

Wprowadzenie do STM

Marek Materzok

ZOSIA 2007

Wprowadzenie

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

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.

Programowanie wielowątkowe

- W tek - niezależny tok wykonania, posiadający własne rejestry procesora, ale dzielący pamięć z innymi wątkami.
- Program wielowątkowy jest wykonywany ~~jednocześnie~~ instrukcje jego wątków ~~zostają wykonywane~~ w dowolny możliwy sposób.
- Przeplot następuje na poziomie ~~języka~~ , 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 w tym może być dowolnie przeplatany, wystarczy, że do kodu pracującego na wspólnych danych będzie miał wstępnie tylko jeden w tek naraz
- | Rozwiązanie: niech sekcje krytyczne będą chronione blokadami, aktywnymi i pasywnymi; nadgorliwy w tek sobie poczeka.

Ujarczianie niedeterminizmu

- | Spostrzeżenie: kod pracujący na danych lokalnych dla w tym może być dowolnie przeplatany, wystarczy, że do kodu pracującego na wspólnych danych będzie miał wstępnie tylko jeden w tek naraz
- | Rozwiązanie: niech sekcje krytyczne będą chronione blokadami, aktywnymi i pasywnymi; nadgorliwy w tek 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: ~~h~~, ~~l~~, 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 może nie istnieć w l...
    insertAfter(succ, e);
}
```

Rozwiązanie 1: komunikaty

- | ródłem **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.

Rozwiązanie 2: pamięć transakcyjna

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

Cechy pamięci transakcyjnej

- | Narzut czasowy związany z zarządzaniem transakcjami.
- | Zakaz uwyświelenia 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);
    }
}
```

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 (ł czno , element neutralny):

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

$$a \gg return = a$$

$$return x \gg a = a$$

- Operacje wej/cia/wyj/cia i imperatywne obliczenia tworzą monad IO.

Monada STM

Podstawy:

```

type Ma          -- typ zmiennych dzielonych
Ma      :: a      M    (Ma   )
Ma      :: Ma     M
Ma      :: Ma     a      M  ()
    
```

No dobrze, ale jak wykona transakcj ?

```

tn      :: M      IOa
    
```


§ KSI @ II.UWr

Co zyskałiśmy?

- ! Nie mo na grzeba w zmiennych dzielonych poza transakcjami.

```

p :: a IO ()
px = !m -- bł d typowania!

```

- ! Nie mo na wykonywa wej cia/wyj cia podczas transakcji.

```

p :: a IO ()
p m = !n (do
  x !m
  !m (x + 1)
  !m) -- bł d typowania!

```

Warunkowa synchronizacja

$T_i :: M_i()$

Realizacja: transakcja jest wznowiana, jak tylko inny w tek zmodyfikuje którąś ze zmiennych przeczytanych przez transakcję.

```

T_i :: | T_j = do M_i()
    if a - b < 0
    then T_j
    else M_i(a - b)
    
```


Własności algebraiczne

- Łączność względem \otimes :

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

- Rozdzielność \otimes względem dem \oplus :

$$a \otimes (\lambda x \oplus \mu y) = (\lambda a \otimes x) \oplus (\mu a \otimes y)$$

- 1 elementem neutralnym względem dem \oplus :

$$a \oplus 1 = 1 \oplus a = a$$

- 0 zerem względem \otimes :

$$0 \otimes a = a \otimes 0 = 0$$

Podsumowanie

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

