Rozdział 1, który mówi o tym, po co stosuje się obliczenia współbieżne, równoległe i rozproszone oraz czy można tego uniknąć

1.1 Wprowadzenie

Rozdział niniejszy wprowadza szereg pojęć wykorzystywanych w dalszych częściach skryptu. Przedstawia także szerszy kontekst, w którym funkcjonują obliczenia równoległe i rozproszone. Zaprezentowane są jedynie podstawowe idee, najważniejsze z nich, zgodnie z konwencją skryptu, zaznaczone drukiem pogrubionym. Więcej o historii i modelach przetwarzania równoległego można znaleźć w Rozdziale **??** omawiającym sprzęt do obliczeń równoległych oraz Dodatku **??**.

1.2 Czym jest przetwarzanie współbieżne, równoległe i rozproszone?

Przetwarzanie współbieżne, równoległe i rozproszone to powszechne dziś formy wykonania programów. Programem jest system operacyjny komputera, kompilator języka programowania, maszyna wirtualna interpretująca języki skryptowe czy dowolny program użytkowy realizujący rozmaite funkcje i zadania. Przedmiotem naszego zainteresowania będzie głównie ta ostatnia grupa – programy użytkowe.

Mówiąc o wykonaniu programu rozważamy realizację zbioru rozkazów i instrukcji, zmierzających do rozwiązania postawionego problemu obliczeniowego (rozkazami będziemy nazywali pojedyncze polecenia wykonywane przez procesory, instrukcjami bardziej złożone zadania, zapisywane w językach programowania i tłumaczone na szereg rozkazów procesora). Realizacja zbioru rozkazów odnosi się do procesu zachodzącego w czasie i nie jest jednoznacznie związana z kodem programu. Zbiór rozkazów oznacza zestaw wykonywany przy konkretnym uruchomieniu programu, dla konkretnych danych wejściowych. Każde wykonanie programu może odpowiadać innemu zbiorowi rozkazów (np. zależnie od danych wejściowych programu). Kod występującej w programie pętli zawierającej kilka instrukcji może być realizowany kilka tysięcy lub kilka milionów razy i wykonanie programu odnosić się będzie wtedy do realizacji zbioru złożonego odpowiednio z kilku tysięcy lub kilku milionów instrukcji.

Będziemy rozważać sytuację, kiedy zbiór rozkazów dzielony jest na podzbiory (na przykład po to, żeby każdy podzbiór uruchomić na innym procesorze). W ramach każdego podzbioru rozkazy wykonywane są w jednoznacznej kolejności określonej przez programistę lub przez kompilator tłumaczący kod źródłowy – takie wykonanie będziemy nazywać sekwencyjnym. **Sekwencyjnie wykonywany zbiór rozkazów będziemy nazywali wątkiem**[1]. **O wykonaniu współbieżnym dwóch wątków**

[1] W kolejnych rozdziałach, m.in. przy omawianiu wsparcia systemów operacyjnych dla obliczeń współbieżnych (rozdział

będziemy mówili wtedy, gdy rozkazy jednego wątku zaczną być wykonywane, zanim zakończy się wykonywanie rozkazów drugiego, uruchomionego wcześniej. Sytuację tę ilustrują rysunki 1.1 i 1.2. Na obu z nich, po lewej stronie, wzdłuż biegnącej pionowo osi czasu, widać kolejno wykonywane rozkazy wątku A uruchomionego wcześniej, po prawej znajdują się kolejne rozkazy z wykonywanego współbieżnie wątku B. Wątki A i B mogą być związane z tym samym programem, mogą też należeć do dwóch różnych programów.

Rysunek 1.1: Współbieżne wykonanie dwóch wątków – w przeplocie

Rysunek 1.2: Równoległe wykonanie dwóch wątków

O przetwarzaniu równoległym będziemy mówili wtedy, kiedy przynajmniej niektóre z rozkazów wątków wykonywanych współbieżnie są realizowane w tym samym czasie (jak to zobaczymy później, im więcej jednocześnie wykonywanych rozkazów, tym lepiej). Rys. 1.1 pokazuje sytuację, kiedy wątki wykonywane są współbieżnie, ale nie równolegle. Wykonywanie sekwencyjne wątku A jest przerywane (w terminologii systemów operacyjnych mówimy o wywłaszczeniu wątku), na pewien czas uruchamiane jest wykonywanie rozkazów wątku B, po czym system wraca do realizacji rozkazów wątku A. Takie wykonanie nazywane jest wykonaniem w przeplocie. Nie wymaga wielu jednostek wykonywania rozkazów, może zostać zrealizowane na pojedynczym procesorze, wystarczą odpowiednie możliwości systemu operacyjnego zarządzającego wykonywaniem wątków. Rys. 1.2 przedstawia wykonanie tych samych rozkazów w sposób równoległy. Takie wykonanie wymaga już specjalnego sprzętu, maszyny równoległej. Może to być maszyna z wieloma procesorami (rdzeniami) pracującymi pod kontrolą jednego systemu operacyjnego, może też być zbiór komputerów połączonych siecią, z których każdy posiada własny system operacyjny.

