Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Konrad Paziewski
Nr albumu: 306410

Automatyczne zrównoleglanie programów w językach czysto funkcyjnych

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Marcina Benke
Instytut Informatyki

Listopad 2015

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W pracy przedstawiono język funckyjny Cthulhu i jego implementację pozwalającą na automatyczne zrównoleglanie programów.

Słowa kluczowe

blabaliza różnicowa, fetory σ-ρ, fooizm, blarbarucja, blaba, fetoryka, baleronik

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.127. Blabalgorithms
D.127.6. Numerical blabalysis

Tytuł pracy w języku angielskim

Automatic parallelisation of pure functional programming languages

Spis treści

Wprowadzenie 5

1. Podstawowe pojęcia 7
1.1. Definicje 7
1.2. Implementacja 7

2. Wcześniejsze implementacje blabalizatora różnicowego 9
2.1. Podejście wprost 9

3. Idea 11

4. Opis języka 13
4.1. System typów 13

5. Semantyka 15

6. Implementacja 17
6.1. Asembler cthulhu 17
6.2. Generator kodu 17
6.2.1. Analiza leksykalno-składniowa 17
6.2.2. Analiza semantyczna 17
6.2.3. Przygotowanie listy instrukcji 18
6.3. Środowisko uruchomieniowe 18

7. Dokumentacja użytkowa i opis implementacji 19

8. Podsumowanie 21
8.1. Perspektywy wykorzystania w przemyśle 21

A. Główna pętla programu zapisana w języku TōFoo 23

B. Przykładowe dane wejściowe algorytmu 25

C. Przykładowe wyniki blabalizy (ze współczynnikami σ-ρ) 27

Bibliografia 29

Wprowadzenie

Prawo Moore'a od lat siedemdziesiątych dwudziestego wieku dyktowało tempo wzrostu wydajności układów scalonych. Wykładniczy wzrost ilości tranzystorów na jednostkę powierzchni przy zachowaniu stałej ceny pozwalał zakładać, że częstotliwość taktowania przeciętnego procesora będzie rosnąć dwukrotnie co dwa lata. Niestety, w ostatnich latach tendencja ta zanika. Prawo Moore'a nadal działa w sferze miniaturyzacji elementów układów scalonych. Jednakże, nie przekłada się to już na oczekiwany wzrost wydajności układu. Częstotliwość taktowania procesorów nie rośnie już wykładniczo.

Dużo prostszym sposobem na zwiększenie teoretycznej mocy obliczeniowej procesora jest zwiększenie liczby rdzeni. Możliwość równoległego wykonywania n wątków obliczeń w teorii zwiększa ilość operacji wykonywanych w jednostce czasu n-krotnie w stosunku do układu jednowątkowego. Uruchomienie kilku procesów wykorzystujących wiele rdzeni procesora jest proste. Nowoczesne systemy operacyjne zapewniają skuteczną izolację między poszczególnymi procesami. Taka architektura pozwala na efektywne równoległe wykonywanie niezależnych od siebie zadań. Niestety wykorzystanie wielu jednostek obliczeniowych procesora do wykonania jednego zadania nie jest zagadnieniem trywialnym. Wymaga to podziału zadania na podzadania, rozdzielenia podzadan pomiędzy poszczególne rdzenie, a następnie scalenia wyniku. Niestety programy współbieżne są trudne do przygotowania, analizy i modyfikacji. Często kod odpowiedzialny za synchronizację i podział pracy między poszczególne wątki wykonania jest bardziej skomplikowany niż sama logika programu. Przyjrzyjmy się następującym dwóm wersjom tej samej funkcji:

```
template<typename A, typename B, typename F>
void map(F& f, std::vector<A>& input, std::vector<B>& output) {
    output.resize(input.size());
    for ( unsigned i = 0; i < input.size(); ++i ){
        output[i] = f(input[i]);
    }
}

template<typename A, typename B, typename F>
void map_par(F& f, std::vector<A>& input, std::vector<B>& output) {
  output.resize(input.size());
  std::vector<std::thread> threads;
  auto operation = [&input, &output, &f](int i) {
    output[i] = f(input[i]);
  };
  for ( unsigned i = 0; i < input.size(); ++i )
    threads.emplace_back(operation, i);
  for ( unsigned i = 0; i < threads.size(); ++i )
    threads[i].join();
}
```

Zauważmy, że większość kodu drugiej z zademonstrowanych tu procedur odpowiada za zrównoleglenie pracy. Co więcej, na pierwszy rzut oka ciężko ocenić jakie rzeczywiście zadanie

wykonuje ta funkcja.

