Instytut Informatyki
Uniwersytet Wrocławski

Wprowadzenie do STM

Marek Materzok

ZOSIA 2007

Wprowadzenie

- Były sobie komputery. Wykonywały programy instrukcja po instrukcji i wszyscy byli szczęśliwi...

Instytut Informatyki
Uniwersytet Wrocławski

Wprowadzenie

- Były sobie komputery. Wykonywały programy instrukcja po instrukcji i wszyscy byli szczęśliwi...
- ...dopóki nie zachcieli, aby komputery robiły kilka rzeczy naraz...

Wprowadzenie

- Były sobie komputery. Wykonywały programy instrukcja po instrukcji i wszyscy byli szczęśliwi...
- ...dopóki nie zachcieli, aby komputery robiły kilka rzeczy naraz...
- ...i oczywiście, jak to często bywa, problem rozwiązano w najbrzydszy możliwy sposób.

Instytut Informatyki
Uniwersytet Wrocławski

Programowanie wielowątkowe

- Wątek - niezależny tok wykonania, posiadający własne rejestry procesora, ale dzielący pamięć z innymi wątkami.
- Program wielowątkowy jest wykonywany [illegible] instrukcje jego wątków [illegible] w dowolny możliwy sposób.
- Przeplot następuje na poziomie [illegible], nie instrukcji języka - nawet `x := x + 1` może być niedeterministyczne!

Przeplot - ilustracja

Wątek A

A1	A2	A3	A4

Wątek B

B1	B2	B3	B4

Przykładowe przeploty

A1	B1	A2	B2	A3	B3	A4	B4

B1	B2	A1	B3	A2	A3	B4	A4

B1	B2	B3	B4	A1	A2	A3	A4

Ujarzmianie niedeterminizmu

- Spostrzeżenie: kod pracujący na danych lokalnych dla wątku może być dowolnie przeplatany, wystarczy, że do kodu pracującego na wspólnych danych będzie miał wstęp tylko jeden wątek naraz
- Rozwiązanie: niech sekcje krytyczne będą chronione blokadą, aktywną bądź pasywną; nadgorliwy wątek sobie poczeka.

Ujarzmianie niedeterminizmu

- Spostrzeżenie: kod pracujący na danych lokalnych dla wątku może być dowolnie przeplatany, wystarczy, że do kodu pracującego na wspólnych danych będzie miał wstęp tylko jeden wątek naraz
- Rozwiązanie: niech sekcje krytyczne będą chronione blokadą, aktywną bądź pasywną; nadgorliwy wątek sobie poczeka.
- I tak narodziło się **ZŁO**: programowanie współbieżne z blokadami.

Dlaczego **ZŁO**?

- Zbyt gruboziarniste blokady ograniczają stopień równoległości.
- Zbyt drobnoziarniste blokady mają duży narzut czasowy i są kłopotliwe dla programisty.
- Podatność na błędy: ***deadlock*** , ***livelock*** , zgubione instrukcje blokowania/odblokowania...
- Na koniec najistotniejsze: blokady naruszają strukturalność - programów z blokadami nie można swobodnie składać.

Przykład

```
void successor(List l, Element e);
void remove(List l, Element e);
Element insertAfter(List l, Element m, Element e);

void swapWithSucc(List l, Element e) {
    Element succ = successor(l, e);
    remove(l, e);
    // do tego czasu succ moze nie istnieć w l...
    insertAfter(succ, e);
}
```

Rozwiązanie 1: komunikaty

- Źródłem **ZŁA** jest dzielona pamięć. Porzucamy więc ją zupełnie!
- Współbieżne procesy posiadają wyłącznie lokalną pamięć, komunikują się wyłącznie poprzez wymianę komunikatów.
- Swoboda rozproszenia: procesy nie muszą działać na tej samej maszynie.
- Wyraźny związek z programowaniem obiektowym.
- Języki: Occam, Erlang.

Instytut Informatyki
Uniwersytet Wrocławski

Rozwiązanie 2: pamięć transakcyjna

- Idea zaczerpnięta z baz danych: operacje na dzielonej pamięci odbywają się w atomowych transakcjach.
- Program wykonuje się jakby transakcje były wykonywane oddzielnie, nie widząc zmian wykonywanych przez inne wątki.
- Typowa realizacja - optymistyczna, nieblokująca. Dziennik pozwala na cofnięcie nieudanej transakcji.