Wnioskiem z obu definicji jest stwierdzenie, że **każde wykonanie równoległe jest wykonaniem współbieżnym, natomiast nie każde wykonanie współbieżne jest wykonaniem równoległym**. Przetwarzanie współbieżne jest pojęciem bardziej ogólnym i ma też bardziej ugruntowaną pozycję w świecie informatyki niż przetwarzanie równoległe. Od wielu już lat każdy uruchomiony program na dowolnym komputerze jest wykonywany współbieżnie z innymi programami. Dzieje się tak za sprawą wielozadaniowych systemów operacyjnych, które zarządzają wykonaniem programów (każdy z popularnych systemów operacyjnych jest systemem wielozadaniowym). W tym wypadku każdy program jest osobnym zbiorem rozkazów i systemy operacyjne pozwalają na współbieżne wykonywanie tych zbiorów. Przeciętnie w jednym momencie typowy współczesny komputer ma uruchomione kilkadziesiąt do kilkuset programów (zdecydowana większość z nich przez zdecydowaną większość czasu przebywa w stanie uśpienia).

Przetwarzanie współbieżne jest od lat omawiane w ramach prezentacji systemów operacyjnych. Niniejszy skrypt nie zajmuje się zagadnieniami specyficznymi dla systemów operacyjnych. Jednak ze względu na to, że każdy program równoległy (i każdy system rozproszony) stosuje przetwarzanie współbieżne, szereg istotnych problemów współbieżności zostanie omówionych w dalszej części skryptu.

Z przetwarzaniem rozproszonym mamy do czynienia wtedy, gdy wątki, składające się na program, wykonywane są na różnych komputerach połączonych siecią[2]. Podobnie jak w przypadku

2), pojęcie wątku zostanie rozwinięte i uściślone

[2]Często, np. dla celów testowania, takie systemy rozproszone uruchamiane są na pojedynczym komputerze (który, jak wiemy, pozwala na współbieżne wykonanie programów). Istotą systemu rozproszonego pozostaje fakt, że **może** on zostać uruchomiony na różnych komputerach połączonych siecią.

wykonania równoległego, także **wykonanie rozproszone jest szczególnym przypadkiem przetwarzania współbieżnego**. Natomiast to, czy system rozproszony będzie wykonywany równolegle, zależy od organizacji obliczeń i pozostaje w gestii programisty. Często pojedynczy program może zostać zakwalifikowany i jako równoległy, i jako rozproszony. O ostatecznej klasyfikacji może zadecydować cel, dla którego zastosowano taką, a nie inną formę przetwarzania współbieżnego.

1.3 Po co przetwarzanie równoległe i rozproszone?

Istnieje kilka podstawowych celów, dla których stosuje się przetwarzanie równoległe i rozproszone. Pierwszym z nich jest zwiększenie wydajności obliczeń, czyli szybsze wykonywanie zadań przez sprzęt komputerowy. Realizowane zadania są rozmaite, zależą od konkretnego programu i konkretnej dziedziny zastosowania (przetwarzanie plików multimedialnych, dynamiczne tworzenie i wyświetlanie stron internetowych, przeprowadzanie symulacji zjawisk i procesów technicznych, analiza plików tekstowych itp.). W wykonywaniu zadań użytkowych może brać udział wiele urządzeń, takich jak monitory, karty sieciowe, twarde dyski. Zawsze jednak, jeśli podstawowym sprzętem przetwarzania jest komputer, mamy do czynienia z wykonywaniem zbiorów rozkazów i instrukcji. Zwiększenie wydajności przetwarzania będzie więc oznaczało realizację przewidzianych w programie rozkazów i instrukcji w krótszym czasie.

Ten cel – zwiększenie wydajności obliczeń i skrócenie czasu rozwiązania konkretnego pojedynczego zadania obliczeniowego – jest podstawowym celem stosowania przetwarzania równoległego. Intuicyjnie wydaje się naturalne, że mając do wykonania pewną liczbę rozkazów i dysponując możliwością uruchomienia dwóch wątków jednocześnie (czyli posiadając dwa procesory), możemy liczyć na zakończenie działania programu w czasie dwa razy krótszym niż w przypadku użycia tylko jednego wątku (procesora). Podobnie mając do dyspozycji cztery procesory chcielibyśmy zakończyć zadanie w czasie cztery razy krótszym, mając osiem w ośmiokrotnie krótszym itd. Kontynuując ten tok myślenia, możemy zadawać pytania:

- **jak maksymalnie skrócić czas wykonania danego programu?**
- **jak budować i wykorzystywać komputery o wielkiej liczbie procesorów?**
- **jakie teoretycznie największe korzyści możemy mieć z przetwarzania równoległego i rozproszonego?**

Tego typu pytania często stawiane są w dziedzinach związanych z obliczeniami wysokiej wydajności, wykorzystaniem superkomputerów, zadaniami wielkiej skali.

