# Paradygmaty Programowania

## Programowanie współbieżne

dr inż. Marcin Bączyk

Wykład 8

25 kwietnia 2022

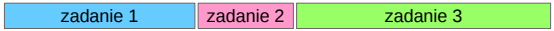
# Treść dzisiejszego wykładu

# Literatura

- Zbigniew. J. Czech “Wprowadzenie do obliczeń równoległych”, PWN, 2013
- https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

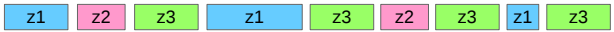
# Przetwarzanie sekwencyjne a współbieżne

- **Przetwarzanie sekwencyjne** – kolejne zadanie jest rozpoczynane dopiero po zakończeniu poprzedniego zadania.




- W przeciwnym razie – **przetwarzanie współbieżne** (ang. *concurrent*):
  - **równoległe** (ang. *parallel*) – zdania podzielone między jednostki obliczeniowe,


  - **z przeplotem** (ang. *interleaving*, *interlacing*) – jedna jednostka obliczeniowa wykonuje naprzemiennie kolejne etapy różnych zadań.




- Wykonanie z przeplotem:
  - wymaga przełączania kontekstu procesora – może wydłużać całkowity czas działania,
  - stwarza *złudzenie* wykonania równoczesnego,
  - może zwiększać płynność działania systemu jako całości.

# Programowanie równoległe

Przetwarzanie równoległe to jednoczesne wykorzystanie wielu jednostek obliczeniowych w celu rozwiązania problemu obliczeniowego.

Typowe konfiguracje obliczeniowe to:

- jeden komputer z wieloma procesorami/rdzeniami,
- komputery połączone przez sieć,
- kombinacja powyższych.

Programowanie równoległe ma na celu tworzenie programów pozwalających na wykorzystanie systemów wieloprocesorowych do rozwiązywania złożonych problemów obliczeniowych.

## Przetwarzanie równoległe – motywacja

- Powala zaoszczędzić czas.
- Pozwala rozwiązywać problemy o większym stopniu złożoności.
- Systemy rozproszone lepiej radzą sobie z przetwarzaniem danych pozyskiwanych w sposób rozproszony (np. radary multistatyczne).
- Wykorzystanie wielu wolnych ale tanich procesorów zazwyczaj jest efektywniejsze kosztowo niż jednego ale szybszego procesora.
- Pojedyncze procesory mają ograniczenie co do ilości obsługiwanej pamięci.
- Zwiększanie częstotliwości taktowania zegarów procesorów nie może być kontynuowane (ograniczenia fizyczne i ekonomiczne).

# Architektura von Neumanna

W architekturze von Neumanna zakłada się, że program przechowywany jest wraz z danymi w pamięci ogólnego przeznaczenia. Jednostka centralna (CPU) wykonuje program (sekwencję instrukcji) zapisany w pamięci, wykonując głównie operacje odczytu z pamięci i zapisu do niej.

## Założenia

- W pamięci przechowywane są zarówno dane, jak i instrukcje programu.
- Dane to informacje wykorzystywane przez program.
- Instrukcje sterują przebiegiem programu.
- CPU odczytuje instrukcje (program) oraz dane, następnie dekoduje odczytane instrukcje, wykonuje je i zapisuje wynik w pamięci w sposób sekwencyjny

# Taksonomia Flynna

Według Flynna przetwarzanie równoległe można podzielić ze względu na:

- **wykonywanie instrukcji**: sekwencyjne lub równoległe,
- **przetwarzanie danych**: pojedynczo lub większymi porcjami.

| | dane | |
|---|---|---|
| instrukcje | **SISD**<br>Single Instruction, Single Data | **SIMD**<br>Single Instruction, Multiple Data |
| | **MISD**<br>Multiple Instruction, Single Data | **MIMD**<br>Multiple Instruction, Multiple Data |

Architektura SISD odpowiada klasycznej architekturze sekwencyjnej (niewspółbieżnej). Taksonomia Flynna dotyczy architektury maszyny, która ma znaczący wpływ na przetwarzanie równoległe.

