Sztuka tworzenia wydajnego kodu

Przewodnik po zaawansowanych technikach wykorzystywania sprzętu i kompilatorów

Fedor G. Pikus

Helion

Tytuł oryginału: The Art of Writing Efficient Programs: An advanced programmer's guide to efficient hardware utilization and compiler optimizations using C++ examples

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-9250-2

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/szttwo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Wydajność i współbieżność — wprowadzenie

Motywacja to kluczowy element procesu uczenia, dlatego trzeba zrozumieć, dlaczego biorąc pod uwagę cały postęp technologii obliczeniowych, programista w dalszym ciągu musi czynić starania, aby dla napisanego kodu uzyskać odpowiednią wydajność. Ponadto konieczne jest uświadomienie sobie, dlaczego sukces wymaga dogłębnego zrozumienia sprzętu informatycznego, języka programowania i możliwości kompilatora. Celem rozdziału jest wyjaśnienie, dlaczego wszystko to jest obecnie nadal konieczne.

W rozdziale jest mowa o powodach, dla których dbamy o wydajność programów. W szczególności są to powody, które sprawiają, że dobra wydajność *tak po prostu nie występuje*. Dowiemy się, dlaczego do osiągnięcia najlepszej wydajności lub czasem nawet odpowiedniej wydajności ważne jest zrozumienie różnych czynników wpływających na wydajność, a także powodów określonego zachowania programu, niezależnie od tego, czy działa on szybko, czy wolno.

W rozdziale zostaną omówione następujące podstawowe zagadnienia:

- Dlaczego wydajność ma znaczenie?
- Dlaczego wydajność wymaga uwagi programisty?
- Co rozumiemy przez wydajność?
- W jaki sposób oceniać wydajność?
- Czym jest wysoka wydajność?

Dlaczego należy brać pod uwagę wydajność?

W początkach technologii obliczeniowych programowanie było utrudnione. Procesory były wolne, pamięć ograniczona, kompilatory prymitywne, a ponadto nic nie mogło zostać osiągnięte bez większego nakładu pracy. Programista musiał znać architekturę procesora i rozmieszczenie pamięci. Gdy kompilator nie obsługiwał jej poprawnie, kluczowy kod musiał być pisany w asemblerze.

Później sytuacja się poprawiła. Każdego roku procesory były coraz szybsze, a wielkość, która określała pojemność pokaźnego dysku twardego, stała się wielkością pamięci głównej przeciętnego komputera PC. Twórcy kompilatorów opanowali kilka sztuczek pozwalających przyspieszyć programy. Programiści mogli poświęcać więcej czasu na faktyczne rozwiązywanie problemów. Było to odzwierciedlane w językach programowania i stylach projektowania: w czasie, gdy pojawiały się języki wyższego poziomu oraz rozwijały się metody projektowania i programowania, punkt uwagi programistów przesunął się z tego, *co* chcieli zapisać w kodzie, na to, *w jaki sposób* chcieli to zapisać.

Obowiązująca wcześniej ogólna wiedza, taka jak dokładna liczba rejestrów procesora oraz ich nazwy, stała się osobliwa i tajemnicza. „Duża baza kodu" była czymś, co do udźwignięcia wymagało sporych nakładów. Obecnie coś takiego byłoby w zasięgu możliwości systemu kontroli wersji. Prawie w ogóle nie było potrzeby tworzenia kodu przeznaczonego dla konkretnego procesora lub systemu pamięci, a przenośny kod stał się normą.

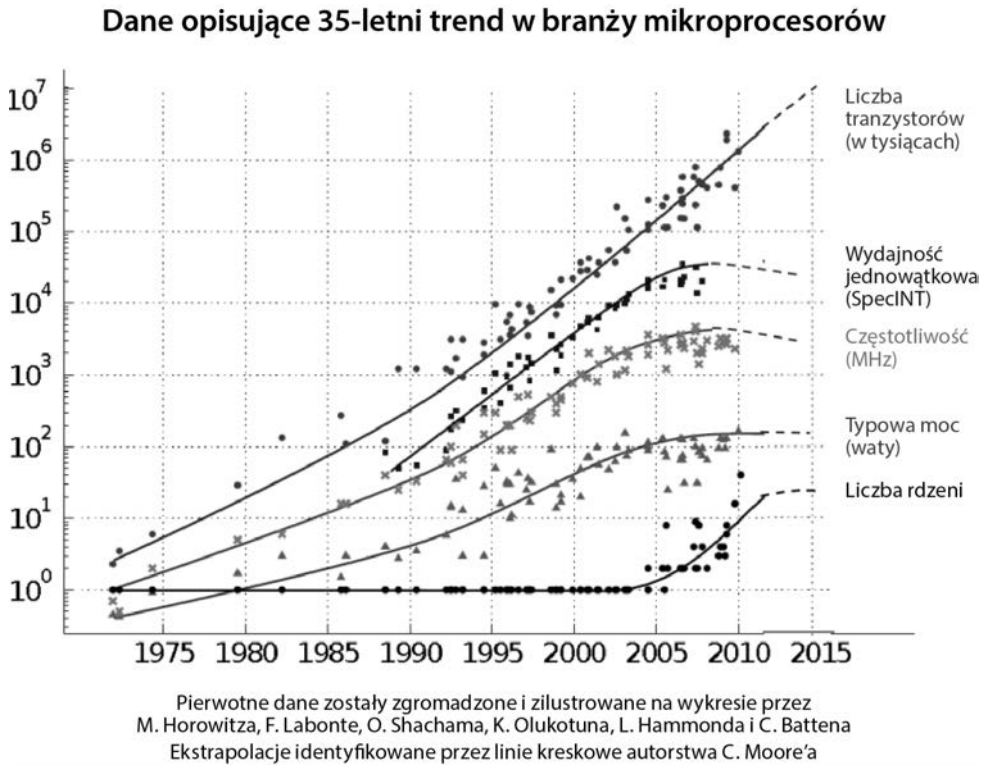
Jeśli chodzi o asembler, właściwie trudno było zdystansować kod generowany przez kompilator. Zadanie to było zdecydowanie poza zasięgiem większości programistów. W przypadku wielu aplikacji oraz osób tworzących je istniało pojęcie „wystarczającej wydajności". Ważniejsze stały się inne aspekty zawodu programistów (aby było jasne, to, że programiści mogli skupić się na czytelności swojego kodu bez martwienia się tym, czy dodanie funkcji z opisową nazwą spowodowałoby nieakceptowane spowolnienie działania programu, było czymś pozytywnym).

