Sprzętowa pamięć transakcyjna

Paweł Dziepak

Instytut Informatyki Uniwersytetu Wrocławskiego

21 stycznia 2015

Co chcemy osiągnąć?

```
bool reserve_memory(size_t amount)
{
    do {
        __transaction {
            if (free_memory < amout) {
                return false;
            }
            free_memory -= amount;
            reserved_memory += amount;
            return true;
        }
    } while (true);
}
```

Pamięć transakcyjna

Transakcja ma jeden z dwóch możliwych rezultatów:

- **commit** – rezultaty operacji w *transaction region* są obserwowalne jakby zostały wykonane w sposób atomowy
- **abort** – przywracany jest stan przed rozpoczęciem transakcji, żaden skutek uboczny nie jest widoczny

Przyczyną abortów są głównie konflikty z innymi procesorami lub wątkami i "powody zależne od implementacji".

Miło byłoby też móc zagnieżdżać transakcje.

Pamięć transakcyjna – konflikty

Odczyty wykonane podczas trasakcji trafiają do *read-set*, zapisy do *write-set*. Konflikt pojawia się gdy inny procesor:

- wykona zapis do adresu w *read-set*
- wykona zapis lub odczyt z adresu w *write-set*

Nowe możliwości

- algorytmy *non-blocking* dostają coś więcej niż *compare-and-swap* i *fetch-and-**
- teoretycy dostają w końcu *double CAS*
- koniec z problemem ABA na platformach bez LL/SC

Stare możliwości 2.0 – seqlocks

writer

```
seqlock.pre++;
// ... write something ...
seqlock.post++;
```

reader

```
unsigned counter;
do {
    counter = seqlock.post.load();
    // ... read something ...
} while (counter != seqlock.pre.load());
```

Pamięć transakcyjna a alternatywy

- blokady są problematyczne z wielu powodów (priority inversion, real time, deadlocks)
- duże blokady – złe, lock contention
- małe blokady – złe, narzut związany z operacjami atomowymi i bardziej skomplikowane algorytmy
- pamięć transakcyjna generalnie pomaga pisarzom
- wiele szybkich odczytów i rzadkie zapisy – RCU raczej niezagrożone

Lock elision

- dodanie *fast path* do sekcji krytycznych
- w pierwszej kolejności próbujemy wyoknać sekcję krytyczną jako transakcję
- jedynie jeśli transakcja się nie powiedzie przechodzimy do *slow path* czyli "po staremu" uzyskujemy blokadę
- nawet nieudane transakcje mogą być korzystne – prefetching
- wszystkie zalety (i wady) jakie mają adaptive mutex
- w przypadku spinlocków prawie same zalety ("prawie" z powodu SMT)
- szczególnie pomaga gdzie szansa konfliktów jest mała: np. hash table, duże drzewa

Pamięć transakcyjna ver. -1 – LL/SC

```
again:
    ldxr x1, [x0]
    add  x1, x1, 1
    stxr w2, x1, [x0]
    cbnz w2, again
```

Zapis `stxr` powiedzie się jedynie jeżeli żaden inny procesor nie zmodyfikuje pamięci odczytanej przez `ldxr`. Transakcji pilnują *global* i *local exclusive monitors*.

Restricted Transactional Memory – x86 – Intel

RTM wprowadza nowe instrukcje `xbegin`, `xend`, `xabort`, `xtest`:

- `xbegin` – rozpoczyna transakcję, jako operand przyjmuje adres pod który skoczyć przy aborcie, zwraca w `rax` kod pozwalajacy na zidentifykowanie przyczyny abortu
- `xend` – kończy transakcję
- `xabort` – abort
- `xtest` – sprawdza czy jesteśmy w transakcji, przydatne do walki z kodem, n.p. `assert(is_locked(some_lock))`

Restricted Transactional Memory

```
again:
    xbegin abort_handler
    addq $1, foo
    subq $1, bar
    xend
    jmp done
abort_handler:
    // retry?
    je again
    jmp slow_path
```

