Obliczenia równoległe w językach interpretowanych

Aleksander Pohl

Katedra Informatyki, Akademia Górniczo-Hutnicza

18 maja 2009

Plan prezentacji

Wstęp teoretyczny

Problem obliczeniowy

Algorytm

Rezultaty

Plan prezentacji

Języki interpretowane

- Ruby
  - dynamiczny, obiektowo-zorientowany
  - popularny dzięki frameworkowi Ruby on Rails
  - idealny do prototypowania
  - alternatywne implementacje: MRI, YARV, JRuby, Rubinius, IronRuby, MagLev, ...
- Python
  - dynamiczny, obiektowo-zorientowany
  - popularny dzięki Google – prototyp wyszukiwarki napisany w Pythonie

Obsługa wątków

- Ruby 1.8 wykorzystuje zielone wątki
- Ruby 1.9 wykorzystuje natywne wątki
- Python wykorzystuje natywne wątki

ale...

- **ze względu na GIL niezależnie od liczby procesorów, tylko jeden wątek realizujący kod Rubiego/Pythona wykonywany w danym momencie**

Przykład (1)

```
1  def calc()
2    10000000.times{|i| i * i}
3  end
4
5  def parallel(runs)
6    Benchmark.measure {
7      threads = []
8      runs.times{ threads << Thread.new{calc()} }
9      threads.each{|t| t.join}
10   }
11 end
```

Przykład (2)

```
1  def parallel_proc(runs)
2    Benchmark.measure {
3      pids = []
4      runs.times{
5        pid = fork
6        unless pid
7          calc()
8          exit
9        end
10     }
11     Process.waitall
12   }
13 end
```

Przykład (3)

Serwer 4-procesorowy

```
1 thread efficiency:  99.673%
1 proc efficiency:  99.849%
2 threads efficiency:  49.973%
2 procs efficiency:  99.696%
3 threads efficiency:  32.777%
3 procs efficiency:  99.934%
```

GIL – Global interpreter lock – Python

- GIL uniemożliwia równoległe wykonanie kodu przez kilka wątków
- w trakcie wykonywania kodu blokowany jest cały interpreter
- niektóre operacji, np. I/O powodują zwolnienie blokady
- blokada może być również zwolniona *explicite* przez natywne rozszerzenia pisane np. w języku C
  - `Py_BEGIN_ALLOW_THREADS`
  - `Py_END_ALLOW_THREADS`
- próby zastosowania mikro-blokad (na pewnych operacjach interpretera) zakończyły się fiaskiem – cały interpreter działał dużo wolniej

GIL – Ruby

- w Ruby 1.9 również wykorzystywany jest GIL, ale w odniesieniu do wirtualnej maszyny Rubiego – YARV
- `rb_thread_blocking_region()`
- dodatkowe problemy w implementacji w niektórych systemach operacyjnych – `exec` w MacOSX powoduje wystąpienie wyjątku, jeśli aktualnie wykonywany jest inny wątek

Propozycja rozwiązania – Erlang

- podejście „share-nothing”
- proste metody komunikacji międzyprocesowej – wbudowane w składnię języka
- „lekkie procesy” – nie są procesami w sensie systemu operacyjnego (działają w tej samej przestrzeni adresowej), ale nie widzą wzajemnie swoich obszarów pamięci
- komunikacja wyłącznie poprzez wymianę komunikatów
- procesy zarządzane są przez scheduler działający w przestrzeni użytkownika
- wykorzystanie wielu procesorów dzięki podejściu m:n – wykorzystywanych jest wiele wątków natywnych, spośród których każdy wyposażony jest we własny scheduler, zarządzający wieloma lekkimi procesami

Propozycja rozwiązania w Rubim

- MVM – Multiple Virtual Machines
- każda maszyna wirtualna uruchomiana w osobnym wątku systemowym
- wszystkie maszyny uruchamiane w ramach jednego procesu – interpretera
- eksperymenty realizowane z wykorzystaniem JRubiego – implementacji języka Ruby w wirtualnej maszynie Javy
- obecnie – ze względu na istnienie wielu alternatywnych implementacji Rubiego ustalane jest wspólne API do obsługi MVM

Plan prezentacji

Motywacja

- Cel doktoratu: opracowanie algorytmów ekstrakcji informacji z polskich tekstów
- Problemy:
  - wymagany jest formalny opis semantyki języka polskiego
  - dla wielu słów trzeba wybrać jedno znaczenie spośród wielu (problem wieloznaczności)
- Propozycja: wykorzystać ontologię *OpenCyc*, która zawiera formalny opis wielu pojęć i przetłumaczyć ją na język polski

OpenCyc – zalety

- Z założenia miała być agnostyczna względem języka
- Posiada bogaty zestaw relacji, które opisują związki pomiędzy pojęciami, które mogą odnosić się zarówno do rzeczowników, czasowników jak i innych części mowy
- Zawiera 300 tys. pojęć oraz 3 mln. asercji dotyczących tych pojęć, wyrażonych w formie logicznej
- Posiada silnik inferencyjny pozwalający na wnioskowanie na temat zgromadzonego zbioru pojęć
- Asercje zorganizowane są w *mikroteorie*, które tworzą strukturę hierarchiczną; pozwala to na zachowanie spójności logicznej wewnątrz danej domeny, bez konieczności zachowania wymogu globalnej niesprzeczności