Później, a raczej nagle, skończył się „darmowy obiad" w rodzaju „wydajność dba sama o siebie". Pozornie niemożliwy do powstrzymania postęp w przypadku cały czas zwiększającej się mocy obliczeniowej po prostu się zakończył.

Około roku 2005 moc obliczeniowa pojedynczego procesora osiągnęła stan nasycenia. W dużej mierze miało to bezpośredni związek z częstotliwością procesora, która również przestała się zwiększać. Z kolei częstotliwość była ograniczona przez kilka czynników, z których jednym był pobór mocy (jeśli trend dotyczący częstotliwości byłby kontynuowany bez zmian, obecnie używane procesory mogłyby się pochwalić mocą obliczeniową przypadającą na milimetr kwadratowy większą niż znakomite silniki odrzutowe wynoszące rakiety w kosmos).

Rysunek 1.1 wyraźnie pokazuje, że w 2005 r. nie zostały zatrzymane wszystkie wskaźniki pomiarowe postępu: nadal zwiększała się liczba tranzystorów umieszczonych w obrębie jednego

układu. Czy zatem nie zapewniało to szybszych procesorów? Odpowiedź na to pytanie jest dwojaka. Po części ujawnia ją krzywa widoczna na dole wykresu. Zamiast zapewniać większy pojedynczy procesor, projektanci byli zmuszeni do podjęcia decyzji o umieszczeniu wewnątrz tego samego układu kilku rdzeni procesorowych. Oczywiście, sumaryczna moc obliczeniowa wszystkich rdzeni zwiększała się wraz z ich liczbą, ale tylko wtedy, gdy programista wiedział, jak z nich skorzystać. Inna część „wielkiej tajemnicy tranzystorów" (dokąd te wszystkie tranzystory zmierzają?) dotyczy tego, że poddano je różnym bardzo zaawansowanym ulepszeniom pod kątem możliwości procesorów. Ulepszenia te mogą zostać wykorzystane do zwiększenia wydajności, ale znów tylko w sytuacji, gdy programista poczyni starania, aby ich użyć.



Rysunek 1.1. Wykres prezentujący 35-letni rozwój mikroprocesorów (sprawdź zasoby dostępne pod adresami https://github.com/karlrupp/microprocessor-trend-data i *https://github.com/karlrupp/microprocessor-trend-data/blob/master/LICENSE.txt*)

Zmiana w rozwoju procesorów przedstawiona na powyższym rysunku jest często utrzymywana jako powód, dla którego rozpowszechniło się programowanie współbieżne. Zmiana była jednak jeszcze bardziej gruntowna. Z lektury książki można się będzie dowiedzieć, że w celu uzyskania najlepszej wydajności programista ponownie musi zrozumieć niuanse architektury procesora i pamięci oraz zachodzące między nimi interakcje. Znakomita wydajność już nie „występuje tak po prostu". Jednocześnie nie zrezygnowano z postępu dokonanego w dziedzinie pisania kodu, w którym wyraźnie jest określane nie to, co musi zostać zrealizowane, lecz w jaki sposób.

W dalszym ciągu ma być tworzony czytelny kod możliwy do utrzymania, który ma (*ma*, a nie *może*) być też wydajny.

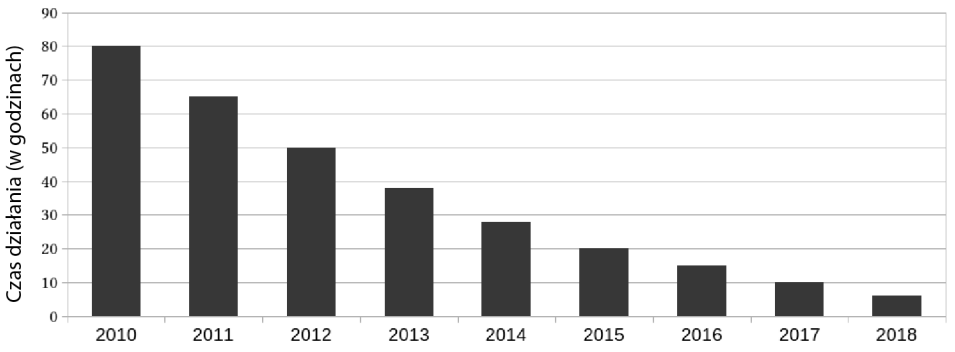
Bez wątpienia w przypadku wielu zastosowań w dalszym ciągu nowoczesne procesory zapewniają *wystarczającą wydajność*, ale zwraca ona większą uwagę niż wcześniej. W dużej mierze wynika to ze zmian w procesie projektowania procesorów, o których właśnie wspomniano. Ponadto dąży się do realizowania większej liczby obliczeń w większej liczbie aplikacji, które niekoniecznie mają dostęp do najlepszych zasobów obliczeniowych (na przykład przenośne urządzenie medyczne może obecnie być wyposażone w pełną wersję sieci neuronowych).

Na szczęście nie ma potrzeby ponownego odkrywania części z *zagubionej sztuki zapewniania wydajności*, przeglądając sterty kart perforowanych znajdujących się w mrocznym magazynie. W każdym momencie pojawiały się problemy nadal trudne do rozwiązania, a stwierdzenie „nigdy nie ma wystarczającej mocy obliczeniowej" było prawdziwe w przypadku wielu programistów. Wraz ze zwiększaniem mocy obliczeniowej w tempie wykładniczym tak samo szybko rosło zapotrzebowanie na nią. Sztuka zapewniania *ekstremalnej wydajności* pozostała żywa w tych kilku obszarach, w których była potrzebna. Przykład takiego obszaru może na tym etapie okazać się pouczający i inspirujący.

Dlaczego wydajność ma znaczenie?