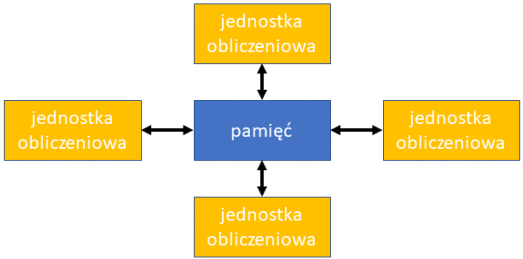
# Taksonomia Flynna

- **Single Instruction, Single Data (SISD)** – brak równoległości.
- **Single Instruction, Multiple Data (SIMD)**:
  - wszystkie jednostki wykonują w danym cyklu zegara tę samą instrukcję, ale...
  - każda z nich operuje na innym elemencie danych.
- **Multiple Instruction, Single Data (MISD)**:
  - każda jednostka wykonuje inną operację na tym samym elemencie danych,
  - bardzo rzadko spotykane.
- **Multiple Instruction, Multiple Data (MIMD)**
  - wiele jednostek wykonawczych,
  - każda przetwarza niezależnie własny strumień danych,
  - elementem tej architektury mogą być niezależne jednostki SIMD,
  - na tej zasadzie działa większość współczesnych komputerów.

# Architektura pamięci w przetwarzaniu współbieżnym

## Pamięć współdzielona

- Istnieje jedna przestrzeń adresowa dla wszystkich zadań.
- Wszystkie jednostki obliczeniowe mają dostęp do wspólnej części pamięci.
- Poszczególne jednostki operują niezależnie – brak **komunikacji**.
- Zmiany wprowadzone do pamięci przez jakąkolwiek jednostkę obliczeniową są widziane przez pozostałe.



# Architektura pamięci w przetwarzaniu współbieżnym

## Zalety stosowania pamięci współdzielonej

- Programista nie musi dbać o komunikację między poszczególnymi zadaniami i przesyłanie danych.
- Współdzielenie danych przez poszczególne jednostki obliczeniowe odbywa się bez opóźnień związanych z komunikacją.

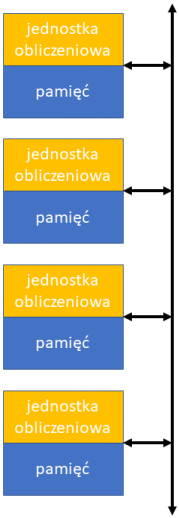
## Wady stosowania pamięci współdzielonej

- Brak skalowalności – dodanie dodatkowych jednostek obliczeniowych może zwiększyć ruch pomiędzy procesorami a pamięcią.
- Narzuca odpowiedzialność na programistę na **synchronizację** dostępu do pamięci.
- Pamięć o dostępie dla dużej liczby jednostek obliczeniowych jest droga w produkcji – konieczność wirtualizacji.

# Architektura pamięci w przetwarzaniu współbieżnym

## Pamięć rozproszona

- Każda jednostka obliczeniowa ma swoją lokalną pamięć oraz swoją przestrzeń adresową.
- Poszczególne jednostki obliczeniowe korzystają ze swojej pamięci niezależnie.
- Zmiana zawartości pamięci lokalnej nie ma wpływu na inne jednostki obliczeniowe.
- Dostęp do danych innej jednostki jest możliwy, ale skomplikowany i czasochłonny.
- Niezbędna jest synchronizacja pomiędzy poszczególnymi jednostkami obliczeniowymi.
- Wymagane jest połączenie do komunikacji pomiędzy poszczególnymi jednostkami obliczeniowymi.



# Architektura pamięci w przetwarzaniu współbieżnym

## Zalety stosowania pamięci rozproszonej

- Pamięć jest skalowalna wraz ze wzrostem liczby jednostek obliczeniowych.
- Każda jednostka obliczeniowa może korzystać ze swojej własnej pamięci "bezzwłocznie", tj. bez potrzeby synchronizacji z innymi jednostkami.
- Niski koszt podzespołów – systemy składane w klastry z dużej liczby relatywnie tanich jednostek obliczeniowych.