Zwiększanie wydajności obliczeń dzięki przetwarzaniu równoległemu ma także aspekt bardziej codzienny, związany ze znaczeniem praktycznym. Chodzi o to, **w jak wielu przypadkach i w jak dużym stopniu przeciętny użytkownik komputerów może uzyskać znaczące zyski z zastosowania obliczeń równoległych?** Odpowiedź na te pytania zależy od tego, jak wiele osób ma dostęp do sprzętu umożliwiającego efektywne przetwarzanie równoległe. Na potrzeby naszych rozważań przyjmiemy, że **przetwarzanie równoległe można uznać za efektywne, jeśli pozwala na co najmniej kilku-, kilkunastokrotne skrócenie czasu realizacji zadań w stosunku do przetwarzania sekwencyjnego**. Czy sprzęt dający takie możliwości jest powszechnie używany?

W ostatnich latach dostępność wysoko efektywnego sprzętu równoległego znacznie się zwiększyła i, jak wiele na to wskazuje, kolejne lata będą przynosiły dalszy postęp. Kilka, kilkanaście lat temu sprzęt równoległy był na tyle drogi, że wykorzystywały go głównie instytucje rządowe, wielkie firmy i centra naukowe. **Pierwszym krokiem na drodze do upowszechnienia sprzętu równoległego było wprowadzenie klastrów**, zespołów komputerów osobistych połączonych siecią, relatywnie tanich w

porównaniu z komputerami masowo wieloprocesorowymi[3] i dających porównywalne zyski czasowe. Obecnie klastry stają się standardowym wyposażeniem także w małych i średnich firmach, znacząco rośnie liczba korzystających z nich osób. Takie małe klastry zawierają kilkanaście, kilkadziesiąt procesorów i pozwalają na przyspieszanie obliczeń (skrócenie czasu wykonywania zadań) w podobnym zakresie – kilkanaście, kilkadziesiąt razy. Na bardzo zbliżonych zasadach konstrukcyjnych i programowych opierają się wielkie superkomputery pozwalające na uzyskiwanie przyspieszeń rzędu dziesiątek i setek tysięcy.

Drugim, znacznie ważniejszym z punktu widzenia popularyzacji obliczeń równoległych, procesem zachodzącym w ostatnich latach jest zmiana architektury mikroprocesorów.

Krótkie przypomnienie – architektura von Neumanna. Sposób przetwarzania rozkazów przez procesory, tak dawne jak i współczesne, najlepiej charakteryzowany jest przez model maszyny von Neumanna. Wymyślony w latach 40-tych XX wieku, stał się podstawą konstrukcji pierwszych komercyjnie sprzedawanych komputerów i do dziś jest podstawą budowy wszelkich procesorów oraz stanowi fundament rozumienia metod przetwarzania rozkazów we wszystkich komputerach. Konkretny sposób realizacji rozkazów przez procesory z biegiem lat stawał się coraz bardziej złożony[4], jednak jego istota nie zmieniła się i pozostaje taka sama jak w pierwotnej architekturze von Neumanna.

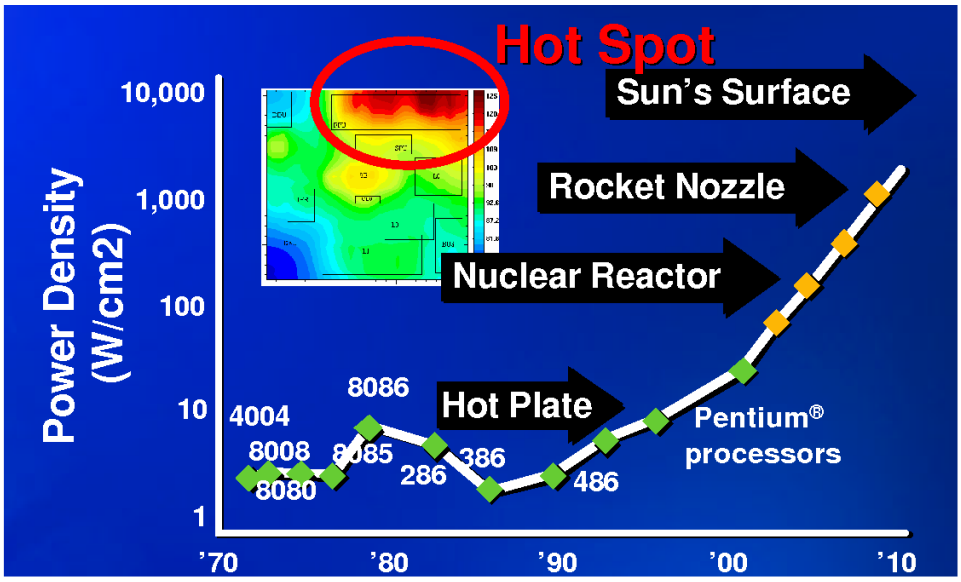
Istotą maszyny von Neumanna, odróżniającą ja od innych współczesnych jej modeli przetwarzania rozkazów, jest wykorzystanie jednej pamięci operacyjnej, w której znajdują się zakodowane (obecnie zawsze w sposób binarny) i kod programu, i dane na których operuje kod. Pamięć operacyjna składa się z komórek przechowujących dane, do których dostęp możliwy jest poprzez jednoznaczny adres, jaki posiada konkretna komórka. Adres jest dodatnią liczbą całkowitą, złożoną z takiej liczby bitów, którą wygodnie jest przetwarzać procesorowi. W miarę rozwoju procesorów długość tej liczby rosła (jest to związane m.in. z liczba bitów podstawowych rejestrów procesora i z szerokością magistrali łączącej procesor z pamięcią), zaczynając od kilku bitów, aż po dominujące dziś procesory 64-bitowe.