Aby znaleźć przykład obszaru, w przypadku którego poziom zwracania uwagi na wydajność nigdy tak naprawdę się nie zmniejszył, przeanalizujmy rozwój technologii umożliwiających same obliczenia. Mowa o narzędziach **EDA** (***Electronic Design Automation***) automatyzujących projektowanie układów elektronicznych, które służą do projektowania samych komputerów.

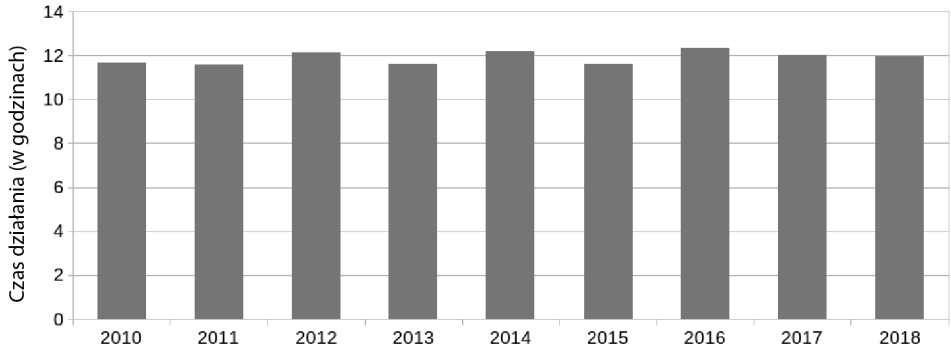
Jeśli pod uwagę weźmie się obliczenia z roku 2010 związane z projektowaniem, symulacją lub weryfikowaniem konkretnego mikroukładu i w każdym kolejnym roku uruchomi się obliczenia, powodujące identyczne obciążenie, uzyska się wyniki podobne do zaprezentowanych na rysunku 1.2.



Rysunek 1.2. Czas przetwarzania (w godzinach) uzyskany w kolejnych latach dla konkretnych obliczeń przeprowadzanych przez narzędzie EDA

To, co w 2010 r. wymagało 80 godzin do przeprowadzenia obliczeń, w 2018 r. zajęło mniej niż 10 godzin (a obecnie jeszcze mniej). Z czego wynika taki postęp? Z kilku powodów jednocześnie: po części z tego, że komputery stały się szybsze, ale też z tego, że oprogramowanie jest wydajniejsze, opracowano lepsze algorytmy, a kompilatory optymalizujące stały się efektywniejsze.

Niestety, w 2021 r. nie są produkowane mikroukłady w wersji z roku 2010: zrozumiałe jest, że wraz z tym, jak komputery stają się wydajniejsze, tworzenie nowszych i lepszych staje się trudniejsze. Stąd też bardziej interesującą kwestią jest to, ile czasu zajmuje zrealizowanie tych samych działań każdego roku w przypadku nowego mikroukładu wyprodukowanego w danym roku (rysunek 1.3).

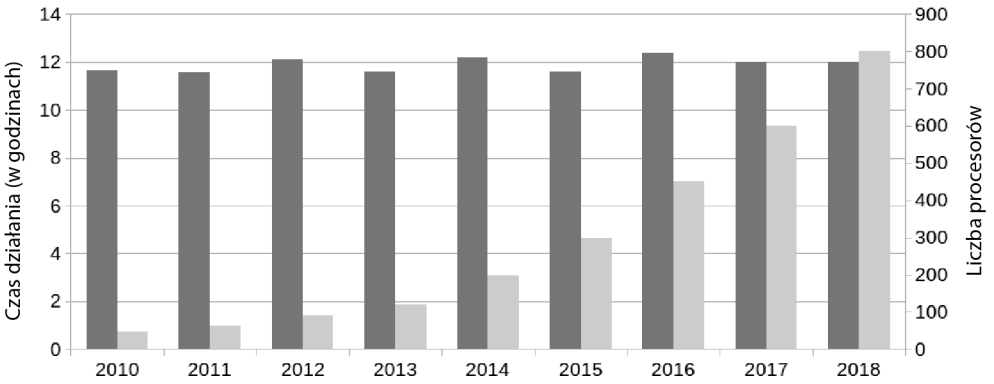


Rysunek 1.3. Czas działania (w godzinach) uzyskany co roku dla konkretnego zadania projektowego w przypadku najnowszego mikroukładu

Faktyczne obliczenia realizowane każdego roku nie są identyczne, ale służą temu samemu celowi. W przypadku najnowszego i najwspanialszego procesora wyprodukowanego co roku może to być na przykład *sprawdzenie, czy układ działa zgodnie z zamierzeniami*. Na zaprezentowanym wykresie widać, że najwydajniejsze procesory bieżącej generacji uruchamiające najlepsze dostępne narzędzia wymagają mniej więcej tyle samo czasu każdego roku do zaprojektowania i opracowania modelu procesora następnej generacji. Tempo się utrzymuje, ale nie są robione żadne postępy.

Prawda jest jednak jeszcze gorsza, a powyższy wykres nie przedstawia wszystkiego. Faktem jest, że najlepszy procesor, który został wyprodukowany co roku w okresie od 2010 r. do 2018 r., mógł zostać sprawdzony w ciągu jednej nocy (przez jakieś 12 godzin) za pomocą komputera wyposażonego w najwydajniejsze procesory z poprzedniego roku. Zapomniano jednak zapytać o to, *ilu takich procesorów to dotyczyło*. Cóż, na rysunku 1.4 zilustrowano całą prawdę.

Co roku najwydajniejsze komputery wyposażone w coraz większą liczbę najnowszych i najwydajniejszych procesorów uruchamiających najnowsze wersje oprogramowania (zoptymalizowanego pod kątem wykorzystania coraz większej liczby procesorów i używającego każdego z nich w bardziej efektywny sposób) realizują działania niezbędne do wyprodukowania najwydajniejszych komputerów w kolejnym roku. Każdego roku zadanie to balansuje na krawędzi tego, co jest ledwie możliwe. To, że nie spada się z tej krawędzi, jest głównie zasługą inżynierów



Rysunek 1.4. Wykres z rysunku 1.3 poszerzony o liczbę procesorów użytych do wykonania każdego obliczenia

zajmujących się sprzętem i oprogramowaniem. Pierwsi z nich zapewniają coraz większą moc obliczeniową, a drudzy korzystają z niej maksymalnie efektywnie. Książka ułatwi zaznajomienie się z umiejętnościami drugiej grupy inżynierów.