Tego typu problemy często sprawiają że programiści uciekają od pisania programów wielowątkow

Wydajność Jednakże, pełne wykorzystanie

Rozdział 1

Podstawowe pojęcia

Pojęciem pierwotnym blabalii fetorycznej jest *blaba*. Blabaliści nie podają jego definicji, mówiąc za Ciach-Pfe t-ām Kûn (fooistyczny mędrzec, XIX w. p.n.e.):

> Blaba, który jest blaba, nie jest prawdziwym blaba.
>
> *tłum. z chińskiego Robert Blarbarucki*

1.1. Definicje

Oto dwie definicje wprowadzające podstawowe pojęcia blabalii fetorycznej:

Definicja 1.1.1 *Silny, zwarty i gotowy fetor bazowy nazwiemy* skupieniem.

Definicja 1.1.2 Fetorem *nazwiemy skupienie blaba spełniające następujący* aksjomat reperkusatywności*:*

$$\forall \mathcal{X} \in Z(t)\ \exists \pi \subseteq \oint_{\Omega^2} \kappa \leftrightarrow 42$$

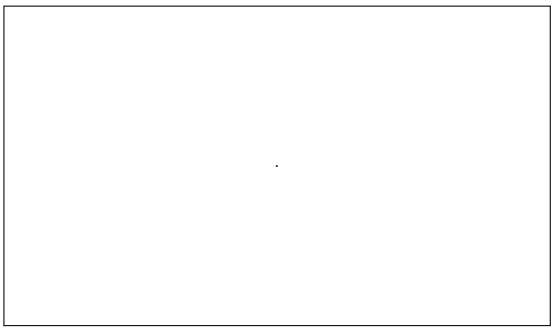
1.2. Implementacja

Teoretycy blabalii (zob. np. pracę [Głomb04]) zadowalają się niekonstruktywnym opisem natury fetorów.

Podstawowym narzędziem blabalii empirycznej jest blabalizator różnicowy. Przyrząd ten pozwala w sposób przybliżony uzyskać współczynniki rozkładu Głombaskiego dla fetorów bazowych i harmonicznych. Praktyczne znaczenie tego procesu jest oczywiste: korzystając z reperkusatywności pozwala on przejść do przestrzeni Λ^{∇}, a tym samym znaleźć retroizotonalne współczynniki semi-quasi-celibatu dla klatek Rozkoszy (zob. [Rozk93]).

Klasyczne algorytmy dla blabalizatora różnicowego wykorzystują:

1. dualizm falowo-korpuskularny, a w szczególności
   (a) korpuskularną naturę fetorów,
   (b) falową naturę blaba,
   (c) falowo-korpuskularną naturę gryzmołów;
2. postępującą gryzmolizację poszczególnych dziedzin nauki, w szczególności badań systemowych i rozcieńczonych;



Rysunek 1.1: Artystyczna wizja blaba w obrazie węgierskiego artysty Josipa A. Rozkoszy pt. „Blaba”

3. dynamizm fazowy stetryczenia parajonizacyjnego;
4. wreszcie tradycyjne opozycje:
   - duch — bakteria,
   - mieć — chcieć,
   - myśl — owsianka,
   - parafina — durszlak[1],
   - logos — termos

   z właściwym im przedziwym dynamizmem.

[1]Więcej o tym przypadku — patrz prace Gryzybór-Głombaskiego i innych teoretyków nurtu teoretyczno-praktycznego badań w Instytucie Podstawowych Problemów Blabalii w Fifie.

Rozdział 2

Wcześniejsze implementacje blabalizatora różnicowego

2.1. Podejście wprost

Najprostszym sposobem wykonania blabalizy jest siłowe przeszukanie całej przestrzeni rozwiązań. Jednak, jak łatwo wyliczyć, rozmiar przestrzeni rozwiązań rośnie wykładniczo z liczbą fetorów bazowych. Tak więc przegląd wszystkich rozwiązań sprawdza się jedynie dla bardzo prostych przestrzeni lamblialnych. Oznacza to, że taka metoda ma niewielkie znaczenie praktyczne — w typowym przypadku z życia trzeba rozważać przestrzenie lamblialne wymiaru rzędu 1000.

W literaturze można znaleźć kilka prób opracowania heurystyk dla problemu blabalizy (por. [Zen69]). Korzystając z heurystyk daje się z pewnym trudem dokonać blabalizy w przestrzeni o np. 500 fetorach bazowych. Należy jednak pamiętać, że heurystyka nie daje gwarancji znalezienia najlepszego rozwiązania. Fifak w pracy [Fif01] podaje, jak dla dowolnie zadanej funkcji oceniającej skonstruować dane, dla których rozwiązanie wygenerowane przez algorytm heurystyczny jest dowolnie odległe od rzeczywistego.

