Rozdział 1

Programowanie funkcyjne wraca do łask

Michael Swaine

Lato, gdy przeniosłem się do Doliny Krzemowej, było tym latem, w którym zlikwidowano ILLIAC IV w Moffett Field.

Widzieliśmy już ten film

W 1981 roku centrum zainteresowania informatyki przeniosło się z ogromnych maszyn obsługiwanych przez kapłanów w białych fartuchach na tanie komputery osobiste tworzone i programowane przez niechlujnych hakerów. Ja przeniosłem się ze środkowego zachodu do Palo Alto i zgłosiłem się do pomocy w utworzeniu nowego tygodnika obsługującego tych niechlujów. W międzyczasie, w niewielkiej odległości – w ośrodku NASA Moffett Field oficjalnie zamknięto i rozłożono na części komputer, który zainspirował Stanleya Kubricka i Arthura C. Clarke'a do wymyślenia komputera HAL 9000.

ILLIAC IV to legendarny punkt zwrotny w projektowaniu komputerów, mający podobną pozycję w długiej i skomplikowanej historii programowania funkcyjnego.

Koncepcją leżącą u podstaw ILLIAC IV było oderwanie się od modelu sekwencyjnego, który od początku dominował w informatyce. Pewne obszary problemów, takie jak mechanika płynów, lepiej nadawały się do przetwarzania równoległego, a ILLIAC IV został specjalnie zaprojektowany dla tego rodzaju równoległych problemów – takich, w których jedna instrukcja mogła zostać zastosowana równolegle do wielu zbiorów danych. Jest to znane pod skrótem SIMD (*single instruction, multiple data* – jedna instrukcja, wiele danych). Bardziej ogólny przypadek – MIMD (*multiple instructions operating on multiple data*

sets – wiele instrukcji działających na wielu zbiorach danych) jest trudniejszy. Ale każdy rodzaj równoległego programowania wymaga nowych algorytmów i nowego podejścia. I stanowi zaproszenie do programowania funkcyjnego.

Programowanie funkcyjne, określane skrótem FP, zgodnie z Wikipedią „traktuje obliczenia jako wyznaczanie matematycznych wartości funkcyjnych i unika zmiany stanu i zmiany danych". Może to posłużyć za podstawową definicję, ale my musimy się zagłębić dużo bardziej.

FP funkcjonuje od lat pięćdziesiątych XX wieku, gdy John McCarthy wymyślił język Lisp, ale dążenie do przetwarzania równoległego dało programowaniu funkcyjnemu nowy impet. Unikanie zmiany stanu było wtedy trudną do przełknięcia pigułką, ale pasowało do modelu SIMD. To doprowadziło do rozwoju nowych języków i nowych cech FP w istniejących językach. Fortran był wtedy językiem, w którym większość wykonywała obliczenia naukowe, więc naturalne było jego rozszerzanie. Programiści ILLIAC IV mogli pisać programy w IVTRAN, TRANQUIL lub CFD, zrównoleglonych wersjach FORTRAN-u. Powstała też zrównoleglona wersja języka ALGOL.

Dla odpowiednich typów problemów, które można rozwiązać za pomocą SIMD, ILLIAC IV był najszybszym komputerem na świecie i utrzymał ten tytuł do czasu jego wycofania z eksploatacji w 1981 roku. Był o rząd wielkości szybszy od każdego komputera w tamtych czasach i był doskonale dopasowany do swoich docelowych zastosowań i programowania funkcyjnego.

Ale era ta gwałtownie się skończyła. 7 września 1981 roku ILLIAC IV został na dobre zatrzymany. Można dziś zobaczyć jego fragment w muzeum historii komputerów w Mountain View, niedaleko miejscowości Moffett Field.

Dlaczego ta era dobiegła końca? Według Wikipedii: „Illiac IV należał do klasy komputerów równoległych, określanych jako SIMD. Prawo Moore'a prześcignęło specjalizowane podejście stosowane w SIMD ILLIAC, sprawiając, że podejście MIMD stało się preferowane dla niemal wszystkich obliczeń naukowych".

Po drodze zyskał jeszcze jedno miejsce w historii – inspirację dla komputera HAL 9000 w filmie *2001: Odyseja kosmiczna*. Arthur C. Clarke nie był neofitą w kwestii komputerów – rozmawiał z Turingiem w Bletchley Park i pilnie śledził postępy w dziedzinie mikrokomputerów. Clarke był zaintrygowany, gdy dowiedział się o działaniu ILLIAC IV na Uniwersytecie Stanu Illinois w Urbana-Champaign i uczcił ten kampus w swoim filmie jako miejsce narodzin HAL-a 9000.

Przejdźmy do roku 2000. Zastosowanie, które sprawiło, że FP stało się warte nauki, okazało się znowu przetwarzaniem równoległym, ale napędzanym pojawieniem się procesorów wielordzeniowych. „Twórcy chipów", jak wtedy twierdziłem, „mówili w istocie, że wdrażanie prawa Moore'a to teraz domena

oprogramowania. Oni skoncentrują się na wstawianiu coraz więcej rdzeni do kości, a programiści muszą tak przerobić swoje programy, aby wykorzystać możliwości przetwarzania równoległego w ich układach".