Uświadomiono już doniosłość tematyki książki. Zanim zagłębimy się w związane z tym szczegóły, pomocne będzie dokonanie ogólnego przeglądu. Można powiedzieć, że jest to analiza mapy terytorium, na którym zostanie rozpoczęta kampania eksploracyjna.

Co rozumiemy przez wydajność?

Przybliżono kwestię wydajności programów. Wspomniano o bardzo wydajnym oprogramowaniu. Co jednak mamy na myśli, mówiąc o wydajności? Intuicyjnie rozumiemy, że program o dużej wydajności jest szybszy od programu mającego kiepską wydajność. Nie oznacza to jednak, że szybszy program zawsze zapewnia *dobrą* wydajność (obie kategorie programów mogą cechować się kiepską wydajnością).

Wspomniano również o efektywnych programach. Czy jednak efektywność jest tym samym co duża wydajność? Choć efektywność jest *powiązana* z wydajnością, nie jest dokładnie tym samym. Efektywność oznacza optymalne korzystanie z zasobów, a nie marnowanie ich. Efektywny program odpowiednio wykorzystuje sprzęt do obliczeń.

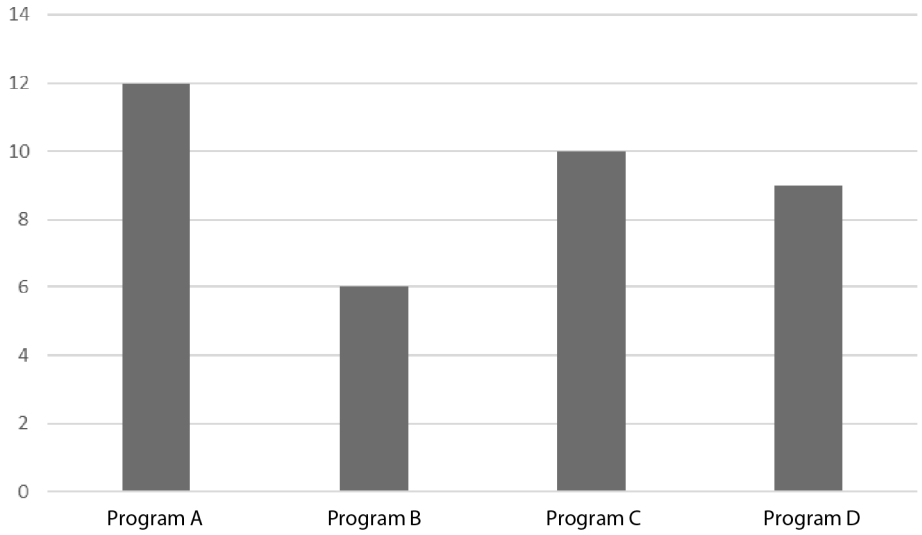
Z jednej strony efektywny program nie pozostawia dostępnych zasobów w stanie bezczynności. Jeśli istnieją obliczenia wymagające przeprowadzenia oraz procesor, który nie jest niczym zajęty, powinien on wykonać kod oczekujący na to. Idea ta sięga dalej: procesory oferują wiele zasobów obliczeniowych, a efektywny program próbuje jednocześnie skorzystać z ich jak największej liczby. Z drugiej strony efektywny program nie marnuje zasobów na realizowanie niepotrzebnych działań. Nie są przez niego wykonywane obliczenia, które nie są wymagane, nie marnuje pamięci na przechowywanie danych, które nigdy nie zostaną użyte,

nie przesyła danych za pośrednictwem sieci, jeśli nie jest to konieczne itd. Podsumowując, efektywny program nie pozostawia dostępnego sprzętu w stanie bezczynności ani nie wykonuje żadnych działań, które nie muszą być realizowane.

Z kolei wydajność zawsze jest powiązana z jakimiś wskaźnikami pomiarowymi. Najczęściej jest to „szybkość" lub określenie tego, jak szybki jest program. Bardziej rygorystyczną metodą definiowania takiego wskaźnika pomiarowego jest przepustowość, czyli liczba obliczeń wykonywanych przez program w danym czasie. Odwrotnym wskaźnikiem pomiarowym często używanym w tym samym celu jest czas realizacji lub czas, jaki jest niezbędny do obliczenia konkretnego wyniku. Nie jest to jednak jedyna możliwa definicja wydajności.

Wydajność jako przepustowość

Rozważmy cztery programy używające różnych implementacji do obliczenia tego samego końcowego wyniku. Na poniższym rysunku 1.5 pokazano czasy działania wszystkich czterech programów (podane jednostki są względne; faktyczne wartości nie są istotne, gdyż interesuje nas wydajność względna):



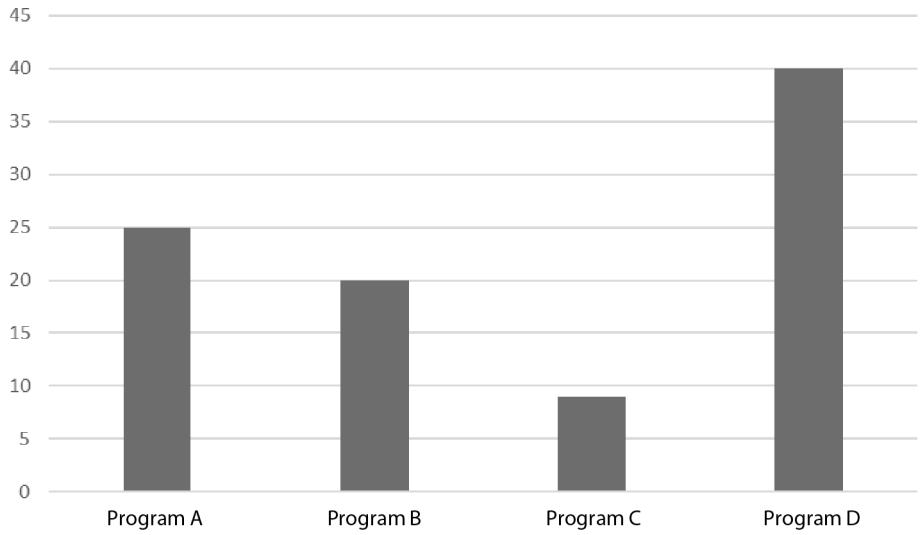
Rysunek 1.5. Czasy działania czterech różnych implementacji jednego algorytmu (użyto jednostek względnych)