Wykonanie programu to przetwarzanie rozkazów zapisanych w pamięci operacyjnej komputera. Przetwarzanie pojedynczego rozkazu składa się z szeregu faz. Liczba faz zależy od konkretnego rozkazu, np. od tego czy operuje na danych pobieranych z pamięci, czy zapisuje wynik w pamięci itp. Kilka faz występuje we wszystkich rozkazach. Są to:

- pobranie rozkazu z pamięci (ang. *fetch*)
- dekodowanie rozkazu (ang. *decode*)
- wykonanie rozkazu (ang. *execute*)
- pewna forma zapisu efektu realizacji rozkazu (ang. *write back*)

W przypadku rozkazów operujących na danych z pamięci, dochodzi do tego jeszcze dostęp do pamięci: pobranie argumentów lub ich zapis (zazwyczaj współczesne procesory nie wykonują rozkazów, które pobierałyby argumenty z pamięci **i** zapisywałyby wynik do pamięci). Jako wsparcie działania systemów operacyjnych pojawia się jeszcze faza sprawdzenia, czy nie wystąpiło przerwanie.

[3] Komputerami masowo wieloprocesorowymi (ang. *massively parallel processors, MPP*) nazywamy komputery wyposażone w dużą liczbę (co najmniej kilkaset) procesorów, posiadających własne, niezależne pamięci operacyjne i połączonych szybką, zaprojektowaną specjalnie na potrzeby danego komputera, siecią.



Rysunek 1.3: Gęstość wydzielania ciepła [źródło: Intel, 2001]

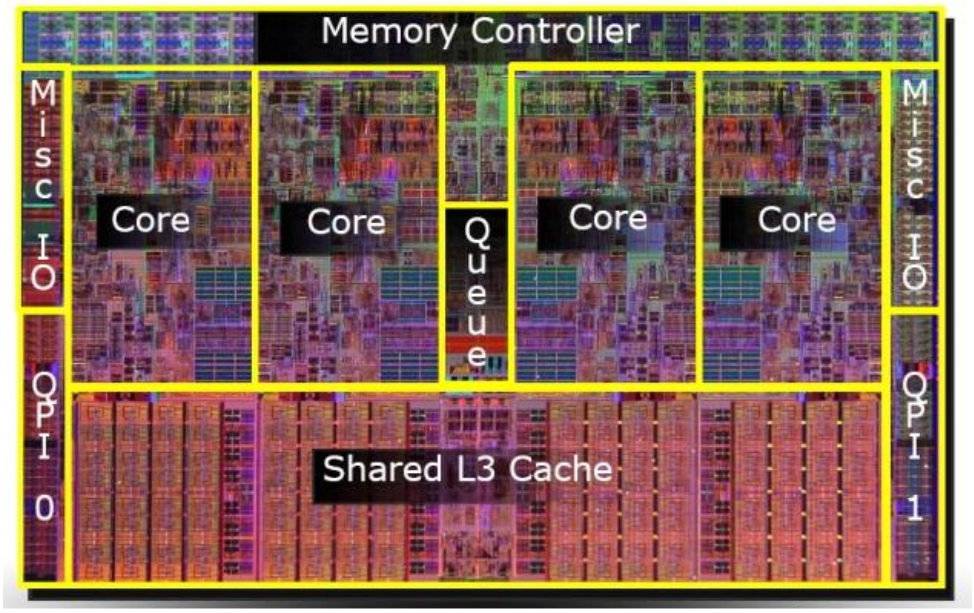
Zanim omówimy następstwa zmiany architektury mikroprocesorów w ostatnich latach, konieczne jest poczynienie kilku uwag odnośnie terminologii. Pierwotnie **procesorem nazywano jednostkę centralną komputera, która w sposób sekwencyjny wykonuje rozkazy pobierane z pamięci operacyjnej**. Tak też stosowane było to określenie dotychczas w skrypcie – procesor służył w pewnej konkretnej chwili do realizacji pojedynczego wątku.

W późniejszych latach nazwą procesor objęto także **mikroprocesory – pojedyncze układy scalone realizujące funkcje procesora**. Wprowadzone w latach 70-tych XX wieku, jako konkurencja dla układów stosujących obwody drukowane, są dziś jedyną formą procesorów. Będące początkowo prostymi układami zawierającymi kilka tysięcy tranzystorów, mikroprocesory w miarę upływu lat stawały się coraz bardziej złożone. Postęp elektroniki powodował, że rozmiar pojedynczego tranzystora w układzie scalonym stawał się coraz mniejszy, a ich liczba coraz większa. W pewnym momencie, w pojedynczym układzie scalonym zaczęto umieszczać, oprócz układów bezpośrednio realizujących przetwarzanie rozkazów, nazywanych rdzeniami mikroprocesora, także układy pamięci podręcznej czy układy komunikacji mikroprocesora ze światem zewnętrznym. Przez ponad trzydzieści lat rdzenie stawały się coraz bardziej złożone, a mikroprocesory coraz bardziej wydajne.