Impet, jaki zyskało FP na początku XXI wieku, wynikał z chęci oderwania się od modelu sekwencyjnego, przy czym zaoferowane zostały różne podejścia. Osoby, które zainwestowały wiele w umiejętności i narzędzia w Javie, nie chciały odkładać na bok bibliotek kodów, umiejętności i narzędzi, na których się opierały, mogły używać języka Scala Martina Odersky'ego, ostatnio wprowadzonego na platformie Javy. Zaoferowano go też na platformie .NET, choć (niestety) zaniechano jego wsparcia w 2012 roku. Użytkownicy .NET powinni raczej przymierzać się do F#, utworzonego przez Dona Syme'a z Microsoft Research. Kto chce mieć czysto funkcyjny język, może skorzystać z Erlang, opracowanego dla wysoce równoległego programowania przez Joe Armstronga z Ericssona, z języka Haskell lub wielu innych możliwości.

Tym, co oferują wszystkie te języki, jest możliwość pracy w paradygmacie funkcyjnym. Dwie podstawowe cechy paradygmatu funkcyjnego to potraktowanie wszystkich obliczeń jako wyznaczania wartości funkcyjnych oraz unikanie zmiany stanu i mutowalnych danych. Ale są też inne wspólne cechy programowania funkcyjnego:

- funkcje pierwszoklasowe – funkcje mogą służyć jako argumenty i wyniki funkcji;
- rekurencja jako podstawowe narzędzie iteracji;
- szerokie stosowanie dopasowywania do wzorców;
- leniwe wartościowanie, co pozwala na tworzenie nieskończonych ciągów.

Programiści iteracyjni, którzy po raz pierwszy stykają się z FP, mogą jeszcze dodać – jest ono wolne i niejasne. W istocie żadna z tych opinii nie występuje zawsze, ale wymaga to zmiany sposobu myślenia o problemach i różnych algorytmach. Aby programy działały odpowiednio, muszą mieć lepsze wsparcie ze strony języka niż to, które było powszechne w pierwszej dekadzie XXI wieku. To się zmieni w kolejnym dziesięcioleciu, a wraz z tym zmienią się przyczyny, dla których FP przyciąga ludzi do siebie.

Nowe argumenty za programowaniem funkcyjnym

Po upływie dekady ponownie wróciła sprawa programowania funkcyjnego. Pojawiło się nowe wsparcie językowe i sprawa FP stała się szersza.

Chociaż równoległość była tradycyjnie siłą napędową programowania funkcyjnego, teraz gdy mówi się o FP, częściej odwołuje się do zdolności spojrzenia na problem na wyższym poziomie oraz o zaletach niemutowalności. Zamiast

patrzeć na FP jak na dziwny sposób, który trzeba zastosować, aby poradzić sobie z równoległością, nowym argumentem jest *bardziej naturalny* sposób pisania, bliższy pojęciom i terminologii dla naszej dziedziny. Ludzie używają FP do aplikacji, które nie wymagają równoległości, aby programować bardziej efektywnie i jasno, działając bliżej sposobu myślenia o swoich problemach.

Mówi się, że zamiast konieczności przełożenia problemu na język programowania, adaptujemy język do problemu.

A jeśli mamy dostępnych wiele rdzeni i równoległość może być korzystna dla naszego kodu, to mamy równoległość za darmo. Neal Ford powiedział: „Ostatnie mądre innowacje w bibliotekach [Clojure] pozwoliły przepisać funkcję odwzorowania tak, aby automatycznie stawała się równoległa, co oznacza, że wszystkie działania związane z odwzorowaniami skorzystają z przyspieszenia wydajności bez interwencji dewelopera".

Dobre dopasowanie FP do współbieżności przemawia do ludzi, którzy piszą aplikacje wieloprocesorowe, aplikacje o dużej dostępności, serwery WWW dla sieci społecznościowych i do wielu innych zastosowań. Wyższy poziom abstrakcji FP przemawia do osób, które szukają krótszego czasu pracy dewelopera lub lepiej zrozumiałego kodu. FP kładzie nacisk na niemutowalność, co silnie przemawia do każdego, kto jest zainteresowany niezawodnością.

Wzrost sprzedaży stanowi prawdziwą zmianę z argumentacji na rzecz programowania funkcyjnego z czasów ILLIAC IV. Dziś są nowe powody, aby spojrzeć na FP, a także lepsze wsparcie językowe FP i większy wybór podejść. Zaczyna się to od wyboru języka. Można wygodnie trzymać się znanych sobie języków i ich narzędzi, wprowadzając funkcyjność tam, gdzie jest dla nas użyteczna. Lub też można przejść do języka zbudowanego od podstaw do programowania funkcyjnego. W tej książce zobaczymy oba te podejścia.

To ekscytujący czas na poznawanie programowania funkcyjnego, a korzyści z niego wynikające są znaczne. Ale wymaga to jednak innego sposobu myślenia.

W następnym rozdziale Michael Bevilacqua-Linn zaprasza nas do myślenia o programowaniu w sposób funkcyjny.