CYBERUS TECHNOLOGY

Brandenburg University of Technology Cottbus - Senftenberg

21. November 2019

CACHE-LINE TRANSACTIONS

Building Blocks for Persistent Kernel Data Structures Enabled by AspectC++

Marcel Köppen, Jana Traue, Christoph Borchert, Jörg Nolte, Olaf Spinczyk

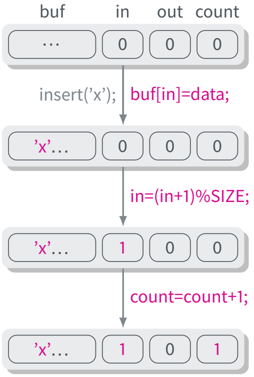
jana.traue@cyberus-technology.de

Example: Bounded Buffer

```
class BoundedBuffer {
  static constexpr int SIZE = 28;
  char buf[SIZE];
  uint8_t in, out, count;

public:
BoundedBuffer():
  in(0), out(0), count(0) {}

void addByte(char data) {
  if ((in + 1) % SIZE == out) {
    return;
  }
  buf[in] = data;
  in = (in + 1) % SIZE;
  count = count + 1;
}
```



Bounded Buffer with Intel PMDK

```
class PMDKBoundedBuffer {
  static constexpr int SIZE = 28;
  typedef pmem::obj::p<char> pchar;
  pmem::persistent_ptr<pchar[]> buf;
  pmem::obj::p<uint8_t> in, out, count;

public:
PMDKBoundedBuffer():
  in(0), out(0), count(0) {
  buf = pmem::obj::make_persistent<pchar[]>(
    SIZE);
}

void addByte(char data) {
  auto pop = pmem::obj::pool_by_vptr(this);
  pmem::obj::transaction::exec_tx(pop, [&] {
    if ((in + 1) % SIZE == out) { return; }
    buf[in] = data;
    in = (in + 1) % SIZE;
    count = count + 1;
  });
}
```

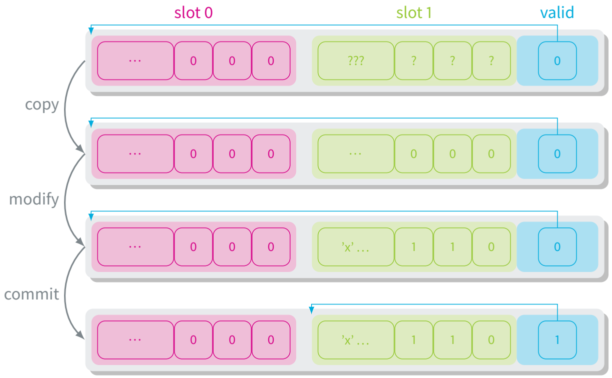
PMDK

- Persistent Memory Development Kit
- transactions based on undo logging
- complex code changes
- significant runtime overhead
- …because of flushes

Reducing the Number of Flushes

Cache-Line Transactions

- limit whole transaction to a single cache line
- stores to a single cache line are not re-ordered



Mission Accomplished

Cache-Line Transactions

- need only a single flush
- have implicit recovery

…but

- modify data structure layout
- follow scheme: copy, modify, commit
- compiler barriers to prevent reordering
- `CLWB` and `SFENCE` to flush to PMEM

Mission Accomplished

Cache-Line Transactions

- need only a single flush
- have implicit recovery

…but

- modify data structure layout
- follow scheme: copy, modify, commit
- compiler barriers to prevent reordering
- CLWB and SFENCE to flush to PMEM

Bounded Buffer with AspectC++

```
class [[NVM::transactional]] alignas(64)
BoundedBuffer {
    static constexpr int SIZE = 28;
    char buf[SIZE];
    uint8_t in, out, count;

public:

BoundedBuffer():
    in(0), out(0), count(0) {}

void addByte(char data) {
    if ((in + 1) % SIZE == out) {
        return;
    }
    buf[in] = data;
    in = (in + 1) % SIZE;
    count = count + 1;
}

// ...
```

AspectC++

- adds copy and valid bit
- wraps non-const methods in transactions
- redirects member access to correct slot

AspectC++ Internals

```
// all non-const member functions should be transactions
advice call(transaction()) && !within(transaction()) : around() {
    tjp->target()->log();
    tjp->proceed();
    tjp->target()->commit();
}

void log() {
    memcpy(getActive(this), getValid(this), sizeof(Copy));
}

void commit() {
    barrier();
    _index ^= 32;
    clwb(this);
    sfence();
}
```