Wydaje się oczywiste, że program B ma największą wydajność: zakończył pracę przed trzema innymi programami. Dla porównania wyznaczenie tego samego wyniku najwolniejszemu programowi zajęło dwa razy więcej czasu. W wielu sytuacjach byłyby to wszystkie dane niezbędne do wybrania najlepszej implementacji.

Istotny jest jednak kontekst problemu. Nie wspomniano, że przykładowy program działa na urządzeniu zasilanym baterią, takim jak telefon komórkowy, dlatego znaczenie ma również pobór mocy.

Wydajność jako pobór mocy

Na poniższym rysunku 1.6 zilustrowano moc wykorzystywaną przez wszystkie cztery programy w trakcie trwania obliczeń.



Rysunek 1.6. Pobór mocy dla czterech różnych implementacji jednego algorytmu (użyto jednostek względnych)

Pomimo dłuższego czasu potrzebnego na uzyskanie wyniku program C zużył ogólnie mniej mocy. Który zatem program ma najlepszą wydajność?

I tym razem jest to trudne pytanie, gdy nie jest znany pełny kontekst. Program, który służy do przetwarzania danych dźwiękowych, nie tylko działa na urządzeniu przenośnym, ale też wykonuje obliczenia w czasie rzeczywistym. Czy nie powinno to powodować stawiania wyżej kwestii szybszego uzyskania danych w czasie rzeczywistym? Nie do końca.

Wydajność w przypadku aplikacji czasu rzeczywistego

Program czasu rzeczywistego musi cały czas być na bieżąco z przetwarzanymi przez siebie zdarzeniami. Procesor dźwiękowy musi w szczególności nadążać za danymi generowanymi w wyniku mowy. Jeśli program może przetwarzać dane dźwiękowe 10 razy szybciej, niż ktoś potrafi mówić, nie zapewni żadnych korzyści. Uwagę można też zwrócić na kwestię poboru mocy.

Z kolei, jeśli program sporadycznie nie nadąża, mogą zostać zgubione jakieś dźwięki lub nawet słowa. Sugeruje to, że czas rzeczywisty lub szybkość ma znaczenie do pewnego stopnia, ale muszą być zapewniane w przewidywalny sposób.

W odniesieniu do tego udostępniono oczywiście wskaźnik pomiaru wydajności, czyli opóźnienie końcowe. W tym przypadku jest to opóźnienie między czasem przygotowania danych (zarejestrowanie głosu) i czasem zakończenia przetwarzania. Wspomniany wcześniej wskaźnik pomiarowy w postaci przepustowości odzwierciedla średni czas przetwarzania dźwięku: jeśli rozmawiamy przez telefon przez godzinę, to ile czasu zajmie procesorowi dźwiękowemu wykonanie wszystkich niezbędnych obliczeń? W tym kontekście tak naprawdę znaczenie ma jednak to, że każde drobne obliczenie realizowane dla każdego dźwięku kończy się na czas.

Na niższym poziomie szybkość obliczeń zmienia się: czasami są one kończone szybciej, a czasami zajmują więcej czasu. Dopóki średnia szybkość jest akceptowalna, istotne są rzadko występujące długie opóźnienia.

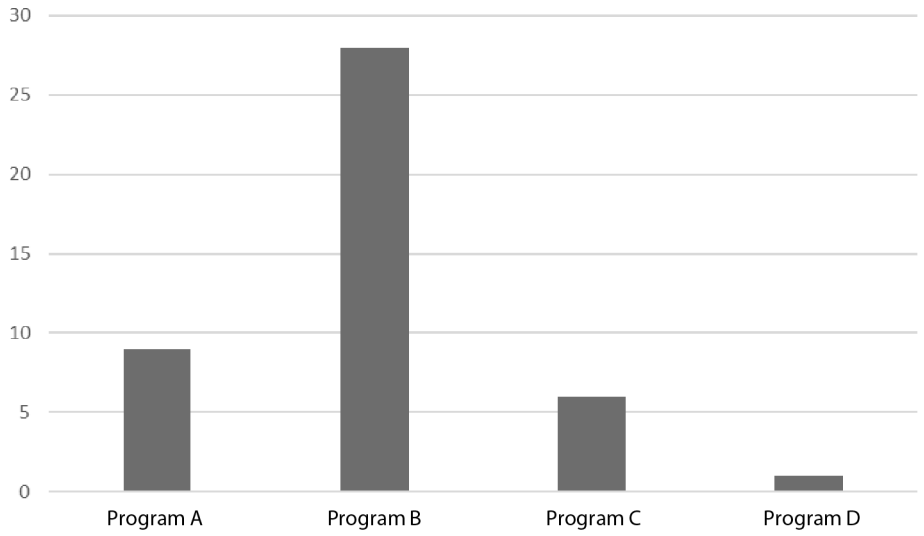
Wskaźnik pomiarowy w postaci opóźnienia końcowego jest obliczany jako konkretny centyl opóźnienia (na przykład 95. centyl). Jeśli t to czas z opóźnieniem 95. centyla, 95% wszystkich obliczeń zajmuje czas krótszy niż t. Sam wskaźnik pomiarowy jest stosunkiem czasu t z opóźnieniem 95. centyla do średniego czasu obliczeń t_0 (często jest to też wyrażane jako wartość procentowa, dlatego opóźnienie wynoszące 30% dla 95. centyla oznacza, że czas t jest 30% dłuższy niż czas t_0).