Instytut Informatyki
Uniwersytet Wrocławski

Cechy pamięci transakcyjnej

- Narzut czasowy związany z zarządzaniem transakcjami.
- Zakaz używania wejścia-wyjścia wewnątrz transakcji:

```
atomic {
    wystrzelRakietyJadrowe();
    ...cos jeszcze...
}
```

- A co, jeśli programista zapomni o bloku `atomic`?

Cechy pamięci transakcyjnej

- Wysoki stopień równoległości - transakcje działające na różnych danych nie przeszkadzają sobie wzajemnie.
- Liczna klasa błędów, w tym ~~deadlock~~, nie istnieje.
- Odzyskujemy strukturalność! Złożenie operacji poprawnych jest nadal operacją poprawną.

```
void swapWithSucc(List l, Element e) {
    atomic {
        Element succ = successor(l, e);
        remove(l, e);
        insertAfter(succ, e);
    }
}
```

Instytut Informatyki
Uniwersytet Wrocławski

Haskell: krótki wstęp do monad

- Monada - algebra sekwencyjnych operacji, podobna do półgrupy.

```
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a
```

Prawa (łączność, element neutralny):

$$(a \gg\!= b) \gg\!= c = a \gg\!= (\lambda x \rightarrow b\,x \gg\!= c)$$

$$a \gg\!= return = a$$

$$return\,x \gg\!= a = a\,x$$

- Operacje wejścia/wyjścia i imperatywne obliczenia tworzą monadę *IO*.

Monada STM

- Podstawy:

```
type STM a          -- typ zmiennych dzielonych
newTVar  :: a -> STM (TVar a)
readTVar  :: TVar a -> STM a
writeTVar :: TVar a -> a -> STM ()
```

- No dobrze, ale jak wykonać transakcję?

```
atomic :: STM a -> IO a
```

Co zyskaliśmy?

- Nie można grzebać w zmiennych dzielonych poza transakcjami.

```
foo :: a → IO ()
foo x = writeTVar x                    -- błąd typowania!
```

- Nie można wykonywać wejścia/wyjścia podczas transakcji.

```
foo :: a → IO ()
foo x = atomically (do
  x ← readTVar x
  writeTVar x (x + 1)
  putStrLn "Hello"
  )                                    -- błąd typowania!
```

Instytut Informatyki
Uniwersytet Wrocławski

Warunkowa synchronizacja

```
[illegible] :: [illegible] ()
```

Realizacja: transakcja jest wznawiana, jak tylko inny wątek zmodyfikuje którąś ze zmiennych przeczytanych przez transakcję.

```
[illegible] :: [illegible] → [illegible] → [illegible] ()
[illegible] = do
    [illegible] ← [illegible]
    if [illegible] − [illegible] < 0
        then [illegible]
        else [illegible] ([illegible] − [illegible])
```

Operacja wyboru

```
(<|>) :: STM a → STM a → STM a
```

Znaczenie: spróbuj wykonać pierwszą operację; jeśli wywoła `retry`, wycofaj i wykonaj drugą.

```
withdraw :: Int → [TVar Int] → STM ()
withdraw (k : t) =
  withdraw k `orElse` withdraw t
withdraw [] = retry
```

Własności algebraiczne

- Łączność $\diamond$:

  $$a \diamond (b \diamond c) = (a \diamond b) \diamond c$$

- Rozdzielność $\gg=$ względem $\diamond$:

  $$a \gg= (\lambda x \to b\,x \diamond c\,x) = (a \gg= b) \diamond (a \gg= c)$$

- $\bot$ elementem neutralnym względem $\diamond$:

  $$a \diamond \bot = \bot \diamond a = a$$

- $\bot$ zerem względem $\gg=$:

  $$\bot \gg= a = b \gg= (\lambda\_ \to \bot) = \bot$$

Podsumowanie

- Programowanie współbieżne z blokadami jest **ZŁE** i, poza bardzo niskopoziomowymi programami, nie powinno być używane.
- Przekazywanie komunikatów i pamięć transakcyjna są mechanizmami o przyjemnych własnościach, co skutkuje prostszymi, poprawniejszymi programami.
- Zmiana sposobu tworzenia programów równoległych jest konieczna, aby móc w pełni wykorzystać możliwości wielordzeniowych procesorów.

Literatura

- Harris T., Marlow S., Peyton Jones S., Herlihy M.: Composable Memory Transactions. *[illegible]* .
- Peyton Jones S.: *[illegible]* . Microsoft Research, Cambridge, 2007.