Słownik angielsko-polski/polsko-angielski Oxford-PWN

- Elektroniczny słownik stworzony dla ludzi (a nie algorytmów tłumaczących)
- Zawiera największą liczbę haseł (weryfikowane w 2004):
  - ok 52 tys. pol. $\rightarrow$ ang.
  - ok 63 tys. ang. $\rightarrow$ pol.
- Zawiera słownictwo ogólne oraz specjalistyczne
- Hasła opisane w języku SGML (protoplaście XML) – trudniejsze do parsowania
- Brak publicznej dokumentacji formatu haseł

Plan prezentacji

Ogólny schemat algorytmu

- Określić listę słów, dla których dostępne są tłumaczenia z angielskiego na polski
- Dla danego słowa angielskiego znaleźć odpowiadające mu pojęcia w Cyc
- Dla każdego znalezionego pojęcia dopasować odpowiadające mu polskie słowo (lub słowa)

Ogólna charakterystyka

- Algorytm posiłkuje się heurystykami skonstruowanymi na podstawie analizy kilkuset przypadków mapowania
- Dodatkowo wykorzystywane są zdefiniowane na potrzeby SSJP podstawowe kategorie semantyczne:
  - AbstractObject , Animal , Plant , ...

  które zostały ręcznie zmapowane na odpowiednie pojęcia OpenCyc
- Heurystyki biorą pod uwagę dwa czynniki:
  - liczbę grup semantycznych tłumaczenia i liczbę pojęć OpenCyc odpowiadających danemu angielskiemu słowu
  - kwalifikatory pojawiające się w opisie tłumaczenia
- Mapowanie posiad „moc”, wskazującą na ile prawdopodobna jest jego poprawność
  - *strong*, *medium*, *weak*

Przykład mapowania

Pojęcie Cyc:

- **Absorbency State** tangible physical quantity, non numeric scalar quantity, disposition

Hasło w słowniku:

- (Chem, Phys) **absorpcyjność, chłonność**

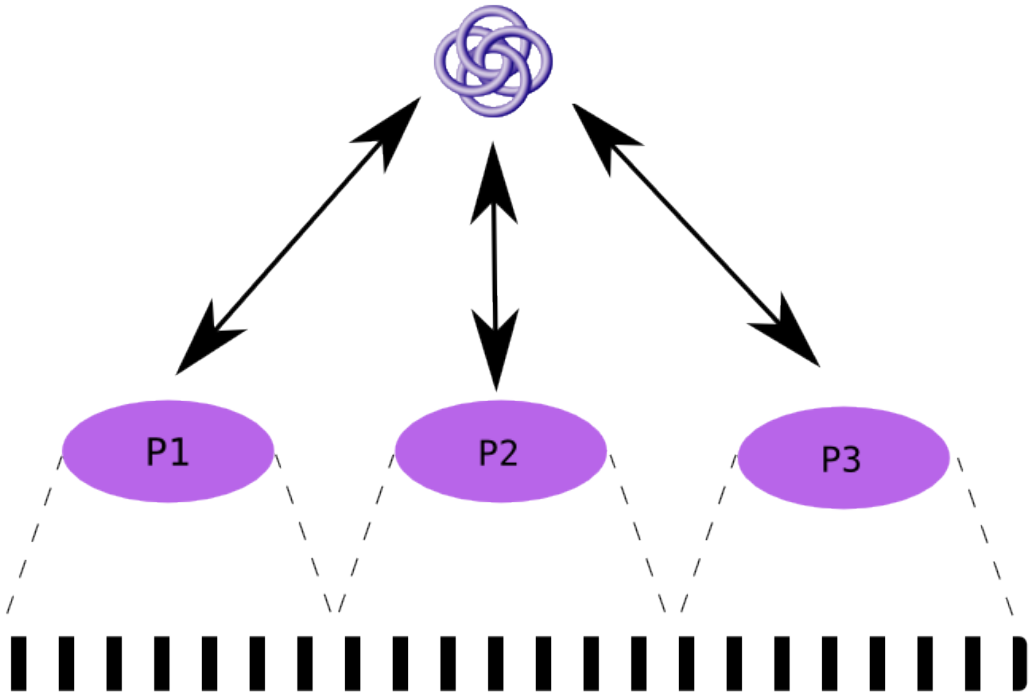
Przypisanie:

- **absorpcyjność** -> **Absorbency State** tangible physical quantity, non numeric scalar quantity, disposition Chem, Phys
- **chłonność** -> **Absorbency State** tangible physical quantity, non numeric scalar quantity, disposition Chem, Phys