Rozdział 3

Idea

Należy zwrócić uwagę że cthulhu jest językiem w pełni czystym. Semantyka nie wyznacza żadnej szczególnej kolejności operacji, nie zawiera też pojęcia stanu. Oznacza to zatem, że mamy dowolność co do wyboru wyrażeń które zostaną wykonane równolegle.

Rozdział 4

Opis języka

4.1. System typów

Zbiór

Cthulhu jest prostym językiem funkcyjnym o ograniczonej liczbie dostępnych konstrukcji. Program składa się z listy definicji typów i funkcji.

Program w tym języku modeluje funkcję ze zbioru liczb naturalnych w zbior liczb naturalnych.

Rozdział 5

Semantyka

Proponowany język programowania jest pełnowartościowym językiem funkcyjnym.

Rozdział 6

Implementacja

Przygotowana przez mnie i opsywana w tym dokumencie realizacja języka Cthulhu składa się z dwóch części. Pierwszą z nich jest napisany w Haskellu generator kodu. Jego zadaniem jest statyczna analiza programu wejściowego i wygenerowanie równoważnego programu w C++. Generator kodu sprawdza poprawność kodu a także dba o optymalizacje oraz wybór wyrażeń które będą obliczane równolegle. Drugim elementem jest przygotowane w C++ środowisko uruchomieniowe. Fragmenty te są połączone przez pewnego rodzaju zbiór instrukcji przypominających ideowo instrukcje asemblerowe.

6.1. Asembler cthulhu

6.2. Generator kodu

Zadaniem generatora kodu jest przygotowanie ciągu instrukcji które zostaną później wykonane przez środowisko uruchomieniowe. Proces przygotwania tych instrukcji składa się z czterech etapów:

- analiza leksykalno-składniowa
- analiza semantyczna
- przygotowanie listy instrukcji
- wygenerowanie pliku nagłówkowego C++ zawierającego instrukcje z poprzedniego kroku

6.2.1. Analiza leksykalno-składniowa

Parser języka jest generowany przy pomocy narzędzia BNFC. Automatycznie, na podstawie zadanej gramatyki generowane są moduły odpowiedzialne za wszystkie tradycyjne etapy analizy leksykalno-składniowej - to znaczy lekser i parser. Wygenerowany zostaje też moduł opisujący strukturę będącą typem wyniku parsowania.

6.2.2. Analiza semantyczna

Celem analizy semantycznej jest sprawdzenie poprawności programu a także powiązanie identyfikatorów z ich znaczeniem i typem. Niestandardowe jest to że nie jest kompilowany cały program a jedynie te funkcje które są osiągalne - byc może nie bezpośrednio z funkcji głównej

programu. Z drugiej strony typy i funkcje parametryzowane kompilowane są oddzielnie dla każdej kombinacji parametrów.

6.2.3. Przygotowanie listy instrukcji

W pierwszej fazie analizy generator Co więcej, generator kodu musi wygenerować dodatkowe informację w celu umoż Przygotowana przeze mnie implementacja Cthulhu inicjalizuje obliczenie równoległe tylko dla wyrażeń będących wywołaniami funkcji. Co więcej, jedyne funkcje które

6.3. Środowisko uruchomieniowe

Środowisko uruchomieniowe

Rozdział 7

Dokumentacja użytkowa i opis implementacji

Program przygotowany dla systemu operacyjnego M$ Windows uruchamia się energicznym dwumlaskiem na jego ikonce w folderze `\\FIDO\FOO\BLABA`. Następnie kolistym ruchem ręki należy naprowadzić kursor na menu `Blabaliza` i uaktywnić pozycję `Otwórz plik`. Po wybraniu pliku i zatwierdzeniu wyboru przyciskiem `OK` rozpocznie się proces blabalizy. Wyniki zostaną zapisane w pliku o nazwie `99-1a.tx.43` w bieżącym folderze.

Rozdział 8

Podsumowanie

W pracy przedstawiono pierwszą efektywną implementację blabalizatora różnicowego. Umiejętność wykonania blabalizy numerycznej dla danych „z życia” stanowi dla blabalii fetorycznej podobny przełom, jak dla innych dziedzin wiedzy stanowiło ogłoszenie teorii Mikołaja Kopernika i Gryzybór Głombaskiego. Z pewnością w rozpocznynającym się XXI wieku będziemy obserwować rozkwit blabalii fetorycznej.