Na początku XXI-go wieku okazało się, że budowanie jeszcze bardziej złożonych rdzeni prowadzi do nadmiernego wydzielania ciepła przez mikroprocesory. Ilustruje to rys. 1.3, na którym porównane są gęstości wytwarzania ciepła procesorów i kilku wybranych urządzeń, takich jak płyta kuchenki elektrycznej, reaktor nuklearny i dysza silnika rakietowego, a także gęstość wydzielania ciepła na powierzchni słońca.

W konsekwencji nierozwiązania problemów z odprowadzaniem ciepła z coraz szybszych mikroprocesorów jednordzeniowych, producenci zdecydowali się na umieszczenie w pojedynczym układzie scalonym wielu rdzeni[5]. Powstały mikroprocesory wielordzeniowe, które dziś są już praktycznie jedyną formą mikroprocesorów. Rys. 1.4 przedstawia typowy współczesny mikroprocesor wielordzeniowy. Widać na nim rdzenie oraz układy pamięci podręcznej, a także inne elementy, w skład których wchodzą między innymi układy sterowania dostępem do pamięci oraz komunikacji ze światem zewnętrznym.

[5]Pierwszym wielordzeniowym mikroprocesorem ogólnego przeznaczenia był układ Power4 firmy IBM z 2001 roku

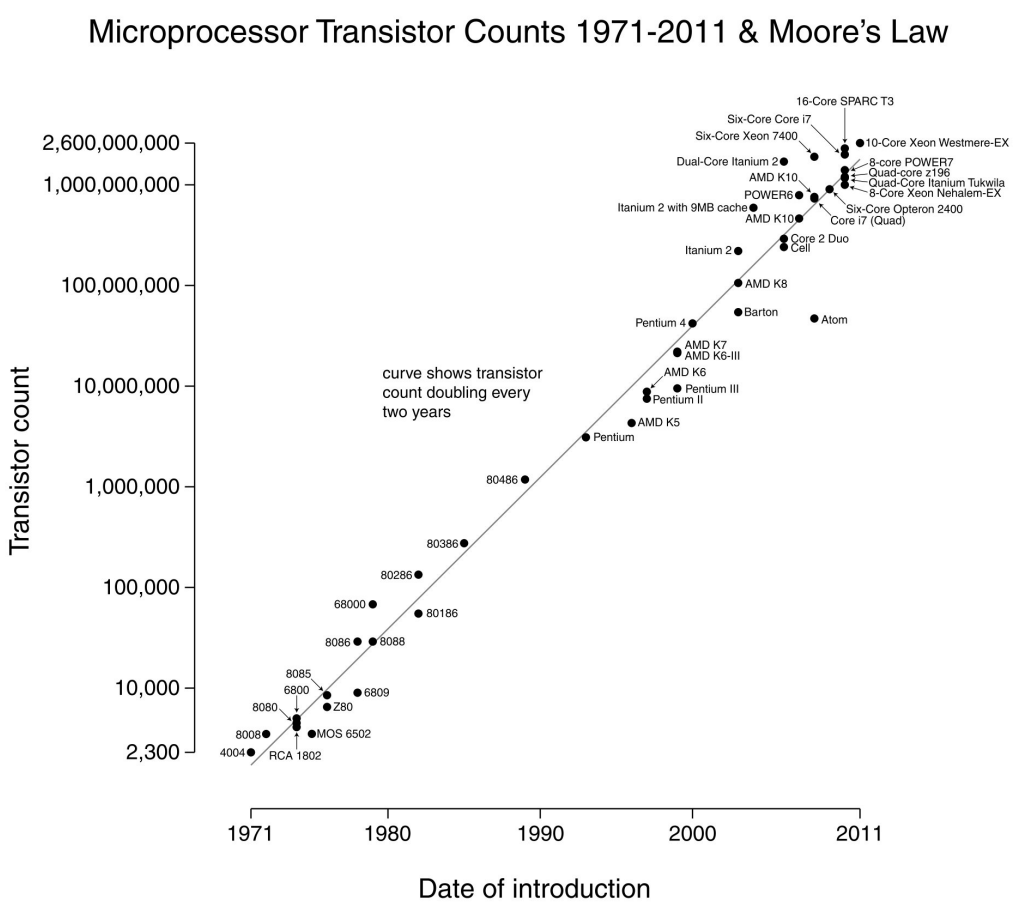


Rysunek 1.4: Procesor wielordzeniowy - widoczne rdzenie, układy pamięci podręcznej oraz inne elementy, m.in. układy sterowania dostępem do pamięci oraz komunikacji ze światem zewnętrznym. [źródło: Intel, 2010]