Impact on Source Code

Original

```
class
BoundedBuffer {
  static constexpr int SIZE = 28;
  char buf[SIZE];
  uint8_t in, out, count;

public:
  BoundedBuffer() : in(0), out(0), count(0) {}

  void addByte(char data) {
    if ((in + 1) % SIZE == out) { return; }
    buf[in] = data;
    in = (in + 1) % SIZE;
    count = count + 1;
  }

  char getByte() {
    if (out == in) { return 0; }
    char result = buf[out];
    out = (out + 1) % SIZE;
    count = count - 1;
    return result;
  }
};
```

PMDK

```
class PMDKBoundedBuffer {
  static constexpr int SIZE = 28;
  typedef pmem::obj::p<char> pchar;

  pmem::obj::persistent_ptr<pchar[]> buf;
  pmem::obj::p<uint8_t> in, out, count;

public:
  PMDKBoundedBuffer() : in(0), out(0), count(0) {
    buf = pmem::obj::make_persistent<pchar[]>(SIZE);
  }

  ~PMDKBoundedBuffer() {
    pmem::obj::delete_persistent<pchar[]>(buf, SIZE);
  }

  void addByte(char data) {
    auto pop = pmem::obj::pool_by_vptr(this);
    pmem::obj::transaction::exec_tx(pop, [&] {
      if ((in + 1) % SIZE == out) { return; }
      buf[in] = data;
      in = (in + 1) % SIZE;
      count = count + 1;
    });
  }

  char getByte() {
    char result = 0;
    auto pop = pmem::obj::pool_by_vptr(this);
    pmem::obj::transaction::exec_tx(pop, [&] {
      if (out == in) { return; }
      result = buf[out];
      out = (out + 1) % SIZE;
      count = count - 1;
    });
    return result;
  }
};
```

CLTX + AspectC++

```
class [[NVM::transactional]] alignas(64)
BoundedBuffer {
  static constexpr int SIZE = 28;
  char buf[SIZE];
  uint8_t in, out, count;

public:
  BoundedBuffer() : in(0), out(0), count(0) {}

  void addByte(char data) {
    if ((in + 1) % SIZE == out) { return; }
    buf[in] = data;
    in = (in + 1) % SIZE;
    count = count + 1;
  }

  char getByte() {
    if (out == in) { return 0; }
    char result = buf[out];
    out = (out + 1) % SIZE;
    count = count - 1;
    return result;
  }
};
```

Micro-Benchmarks

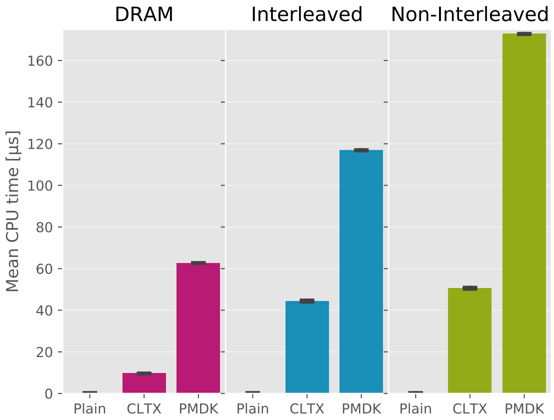
Bounded Buffer

- 29 bytes capacity
- no count
- fill with 29 entries, then drain them using
    - 58 individual transactions
    - 2 large transactions (fill, drain)

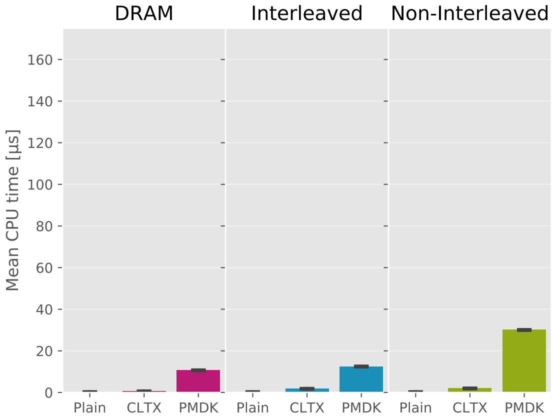
Hardware

- DDR4
- 6 Optane DC DIMMs interleaved
- 1 Optane DC DIMM (non-interleaved)

58 small transactions



2 large transactions



Conclusion

Cache-Line Transactions

- highly efficient
- …for small data structures on PMEM

AspectC++

- hides complexity
- provides convenient API for PMEM programming