Na rysunku 1.7 widać, że **program B**, który oblicza wyniki szybciej niż wszystkie pozostałe implementacje, średnio zapewnia też najbardziej nieprzewidywalne czasy działania. Z kolei **program D**, który dotąd nie wyróżniał się, wykonuje obliczenia jak mechanizm zegarowy i za każdym razem potrzebuje w zasadzie tyle samo czasu do przeprowadzenia danego obliczenia. Jak już zostało to zaobserwowane, program D cechuje się też najgorszym poborem mocy. Niestety, jest to częsta sytuacja, ponieważ techniki zapewniające większą efektywność programu pod względem poboru mocy są z natury probabilistyczne: przeważnie przyspieszają obliczenia, ale nie za każdym razem.

Który zatem program jest najlepszy? Oczywiście, odpowiedź zależy od zastosowania, a nawet wtedy może nie być oczywista.

Wydajność w zależności od kontekstu

Jeśli byłoby to oprogramowanie symulujące działające w dużym centrum danych, które do wykonania obliczeń potrzebuje wielu dni, przepustowość byłaby najważniejsza. W przypadku urządzenia zasilanego baterią największe znaczenie ma przeważnie pobór mocy. W bardziej złożonym środowisku, takim jak przykładowy procesor dźwiękowy czasu rzeczywistego, jest to kombinacja wielu czynników. Oczywiście, średni czas działania ma znaczenie, ale tylko do momentu, aż uzyska się stan „wystarczająco szybkiego”. Jeśli głośnik nie jest w stanie wychwycić



Rysunek 1.7. Opóźnienie 95-procentowe czterech różnych implementacji tego samego algorytmu (podano w procentach)

opóźnień, zapewnianie jeszcze większej szybkości nie da żadnych korzyści. Istotne jest opóźnienie końcowe: użytkownicy nie znoszą, gdy od czasu do czasu ginie słowo w całej konwersacji. Jeśli opóźnienie jest na tyle dobre, że jakość połączenia telefonicznego jest ograniczona przez inne czynniki, dalsze poprawianie jej zapewni bardzo małe korzyści. Na tym etapie lepsze będzie ograniczenie poboru mocy.

Zrozumiałe już jest, że w przeciwieństwie do efektywności, wydajność jest zawsze definiowana w odniesieniu do konkretnych wskaźników pomiarowych, które zależą od zastosowania i rozwiązywanego problemu. W przypadku niektórych wskaźników pomiarowych pojawia się coś takiego jak „wystarczająco dobra", natomiast inne wskaźniki wychodzą na pierwszy plan. Efektywność, która odzwierciedla wykorzystanie zasobów obliczeniowych, to jeden ze sposobów osiągnięcia dobrej wydajności. Być może jest to najczęstszy sposób, lecz nie jedyny.

Ocenianie, szacowanie i przewidywanie wydajności

Jak właśnie wskazano, pojęcie wskaźników pomiarowych jest fundamentalne z punktu widzenia zagadnienia wydajności. W przypadku wskaźników pomiarowych zawsze istnieje domniemana możliwość i potrzeba pomiarów. Stwierdzenie, że dysponuje się wskaźnikiem pomiarowym, oznacza, że uzyskano sposób określenia ilości i zmierzenia czegoś. Jedynym sposobem ustalenia wartości wskaźnika pomiarowego jest dokonanie pomiaru.

Nie można przesadzić z ważnością pomiaru wydajności. Często mawia się, że w przypadku wydajności pierwszą regułą jest to, aby nigdy nie przypuszczać. Następny rozdział książki poświęcono pomiarom wydajności, narzędziom pomiarowym, sposobom korzystania z nich oraz metodom interpretowania wyników.

Przypuszczanie odnośnie do wydajności jest niestety zbyt wszechobecne. Stąd też pojawiają się takie przesadnie ogólnikowe stwierdzenia jak „unikaj stosowania funkcji wirtualnych w języku C++, gdyż są one wolne". Problem z takimi stwierdzeniami nie polega na tym, że są one nieprecyzyjne, lecz na tym, że nie odwołują się do wskaźnika pomiarowego określającego, o ile wolniejsza jest funkcja wirtualna w porównaniu z funkcją, która nie jest wirtualna. W ramach ćwiczenia do wyboru jest kilka następujących odpowiedzi, z których każda zawiera określenie ilościowe:

- Funkcja wirtualna jest 100% wolniejsza.
- Funkcja wirtualna jest około 15 – 20% wolniejsza.
- Funkcja wirtualna jest pomijalnie wolniejsza.
- Funkcja wirtualna jest 10 – 20% szybsza.
- Funkcja wirtualna jest 100 razy wolniejsza.

Która odpowiedź jest poprawna? Jeśli wybrałeś dowolną z tych odpowiedzi, gratulacje, ponieważ zdecydowałeś o wyborze poprawnej odpowiedzi. Zgadza się, każda z tych odpowiedzi jest poprawna w określonych okolicznościach i w konkretnym kontekście (aby dowiedzieć się, dlaczego tak jest, Czytelnik będzie musiał poczekać aż do rozpoczęcia lektury rozdziału 9.).

Niestety, akceptując fakt, że prawie niemożliwe jest przypuszczanie odnośnie do wydajności, ryzykuje się wpadnięciem w kolejną pułapkę, jaką jest używanie tego jako wymówki do tworzenia nieefektywnego kodu „przewidzianego do późniejszego optymalizowania", gdyż *nie formułuje się przypuszczeń w przypadku wydajności*. Choć prawdziwa, ta ostatnia maksyma może zostać wykorzystana zbyt przesadnie, dokładnie tak jak popularne powiedzenie „nie optymalizuj przedwcześnie".

Wydajność nie może być uwzględniana w programie później, dlatego nie powinna być jedynie dodatkiem w trakcie początkowych prac projektowych. Kwestie wydajności i związane z nią cele mają swoje miejsce na etapie projektowania tak jak inne cele projektowe. Pojawia się pewne napięcie między tymi wstępnymi celami powiązanymi z wydajnością i zasadą, aby nigdy nie przypuszczać odnośnie do wydajności. Konieczne jest znalezienie właściwego kompromisu. Dobry sposób na opisanie tego, co naprawdę ma zostać osiągnięte na etapie projektowania w odniesieniu do wydajności, jest następujący: choć prawie niemożliwe jest przewidzenie z góry najlepszych optymalizacji, możliwe jest zidentyfikowanie decyzji projektowych, które spowodowałyby, że dalsze optymalizacje byłyby bardzo utrudnione, a nawet niewykonalne.