Określenia *rdzeń* i *mikroprocesor wielordzeniowy* mają ustalone, powszechnie przyjęte znaczenia. Natomiast pierwotny termin *procesor* bywa używany dwojako – w sensie pierwotnym, a więc jako synonim rdzenia, pojedynczej jednostki przetwarzania rozkazów, lub w sensie mikroprocesora, a więc obecnie jako synonim mikroprocesora wielordzeniowego. Pierwsze znaczenie odnosi się w większym stopniu do przetwarzania, a więc i do programowania, natomiast to drugie do elektroniki i budowy układów scalonych. Dlatego **w niniejszym skrypcie określenie *procesor* stosowane będzie głównie jako synonim określenia *rdzeń* – układ do przetwarzania ciągów (sekwencji) rozkazów, a więc układ sekwencyjny. Równoległym układem będzie natomiast mikroprocesor wielordzeniowy, jako w rzeczywistości system wieloprocesorowy w pojedynczym układzie scalonym**[6].

Rozwój mikroprocesorów przez ostatnie ponad czterdzieści lat możliwy był dzięki postępowi w dziedzinie elektroniki, postępowi, który najlepiej charakteryzuje prawo Moore'a (rys. 1.5). Zostało ono sformułowane przez założyciela firmy Intel, na podstawie zaobserwowanych tendencji, już w roku 1965. W formie odzwierciedlającej rzeczywisty rozwój technologii w ostatnich kilkudziesięciu latach, można je sformułować następująco: **liczba tranzystorów standardowo umieszczanych w pojedynczym układzie scalonym podwaja się w okresie każdych dwóch lat**. Obecnie postęp technologiczny w tworzeniu układów scalonych, który najprawdopodobniej trwać będzie w tempie zgodnym z prawem Moore'a jeszcze przez najbliższe kilka, kilkanaście lat, przekłada się na zwiększanie liczby rdzeni w pojedynczym mikroprocesorze wielordzeniowym. Można więc spodziewać się, że jeśli dziś mamy powszechnie do czynienia z mikroprocesorami czterordzeniowymi, to za dwa lata powszechne będą mikroprocesory ośmiordzeniowe. To oznacza, że najprawdopodobniej już **niedługo nastąpi era**

[6]Procesor jest także określeniem używanym w kontekście działania systemów operacyjnych, jako pojedyncza jednostka, której przydzielany jest do wykonania strumień rozkazów. W tym przypadku nawet mikroprocesory jednordzeniowe, ale wyposażone w tzw. *simultaneous multithreading*, zwany także *hyperthreading*, widziane są przez system operacyjny jako dwa lub więcej procesory (więcej na ten temat w skrypcie "Obliczenia wysokiej wydajności").



Rysunek 1.5: Prawo Moore'a i liczba tranzystorów rzeczywistych mikroprocesorów [źródło: Wikipedia, 2011]

mikroprocesorów masowo wielordzeniowych, mikroprocesorów z liczbą rdzeni rzędu kilkunastu, kilkudziesięciu i więcej.

W efekcie **każdy właściciel komputera będzie posiadał do dyspozycji maszynę zdolną do efektywnej pracy równoległej**. Czy można się spodziewać, że w takiej sytuacji ktoś może zdecydować się na korzystanie z programów sekwencyjnych zamiast programów równoległych, dobrowolnie zrezygnować z możliwości wykonywania programów kilkanaście razy szybciej? Nie jest to prawdopodobne i częściowo odpowiada na postawione wcześniej pytanie o przydatność przetwarzania równoległego. **Skoro sprzęt równoległy staje się tak masowy, równie masowe powinny być efektywne programy równoległe**. Z tej perspektywy wydaje się, że **żaden twórca oprogramowania nie może uniknąć konieczności pisania programów równoległych**. [7]

Jak zwykle, w przypadku kiedy pojawia się konieczność stosowania określonego paradygmatu

[7] Tylko w jednym przypadku zrozumiałe jest korzystanie z programów sekwencyjnych – wtedy kiedy nie da się stworzyć algorytmu równoległego do rozwiązania danego problemu obliczeniowego. Ale także w tym przypadku, żeby jednoznacznie dojść do takiego wniosku, trzeba rozumieć na czym polegają obliczenia równoległe i dlaczego konkretny problem nie pozwala na rozwiązanie równoległe. Co oznacza, że także w tym przypadku trzeba uprzednio przynajmniej podjąć próbę stworzenia programu równoległego.

programowania, pojawiają się także służące do tego narzędzia. Niniejszy skrypt omawia wiele takich narzędzi, koncentruje się jednak na narzędziach relatywnie niskiego poziomu, których użycie najczęściej wymaga zrozumienia funkcjonowania systemów równoległych. Istnieją środowiska programowania oferujące narzędzia (najczęściej biblioteki) służące tworzeniu programów równoległych, przy jak najmniejszym wysiłku ze strony programisty. Wielokrotnie jednak przy ich stosowaniu ujawnia się jedna z zasadniczych cech programowania równoległego: **nie jest trudno stworzyć program równoległy – program, w którym występuje wiele jednocześnie wykonywanych strumieni rozkazów. Znacznie trudniej jest stworzyć *poprawny* program równoległy – program, który przy każdym wykonaniu będzie zwracał poprawne rezultaty. Najtrudniej jest stworzyć *wysoko wydajny* program równoległy – program, który w swoim działaniu będzie osiągał skrócenie czasu przetwarzania zbliżone do liczby zastosowanych procesorów**. Do realizacji tego ostatniego celu, nawet w przypadku stosowania narzędzi tworzenia programów równoległych wysokiego poziomu, przydatne, a czasem konieczne jest dogłębne zrozumienie zasad działania programów i komputerów równoległych.