Algorytm ogólny

```
1  process_entries(limit, offset) do |entry, cyc_terms|
2    print_potential_matches(entry, cyc_terms)
3    mappings = find_mappings(entry, cyc_terms)
4    mappings.each do |mapping|
5      global_result.append([mapping.word.to_s],mapping)
6    end
7    print_matches(entry, cyc_terms, mappings)
8  end
9  post_process(global_result)
```

Podział zadania na procesy



Algorytm równoległy

```
1  def process_entries(limit=100,offset=0,&block)
2    proc_count = 2
3    #...
4    parent_pid = Process.pid
5    proc_count.times do |proc_nr|
6      first, last = compute_indices(...)
7      unless fork
8        process_group(entries_groups[first..last],&block)
9        return
10     end
11   end
12   Process.waitall
13   #...
14 end
```

Problemy związane z równoległością

1. podział haseł dla poszczególnych procesów
   - wczytanie wszystkich haseł przed dokonaniem podziału
   - ignorowanie różnic w czasie wykonania mapowania dla poszczególnych grup haseł
2. wspólny plik logowania
   - semafor IPC System V
3. wspólna tablica wyników
   - ostateczny wynik mapowania zależał m.in. od tego czy dane słowo polskie było przypisane do wielu symboli Cyc
   - tablica asocjacyjna realizowana w postaci współdzielonej pamięci transakcyjnej, zapewniająca możliwość równoczesnego zapisu + semafory dla poszczególnych słów (klucze tablicy)

Plan prezentacji

Szczegóły eksperymentu

- Mapowane były wyłącznie rzeczowniki (kategoria gramatyczna *n*, *npl*, *prn*)
- Zostało stworzonych ∼27 tys. mapowań dla 16 tys. leksemów
- Weryfikacji poddana ∼3,5 tys. mapowań (∼ 12%)
- Precyzja: **54%**
- Pokrycie: **niezweryfikowane ze względu na brak zasobu referencyjnego**

moc	strong	medium	weak
precyzja	64,7%	49,8%	23,1%

Wyniki szczegółowe z podziałem na kategorie

	AbstractObj	Animal	Artifact	BodyPart	Event
size	4652	878	4807	758	6957
strong	48.29%	87.5%	44.86%	70.42%	54.69%
medium	38.97%	61.64%	32.69%	84.21%	32.30%
weak	22.22%	18.75%	31.11%	15.38%	16.26%
overall	**42.39%**	**76.42%**	**40.22%**	**66.01%**	**35.29%**

	Human	Instrument	Location	Meter	NaturalObj
size	2551	3486	2373	110	1432
strong	80.23%	57.26%	62.42%	91.89%	76.92%
medium	79.10%	54.90%	63.43%	80.95%	60.52%
weak	29.62%	12.0%	14.28%	100.0%	69.23%
overall	**74.71%**	**54.37%**	**59.61%**	**88.33%**	**72.61%**

Wyniki szczegółowe z podziałem na kategorie

	Proper	Self	Set	State	Structure
size	168	659	592	1590	358
strong	79.31%	53.84%	51.61%	82.95%	60.60%
medium	73.03%	62.5%	30.0%	69.23%	31.81%
weak	54.54%	46.42%	10.0%	37.14%	0.0%
overall	**72.86%**	**54.65%**	**39.34%**	**69.71%**	**41.53%**

	Food	Plant			
size	489	208			
strong	84.31%	97.5%			
medium	60.0%	83.33%			
weak	21.42%	25.0%			
overall	**67.77%**	**89.28%**			

Wyniki zrównoleglenia – Ruby 1.8, system czteroprocesorowy

200 jednostek					
# procesów	1	2	3	4	5
średnia [s]	56,11	39,15	36,62	30,13	29,86
odchylenie [s]	0,11	0,09	0,4	0,28	0,3
przyspieszenie	1	1,43	1,53	1,86	1,86
efektywność [%]	100	71,67	51,08	46,56	37,25
1000 jednostek					
# procesów	1	2	3	4	5
średnia [s]	369,9	213,18	179,45	165,88	157,42
odchylenie [s]	0,77	3,06	1,38	1,53	5,67
przyspieszenie	1	1,74	2,06	2,23	2,23
efektywność [%]	100	86,76	68,71	55,75	44,6

Czas i efektywność mapowania równoległego – Ruby 1.8

Wyniki zrównoleglenia – Ruby 1.9, system czteroprocesorowy

200 jednostek					
# procesów	1	2	3	4	5
średnia [s]	39,5	28,09	26,92	22,94	23,32
odchylenie [s]	0,97	1,08	0,96	0,78	0,65
przyspieszenie	1	1,41	1,47	1,72	1,69
efektywność [%]	100	70,3	48,92	43,04	33,88
1000 jednostek					
# procesów	1	2	3	4	5
średnia [s]	229,47	141,98	127,31	120,7	116,5
odchylenie [s]	11,4	5,11	5,63	5,43	4,71
przyspieszenie	1	1,62	1,8	1,9	1,97
efektywność [%]	100	80,81	60,08	47,53	39,39

Czas i efektywność mapowania równoległego – Ruby 1.9

Dziękuję!