To samo dotyczy późniejszego etapu opracowywania programu: niemądre jest spędzanie wielu godzin na optymalizowaniu funkcji, która ostatecznie jest wywoływana raz dziennie i działa zaledwie przez sekundę. Z kolei bardzo mądrym działaniem jest hermetyzowanie najpierw takiego kodu wewnątrz funkcji. Dzięki temu, jeśli w trakcie rozwoju programu zmienią się wzorce wykorzystania, kod *może* być później optymalizowany bez potrzeby przebudowywania reszty programu.

Innym sposobem opisywania ograniczeń zasady „nie optymalizuj przedwcześnie" jest uściślanie jej w postaci stwierdzenia „tak, ale też celowo nie stosuj pesymizacji". Rozpoznanie różnicy między tymi dwoma stwierdzeniami wymaga znajomości właściwych metod projektowania, a także zrozumienia różnych aspektów programowania pod kątem zapewnienia dużej wydajności.

A zatem co jako projektant lub programista trzeba poznać i zrozumieć, aby stać się biegłym w sztuce projektowania aplikacji o dużej wydajności? W następnym podrozdziale zaczniemy od skróconej listy związanych z tym celów, a w dalszej kolejności zajmiemy się ich szczegółowym omówieniem.

Poznawanie zagadnienia dużej wydajności

Co sprawia, że program ma dużą wydajność? Można by stwierdzić, że „efektywność", ale po pierwsze nie zawsze jest to prawdą (choć często jest), a po drugie po prostu nie zapewnia odpowiedzi na to pytanie, ponieważ na myśl przychodzi kolejne oczywiste pytanie: „Dobrze, a co powoduje, że program jest efektywny?". Co musimy wiedzieć, aby tworzyć efektywne lub bardzo wydajne programy? Sformułujmy ogólną listę wymaganych umiejętności i wiedzy:

- Wybór właściwego algorytmu.
- Efektywne korzystanie z zasobów procesorów.
- Efektywne używanie pamięci.
- Unikanie zbędnych obliczeń.
- Efektywne stosowanie współbieżności i wielowątkowości.
- Efektywne posługiwanie się językiem programowania przez unikanie nieefektywnych elementów.
- Pomiar wydajności i interpretowanie wyników.

Na drodze do osiągnięcia dużej wydajności najważniejszym czynnikiem jest wybór dobrego algorytmu. Nie można „naprawić" złego algorytmu przez optymalizowanie jego implementacji. Jest to też jednak jeden czynnik, którego omawianie wykracza poza zakres książki. Algorytmy są powiązane z problemami, a ta książka nie prezentuje algorytmów. Konieczne będzie poszukanie we własnym zakresie najlepszych algorytmów w przypadku problemu, z którym ma się do czynienia.

Z kolei metody i techniki pozwalające uzyskać dużą wydajność są przeważnie niezależne od problemu. Oczywiście, zależą one od wskaźników pomiaru wydajności. Na przykład optymalizacja systemów czasu rzeczywistego to bardzo specyficzna dziedzina z wieloma osobliwymi problemami. W książce skupiono się głównie na wskaźnikach pomiaru wydajności w odniesieniu do obliczeń z dużą wydajnością, czyli realizowania wielu obliczeń tak szybko, jak to możliwe.

Aby się to powiodło, niezbędne jest opanowanie umiejętności korzystania w jak największym zakresie z dostępnego sprzętu komputerowego. Cel ten zawiera element przestrzenny

i czasowy. Jeżeli chodzi o przestrzeń, mowa o korzystaniu z większej liczby tranzystorów, które procesor zawiera w ogromnych ilościach. Jeśli już nie szybsze, procesory stają się większe. Do czego jest używany dodatkowy obszar? Prawdopodobnie są w nim zapewniane nowe możliwości wykonywania obliczeń, z których można skorzystać. Nawiązując do czasu, należy rozumieć przez to, że każdorazowo powinno się używać jak największej ilości dostępnego sprzętu. W każdym razie zasoby obliczeniowe są bezużyteczne, jeśli pozostają bezczynne, dlatego celem jest zapobieganie temu. Jednocześnie bezproduktywne działania nie opłacają się, dlatego wskazane jest unikanie robienia czegokolwiek, co nie jest absolutnie konieczne. Nie jest to tak oczywiste, jak wygląda. Istnieje mnóstwo subtelnych sposobów, jakie mogą być wykorzystywane przez program do przeprowadzania obliczeń, które nie są wymagane.

W książce omawianie zaczniemy od pojedynczego procesora i dowiemy się, jak efektywnie używać jego zasobów obliczeniowych. Dalej zdobyta wiedza zostanie poszerzona, tak aby nie dotyczyła tylko procesora, ale też jego pamięci. Oczywiście później przyjrzymy się kwestii jednoczesnego stosowania wielu procesorów.

Efektywne korzystanie ze sprzętu to jednak tylko jeden z niezbędnych składników jakości programu o dużej wydajności. Nie zapewni niczego dobrego efektywne realizowanie działań, których przede wszystkim można było uniknąć. Kluczem do tego, aby nie wykonywać zbędnych działań, jest efektywne używanie języka programowania, którym w naszym przypadku jest język C++ (większość wiedzy zdobytej na temat sprzętu może zostać wykorzystana w dowolnym języku, ale część technik optymalizowania języka jest bardzo ściśle powiązana z językiem C++). Co więcej, kompilatory są umieszczone między językiem używanym do pisania kodu i stosowanym sprzętem, dlatego niezbędne jest uzyskanie wiedzy na temat tego, jak za pomocą kompilatorów tworzyć najbardziej efektywny kod.