Drugim z celów stosowania obliczeń równoległych i rozproszonych jest zwiększenie niezawodności przetwarzania. Dotyczy to głównie dużych systemów, w szczególności takich, które powinny pracować w sposób ciągły. Standardowym sposobem przeciwdziałania skutkom rozmaitych awarii jest powielanie – tworzenie kopii zapasowych, zapewnianie powtarzalności operacji. Przetwarzanie równoległe może służyć zapewnieniu niezawodności – np. poprzez wykorzystanie kilku pracujących jednocześnie i wykonujących te same funkcje wątków, w taki sposób, że gdy jeden ulega awarii pozostają inne, świadcząc odpowiednie usługi, aż do czasu naprawy pierwszego wątku. Do zapewnienia niezawodności szczególnie dobrze nadaje się przetwarzanie rozproszone – jeśli mamy kilka wątków na kilku komputerach, awaria jednego z komputerów nie musi prowadzić do zaprzestania dostarczania usług.

Wreszcie trzecim z celów stosowania obliczeń równoległych i rozproszonych, tym razem dotyczącym głównie przetwarzania rozproszonego, jest zwiększenie elastyczności wykorzystania dostępnych zasobów komputerowych. Koronnym przykładem takiego sposobu funkcjonowania programów są systemy w ramach tzw. ”Grid Computing” i ”Cloud Computing”. W systemach tych użytkownik, korzystający ze swojej lokalnej maszyny połączonej siecią z innymi zasobami, zleca realizację pewnej usługi. System sam dobiera jakie konkretnie oprogramowanie i na jakim sprzęcie zrealizuje usługę.

Zasady działania powyższych systemów omówione są w tomie II niniejszego skryptu. W tomie niniejszym przedstawione są tylko podstawy funkcjonowania systemów rozproszonych, podstawowe problemy z tym związane i wybrane sposoby rozwiązywania tych problemów.

1.4 Pytania

Nie planuję programować równolegle – będę korzystać ze środowisk dostarczających biblioteki procedur wielowątkowych, czy wiedza o obliczeniach równoległych jest mi potrzebna?

Korzystanie z gotowych procedur wielowątkowych lub obiektowych struktur danych obsługiwanych przez procedury wielowątkowe staje się coraz popularniejsze, ze względu na konieczność stosowania obliczeń wielowątkowych i powszechne przekonanie o dużym stopniu trudności programowania równoległego. Nawet taka realizacja obliczeń równoległych wymaga pewnej wiedzy. Pojęciem często pojawiającym się przy opisie bibliotek wielowątkowych jest pojęcie *bezpieczeństwa wielowątkowego* (ang. *thread safety*), czyli zagwarantowania poprawności programu w przypadku wykonania wielowątkowego. Fragment kodu nazywać będziemy *wielowątkowo bezpiecznym*, jeżeli może on być wykonywany przez wiele współbieżnie pracujących wątków bez

ryzyka błędnej realizacji programu lub programów, w ramach których funkcjonują wątki. Nie tylko procedury zawierające jawnie kod wielowątkowy, ale także procedury wywoływane przez współbieżnie wykonywane wątki, powinny być bezpieczne wielowątkowo. Po to, aby wiedzieć, kiedy wykonanie danego fragmentu kodu jest bezpieczne, tzn. kiedy nie prowadzi do błędów wykonania, należy znać zasady realizacji obliczeń wielowątkowych i ogólniej współbieżnych.

Drugim przypadkiem, kiedy wiedza o przetwarzaniu równoległym może okazać się przydatna, jest sytuacja, niestety bardzo częsta, kiedy zastosowany kod wielowątkowy okazuje się być znacznie wolniejszy niż było to planowane (w ostateczności może okazać się wolniejszy niż kod sekwencyjny). Wiedza o zasadach realizacji obliczeń równoległych może pomóc odpowiednio skonfigurować środowisko wykonania tak, aby zdecydowanie zmienić wydajność wykonania równoległego, także w przypadku dostarczanych, gotowych procedur.

