Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Implementacja maszyny wirtualnej dla funkcyjnych języków programowania wspierających przetwarzanie współbieżne.

Implementation of a virtual machine for functional programming languages with support for concurrent computing.

Autor: *Kajetan Rzepecki*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Piotr Matyasik*

Kraków, 2014

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję Lucynie oraz siostrze Alicji za cierpliwość i wsparcie podczas tworzenia pracy dyplomowej.

Spis treści

1. Podsumowanie

Projekt implementuje kompletną maszynę wirtualną nazwaną ThesisVM, w której skład wchodzą trzy moduły: interpreter kodu bajtowego oparty o model **Three Instruction Machine**, kolektor obiektów nieosiągalnych implementujący algorytm **leniwego zliczania referencji** oraz model przetwarzania współbieżnego oparty o **symetryczny multiprocessing** i **Model Aktorowy**.

1.1. Interpreter kodu bajtowego

Interpreter kodu bajtowego pozwala na uruchamianie nietrywialnych programów napisanych w języku pośredniej reprezentacji **TVMIR**. Język ten kompilowany jest do kodu bajtowego, składającego się z szeregu instrukcji w architekturze **CISC**, które operują na danych za pośrednictwem czterych rejestrów: `IP`, `Stack`, `Env` oraz `VStack`.

Niestety reprezentacja kodu bajtowego nie jest optymalna, ponieważ wykorzystuje dedykowane obiekty złożone ThesisVM oraz listy pojedynczo wiązane zbudowane w oparciu o pary znane z języków z rodziny `Lisp`. Alternatywnym i niewątpliwie szybszym pod wieloma względami rozwiązaniem byłoby wykorzystanie ułożonych kolejno w pamięci liczb całkowitych.

Ograniczenia czasowe projektu nie pozwoliły również na rozwiązanie problemu cache'owania obliczonych wartości *kontynuacji*. Interpreter kodu bajtowego jest *leniwy*, to znaczy ewaluuje on wartości pewnych kwantów obliczeń dopiero w momencie, gdy są wymagane. Obecna implementacja nie zapamiętuje ich nowej wartości, co może prowadzić do zwielokrotnienia niektórych obliczeń.

Problem ten jest złożony w kontekście języków programowania wspierających przetwarzanie współbieżne ponieważ wymaga synchronizacji wielu wątków, które mogą próbować ewaluować tę samą kontynuację.

1.2. Kolektor obiektów nieosiągalnych

Kolektor obiektów nieosiągalnych ThesisVM został zaimplementowany w oparciu o algorytm *leniwego* zliczania referencji. Dużą jego zaletą jest szybkość alokacji i dealoka-

cji, która została osiągnięta poprzez zastosowanie *listy niedawno zwolnionych obiektów*, pełniącej rolę bufora alokacji.

Kolejną wadą wybranego algorytmu jest konieczność wykorzystywania operacji atomowych i barier pamięci w celu wymuszania kolejności zachodzenia operacji na pamięci, co nieznacznie ogranicza wydajność maszyny wirtualnej. W przyszłości zastosowane mogą być dodatkowe algorytmy i optymalizacje kolekcji "śmieci" w celu złagodzenia tego problemu.

1.3. Przetwarzanie współbieżne

– Przeanalizować szybkość przesyłania wiadomości/konieczność czekania procesów, wielkość kolejek wiadomości.

1.4. Kierunki przyszłego rozwoju

– Opisać plany na przyszły rozwój projektu (priorytet procesów, load balancing SMP, wsparcie dla `Core Erlang`, bytecode threading, przebiegi optymalizacyjne podczas kompilacji, umożliwienie dystrybucji na wiele maszyn, zapasowy kolektor śmieci cyklicznych, opcja wykorzystania sterty prywatnej i autonomicznego alokatora, natywna kompilacja JIT, wektory, data-level parallelism, optymalizacja wykorzystania stosu, hardłerowa implementacja interpretera kodu bajtowego).

Bibliografia