I wreszcie jedynym sposobem określenia poziomu sukcesu w przypadku dowolnego z właśnie przedstawionych celów jest dokonanie pomiaru: ile zasobów procesora jest używanych? Ile czasu mija w oczekiwaniu na pamięć? Jaki wzrost wydajności uzyskano w wyniku dodania kolejnego wątku? I tak dalej. Uzyskiwanie dobrych danych ilościowych dotyczących wydajności nie jest łatwe. Wymaga to dokładnego zrozumienia narzędzi pomiarowych. Interpretowanie wyników jest często jeszcze trudniejsze.

Dzięki tej książce można oczekiwać opanowania tych umiejętności. Objaśniono architekturę sprzętową, a także to, co kryje się za niektórymi elementami języka programowania. Ponadto przybliżono sposób pozwalający spojrzeć na kod tak, jak ma to miejsce w przypadku kompilatorów. Umiejętności te są ważne, ale jeszcze ważniejsze jest zrozumienie, dlaczego różne rzeczy działają tak, a nie inaczej. Sprzęt komputerowy zmienia się dość często, języki rozwijają się, a ponadto są opracowywane nowe algorytmy optymalizacji dla kompilatorów. A zatem konkretna wiedza powiązana z dowolnym z tych obszarów ma dość krótki żywot. Jeśli jednak znane są nie tylko najlepsze metody wykorzystania konkretnego procesora lub kompilatora, ale też sposoby, które pozwoliły zdobyć taką wiedzę, Czytelnik będzie dobrze przygotowany do powtórzenia procesu poznawania, a tym samym kontynuowania nauki.

Podsumowanie

W tym wprowadzającym rozdziale wyjaśniono, dlaczego zwiększa się zainteresowanie wydajnością i efektywnością oprogramowania pomimo szybkich postępów w zakresie samej mocy obliczeniowej nowoczesnych komputerów. Dokładniej rzecz ujmując, dowiedziałeś się, z jakiego powodu do zrozumienia czynników ograniczających wydajność i sposobów eliminowania ich niezbędny jest powrót do podstawowych elementów wykonywania obliczeń i zaznajomienie się z zasadami działania komputerów i programów na niskim poziomie. Obejmuje to zrozumienie sprzętu i efektywnego korzystania z niego, współbieżności, elementów języka C++ i optymalizacji kompilatora oraz ich wpływu na wydajność.

Taka wiedza niskopoziomowa z konieczności jest bardzo szczegółowa i specyficzna, ale dysponujemy planem na poradzenie sobie z tym. Gdy poznajemy konkretne informacje o procesorach lub kompilatorach, zaznajamiamy się również z procesem, który pozwolił uzyskać związane z tym wnioski. Oznacza to, że na swoim najniższym poziomie książka uczy o tym, jak się uczyć.

Dokładniej uświadomiono to, że pojęcie wydajności jest bezużyteczne bez zdefiniowania wskaźników, za pomocą których ją zmierzono. Potrzeba oceny wydajności w oparciu o konkretne wskaźniki pomiarowe wskazuje, że wszelkie działania związane z wydajnością bazują na danych i pomiarach. W związku z tym następny rozdział poświęcono zagadnieniu pomiaru wydajności.

Pytania

1. Dlaczego wydajność programów jest istotna pomimo postępów w zakresie mocy obliczeniowej?
2. Dlaczego zrozumienie kwestii wydajności oprogramowania wymaga wiedzy niskopoziomowej o sprzęcie komputerowym i językach programowania?
3. Jaka jest różnica między wydajnością i efektywnością?
4. Dlaczego wydajność musi być definiowana w odniesieniu do konkretnych wskaźników pomiarowych?
5. W jaki sposób można ocenić, czy zrealizowano cele powiązane z wydajnością w przypadku konkretnych wskaźników wydajności?

Dobre decyzje projektowe to klucz do wydajności kodu!

Jeszcze kilka lat temu, by zwiększyć szybkość działania programu, wystarczyło wymienić procesor. Dzisiejsze procesory nie są znacząco szybsze od poprzedników. Nowsze architektury zapewniają zaledwie niewielkie przyrosty wydajności istniejących już programów. Nie ma innego wyjścia: jeśli programista chce tworzyć efektywne oprogramowanie, musi wiedzieć, jak odpowiednio używać dostępnych zasobów obliczeniowych. Jest to trudna sztuka, wymagająca ciągłej gotowości do nauki.

To książka przeznaczona dla doświadczonych programistów, którzy chcą sprawnie tworzyć efektywny kod. Omówiono w niej, jak korzystać z zasobów procesora i pamięci, unikać zbędnych obliczeń, mierzyć wydajność kodu i prawidłowo stosować współbieżność i wielowątkowość. Zaprezentowano również zagadnienia związane z optymalizacjami przeprowadzanymi przez kompilator, a także metody efektywniejszego korzystania z właściwości języka programowania. Dokładnie wyjaśniono zasady projektowania oprogramowania pod kątem wydajności i aby ułatwić przyswajanie wiedzy, zamieszczono wiele przykładów, które będą przydatne w czasie samodzielnej nauki. Dzięki dogłębnemu zrozumieniu wiedzy ujętej w książce łatwiej będzie podejmować właściwe decyzje podczas projektowania nowego systemu lub modyfikowania już istniejącej architektury.

Najciekawsze zagadnienia:

- korzystanie ze sprzętowych zasobów obliczeniowych
- uporządkowanie pamięci i należyte zorganizowanie danych
- wydajność operacji a współbieżny dostęp
- stosowanie technik programowania bez użycia blokady
- zwiększanie efektywności optymalizacji za pomocą kompilatora
- interfejsy API dla współbieżnych struktur danych i struktur o dużej wydajności

Fedor G. Pikus jest uznanym ekspertem z zakresu obliczeń o dużej wydajności i programowania w języku C++, a także autorem książek, artykułów w czasopismach branżowych i właścicielem przeszło 25 patentów. Ma na koncie ponad 100 prac i prezentacji na konferencjach dotyczących fizyki, automatyzacji projektowania w elektronice, projektowania oprogramowania i języka C++.

Helion

KOD KORZYŚCI
Sięgnij po więcej!

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-9250-2

9 788328 392502

Cena: 89,00 zł

Packt