1.5 Test

- Przetwarzanie sekwencyjne w standardowych współczesnych systemach komputerowych:
  - ☐ jest sposobem przetwarzania w pojedynczym wątku
  - ☐ jest sposobem przetwarzania w pojedynczym procesie
  - ☐ może być realizowane tylko na mikroprocesorach jednordzeniowych
  - ☐ nie może być założone (przy braku jawnych mechanizmów synchronizacji) jako sposób realizacji procesu wielowątkowego
- Przetwarzanie współbieżne w standardowych współczesnych systemach komputerowych:
  - ☐ oznacza wykonywanie dwóch zbiorów rozkazów w taki sposób, że czasy wykonania nakładają się (nowe zadania zaczynają się zanim stare zostaną zakończone)
  - ☐ jest synonimem przetwarzania równoległego
  - ☐ nie daje się zrealizować w systemach jednoprocesorowych (jednordzeniowych)
  - ☐ służy głównie zwiększeniu stopnia wykorzystania sprzętu
- Przetwarzanie równoległe w standardowych współczesnych systemach komputerowych:
  - ☐ jest synonimem przetwarzania współbieżnego
  - ☐ jest synonimem przetwarzania wielowątkowego
  - ☐ nie daje się zrealizować w systemach jednoprocesorowych (jednordzeniowych)
  - ☐ służy głównie zwiększeniu wydajności przetwarzania
  - ☐ służy głównie zwiększeniu niezawodności przetwarzania
- Przetwarzanie rozproszone w standardowych współczesnych systemach komputerowych:
  - ☐ umożliwia zwiększenie wydajności przetwarzania
  - ☐ umożliwia zwiększenie niezawodności przetwarzania
  - ☐ oznacza uruchamianie programów na różnych komputerach połączonych siecią
  - ☐ jest szczególnym przypadkiem przetwarzania współbieżnego
  - ☐ jest szczególnym przypadkiem przetwarzania równoległego

- Przetwarzanie w przeplocie oznacza sytuacje kiedy:
  - □ system operacyjny przydziela wątki tego samego zadania różnym rdzeniom (procesorom)
  - □ system operacyjny realizuje przetwarzanie współbieżne na jednym procesorze (rdzeniu)
  - □ procesor (rdzeń) stosuje *simultaneous multithreading*
  - □ pojedynczy procesor (rdzeń) na przemian wykonuje fragmenty wielu wątków
  - □ pojedynczy proces korzysta na przemian z wielu rdzeni
- Zwiększenie wydajności obliczeń (skrócenie czasy realizacji zadań przez systemy komputerowe) jest głównym celem przetwarzania:
  - □ współbieżnego
  - □ równoległego
  - □ rozproszonego
  - □ wielowątkowego
- Zgodnie z prawem Moore'a liczba tranzystorów umieszczanych w pojedynczym układzie scalonym podwaja się co ... miesięcy.
- Prawo Moore'a stwierdza, że co stałą liczbę miesięcy podwaja się:
  - □ liczba tranzystorów umieszczanych w pojedynczym układzie scalonym
  - □ wydajność mikroprocesorów
  - □ częstotliwość taktowania zegara mikroprocesora
  - □ rozmiar pamięci podręcznej mikroprocesora
- Zgodnie z prawem Moore'a i współczesnymi kierunkami rozwoju architektur mikroprocesorów liczba rdzeni standardowego mikroprocesora w roku 2022 osiągnie:
  □ kilka □ kilkanaście □ kilkadziesiąt □ kilkaset (ponad 100)
- Zgodnie z prawem Moore'a i współczesnymi kierunkami rozwoju architektur mikroprocesorów liczba rdzeni standardowego mikroprocesora osiągnie kilkadziesiąt (ponad 20) około roku:
  □ 2013 □ 2016 □ 2019 □ 2022
- Producenci procesorów ogólnego przeznaczenia przestali zwiększać częstotliwość pracy procesorów z powodu:
  - □ zbyt wielu etapów w potokowym przetwarzaniu rozkazów, utrudniających zrównoleglenie kodu
  - □ zbyt wysokiego poziomu wydzielania ciepła
  - □ barier technologicznych w taktowaniu układów elektronicznych
  - □ niemożności zagwarantowania odpowiednio szybkiej pracy pamięci podręcznej
- Płyta kuchenki elektrycznej produkuje ok. 10 W/cm2 natomiast współczesne procesory około ... W/cm2
- Rdzeń mikroprocesora wielordzeniowego:
  - □ jest częścią mikroprocesora odpowiedzialną za pobieranie i dekodowanie rozkazów, ale nie wykonuje rozkazów (jednostki funkcjonalne znajdują się poza rdzeniem)

- ☐ jest bardzo zbliżony do dawnych procesorów jednordzeniowych, ale nie potrafi wykonywać bardziej złożonych rozkazów
- ☐ jest bardzo zbliżony do dawnych procesorów jednordzeniowych, ale bez pamięci podręcznej L2 i L3
- ☐ wydziela znacznie mniej ciepła niż procesor jednordzeniowy o tej samej częstotliwości pracy

- Bezpieczeństwo wielowątkowe (*thread safety*) procedury oznacza, że (zaznacz tylko odpowiedzi ujmujące istotę bezpieczeństwa wielowątkowego, a nie każde prawdziwe stwierdzenie):
  - ☐ może ona być wykonywana przez dowolnie dużą liczbę wątków
  - ☐ można w niej tworzyć wątki
  - ☐ może być wykonywana przez wiele współbieżnych wątków bez wprowadzania błędów wykonania
  - ☐ może być procedurą startową wątków
  - ☐ inne procedury z innych wątków nie mogą zakłócić jej prawidłowego przebiegu