Trudno przewidzieć wszystkie nowe możliwości, ale te co bardziej oczywiste można wskazać już teraz. Są to:

- degryzmolizacja wieńców telecentrycznych,
- realizacja zimnej reakcji lambliarnej,
- loty celulityczne,
- dokładne obliczenie wieku Wszechświata.

8.1. Perspektywy wykorzystania w przemyśle

Ze względu na znaczenie strategiczne wyników pracy ten punkt uległ utajnieniu.

Dodatek A

Główna pętla programu zapisana w języku TōFoo

```
[[foo]{,}[[a3,((([(,),{[[]]}]),
  [1; [{,13},[[[11],11],231]]].
  [13;[!xz]].
  [42;[{,x},[[2],{'a'},14]]].
  [br;[XQ*10]].
 ), 2q, for, [1,]2, [..].[7]{x}],[(((,[[1{{123,},},;.112]],
        else 42;
    . 'b'.. '9', [[13141],{13414}], 11),
 [1; [[134,sigma],22]].
 [2; [[rho,-],11]].
 )[14].
 ), {1234}],]. [map [cc], 1, 22]. [rho x 1]. {22; [22]},
       dd.
 [11; sigma].
        ss.4.c.q.42.b.ll.ls.chmod.aux.rm.foo;
 [112.34; rho];
        00111010101010101010101010101011111010010
 [22%f4].
 cq. rep. else 7;
 ]. hlt
```

Dodatek B

Przykładowe dane wejściowe algorytmu

α	β	γ_7
901384	13784	1341
68746546	13498	09165
918324719	1789	1310
9089	91032874	1873
1	9187	19032874193
90143	01938	0193284
309132	−1349	−149089088
0202122	1234132	918324098
11234	−109234	1934

Dodatek C

Przykładowe wyniki blabalizy (ze współczynnikami σ-ρ)

	Współczynniki Głombaskiego	ρ	σ	σ-ρ
γ_0	1,331	2,01	13,42	0,01
γ_1	1,331	113,01	13,42	0,01
γ_2	1,332	0,01	13,42	0,01
γ_3	1,331	51,01	13,42	0,01
γ_4	1,332	3165,01	13,42	0,01
γ_5	1,331	1,01	13,42	0,01
γ_6	1,330	0,01	13,42	0,01
γ_7	1,331	16435,01	13,42	0,01
γ_8	1,332	865336,01	13,42	0,01
γ_9	1,331	34,01	13,42	0,01
γ_{10}	1,332	7891432,01	13,42	0,01
γ_{11}	1,331	8913,01	13,42	0,01
γ_{12}	1,331	13,01	13,42	0,01
γ_{13}	1,334	789,01	13,42	0,01
γ_{14}	1,331	4897453,01	13,42	0,01
γ_{15}	1,329	783591,01	13,42	0,01

Bibliografia

[Bea65] Juliusz Beaman, *Morbidity of the Jolly function*, Mathematica Absurdica, 117 (1965) 338–9.

[Blar16] Elizjusz Blarbarucki, *O pewnych aspektach pewnych aspektów*, Astrolog Polski, Zeszyt 16, Warszawa 1916.

[Fif00] Filigran Fifak, Gizbert Gryzogrzechotalski, *O blabalii fetorycznej*, Materiały Konferencji Euroblabal 2000.

[Fif01] Filigran Fifak, *O fetorach σ-ρ*, Acta Fetorica, 2001.

[Głomb04] Gryzybór Głombaski, *Parazytonikacja blabiczna fetorów — nowa teoria wszystkiego*, Warszawa 1904.

[Hopp96] Claude Hopper, *On some Π-hedral surfaces in quasi-quasi space*, Omnius University Press, 1996.

[Leuk00] Lechoslav Leukocyt, *Oval mappings ab ovo*, Materiały Białostockiej Konferencji Hodowców Drobiu, 2000.

[Rozk93] Josip A. Rozkosza, *O pewnych własnościach pewnych funkcji*, Północnopomorski Dziennik Matematyczny 63491 (1993).

[Spy59] Mrowclaw Spyrpt, *A matrix is a matrix is a matrix*, Mat. Zburp., 91 (1959) 28–35.

[Sri64] Rajagopalachari Sriniswamiramanathan, *Some expansions on the Flausgloten Theorem on locally congested lutches*, J. Math. Soc., North Bombay, 13 (1964) 72–6.

[Whi25] Alfred N. Whitehead, Bertrand Russell, *Principia Mathematica*, Cambridge University Press, 1925.

[Zen69] Zenon Zenon, *Użyteczne heurystyki w blabalizie*, Młody Technik, nr 11, 1969.