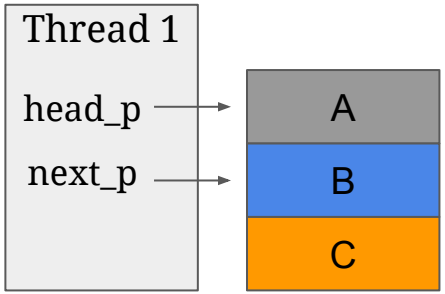
Weile Wei
Ph.D. Student
Louisiana State University
wwei9@lsu.edu

Backup slides

Break stack in ABA problem

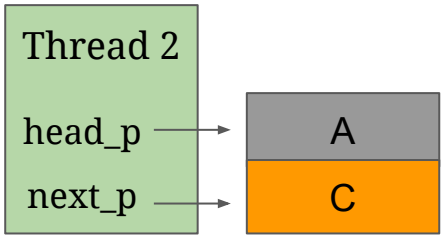
1

Thread 1 is attempting to pop() A.



Before thread 1 finishes pop(), thread 2 pops A, B, then push A back to stack

2



3

B becomes the new head but B was already deleted. Therefore, the program has undefined behaviour.