Transactional Memory – POWER8 – IBM

TM wprowadza nowe instrukcje m.in.: `tbegin`, `tend`, `tabort`, `tcheck`:

- `tbegin` – rozpoczyna transakcję, nie przyjmuje żadnego operandu
- kolejne 3 to Ctrl-C, Ctrl-V ze slajdu o RTM
- `tsuspend` – wstrzymuje transakcję
- `tresume` – wznawia transakcję
- `treclaim` – abort wstrzymanej transakcji, przywraca wszystkie rejestry oprócz program countera
- `trechkpt` – ustawienie stanu wstrzymanej transakcji tak aby `tresume` wykonało poprawny abort

Pamięć transakcyjna i lock elision – HLE – x86 – Intel

Hardware Lock Elision interfejs do pamięci transakcyjnej pozwalający na łatwe dodanie lock elision do istniejącego kodu. Całośc sprowadza się do dwóch prefiksów `xacquire` i `xrelease`.

`spinlock_acquire()`

```
xacquire xchgq SPINLOCK, $1
...
```

`spinlock_release()`

```
xrelease movq $0, SPINLOCK
```

Pamięć transakcyjna i lock elision – HLE

- jedna próba wykonania sekcji krytycznej jako transakcji
- rozsądna implementacja spinlocków używa `pause` która powoduje abort
- specjalny przypadek dla zmiennej `SPINLOCK`:
  - trafia do *read-set* zamiast *write-set*
  - mimo to wszystkie odczyty z wnętrza *transaction region* dają wartośc zapisaną w `spinlock_acquire()`
  - nawet przy sukcesie transakcji zapis nie jest wykonywany

Advanced Synchronization Facility – x86 – AMD

Jak na razie AMD ogranicza produkcję do plików PDF.

- standardowy zestaw: `speculate`, `commit`, `abort`
- tylko adresy zadeklarowane przez `lock mov` lub `lock prefetch` są "chronione"
- stan rejestrów i niechronionych obszarów pamięci nie jest odtwarzany przy aborcie

Implementacja

- wykrywanie konfliktów
  - wykorzystywane są protokoły *cache-coherence*
  - linie z *read-set* i *write-set* muszą być w L1
  - L1d w Haswellach ma asocjacyjność 8 – przy 9 dostępach do pamięci i braku szczęscia mamy abort (a jest jeszcze SMT)
  - ale abort może też wystąpić przy 0 dostępach do pamięci
- rollback
  - pamięć: unieważnianie *cache*
  - rejestry: *register remapping*

Transaction aborts

Nowy supervillain – *transaction aborts*. Możliwe przyczyny:

- konflikt
- `xabort`
- ograniczenia implementacji (brak miejsca w L1, za dużo zagnieżdżeń)
- przerwanie (nie na PowerPC) lub wyjątek
- niektóre instrukcje (np. `pause`, `cpuid`)
- dostęp do obszarów pamięci z wyłączonym cache lub write-back
- system call
- "nie wiadomo co, ale może drugim razem się uda"

Lock elision dla muteksów

- sugeruje się aby próbować wykonać transakcję kilka razy zanim przejdzie się do *slow path*
- ale, jeśli blokada jest zajęta to poczekajmy aż zostanie zwolniona zanim spróbujemy z transakcją
- nie należy próbować jednak zbyt wiele razy, to nie spinlock
- blokada musi być zawsze w *read-set* na wypadek gdyby inny wątek przeszedł do *slow path*
- uważajmy na zbieranie statystyk, możliwe źródła abortów

Debugowanie transakcji

...pozostaje głównie w sferze marzeń.

- breakpoint w transaction region powoduje abort
- single stepping? – tym bardziej nie
- debugger i tak musi rozumieć transakcje aby się nie pogubić w śledzeniu stanu
- można próbować coś osiągnąć z QEMU

Dziękuję za uwagę.