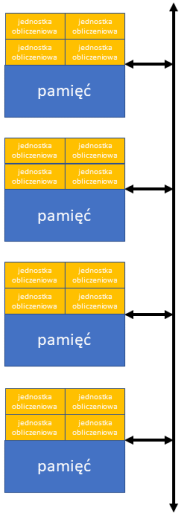
## Wady stosowania pamięci rozproszonej

- Programista jest odpowiedzialny za oprogramowanie dużej ilości szczegółów związanych z komunikacją pomiędzy jednostkami obliczeniowymi
- Istnieją duże trudności z odwzorowaniem istniejących struktur danych.

# Architektura pamięci w przetwarzaniu współbieżnym

## Pamięć hybrydowa

- Wykorzystywana w przypadku łączenia wielu jednostek obliczeniowych zawierających wiele procesorów.
- Poszczególne procesory wykorzystują swoją pamięć lokalną.
- Żądanie skorzystania z pamięci innej jednostki obliczeniowej musi być zaimplementowane przez programistę.
- Łączy wady i zalety stosowania zarówno pamięci współdzielonej jak i rozproszonej.



### Wykorzystanie pamięci współdzielonej

- Poszczególne zadania współdzielą przestrzeń adresową, odczytują ją i zapisują do niej w sposób asynchroniczny.
- Poszczególne zadania nie są wyłącznymi właścicielami danych na których operują (mimo że żadne inne zadanie nie musi operować na tym obszarze danych).
- Nie ma konieczności specyfikacji komunikacji pomiędzy poszczególnymi zadaniami (należy wziąć pod uwagę związane z tym konsekwencje).
- Istnieją trudności w zarządzaniu danymi.

## Wątki

- Pojedynczy **proces** może uruchomić wiele równoległych **wątków**.
- Poszczególne wątki mogą przetwarzać równolegle różne dane lub uruchamiać różne podprogramy.
- Standardowo wykonywanie wątków zarządzane jest przez system operacyjny, ale istnieją również inne implementacje.
- Każdy z wątków może mieć własne dane (zmienne lokalne), ale korzystają również z zasobów przekazanych im przez proces nadrzędny (wspólnych).
- Synchronizacja jest konieczna, aby w tym samym czasie, żadne dwa wątki nie aktualizowały tego samego obszaru pamięci.
- Programowanie równoległe z wykorzystaniem **wątków** kojarzone jest głównie z wykorzystaniem **pamięci współdzielonej**.

# Modele programowania współbieżnego

## Proces

W systemach wielozadaniowych w tym samym czasie mogą być uruchomione różne programy i procesy. System operacyjny zarządza procesami i przydziela im zasoby, takie jak czas procesora czy pamięć.

## Procesy

- Programowanie współbieżne bazujące na **procesach** kojarzone jest z wykorzystaniem **rozproszonej** architektury pamięci.
- W celu wymiany danych stosuje się komunikację międzyprocesową:
  - pliki
  - sygnały
  - semafory
  - potoki
  - wiadomości
  - gniazda
  - pamięć współdzieloną

## Przesyłanie komunikatów

- Poszczególne zadania wykorzystują własną pamięć w trakcie wykonywania obliczeń.
- Poszczególne zadania mogą być uruchomione na tej samej lub różnych jednostkach obliczeniowych.
- Zadania komunikują się poprzez wysyłanie i odbieranie komunikatów.
- Przesyłanie danych wymaga określonych operacji wykonanych przez każdy z procesów.

## Zrównoleglanie

Projektowanie i implementowanie programów (algorytmów) współbieżnych jest odpowiedzialnością programisty.

Niemniej istnieje kilka technik pozwalających na częściowe zautomatyzowanie procesu zrównoleglania:

- Włączenie odpowiedniej opcji kompilatora, która pozwala kompilatorowi na identyfikację fragmentów kodu nadających się do zrównoleglania.
- Kompilator często identyfikuje różnego rodzaju pętle jako potencjalne fragmenty nadające się do zrównoleglania.
- Dzięki odpowiednim dyrektywom kompilatora możliwe jest instruowanie go, w jaki sposób ma przeprowadzić proces zrównoleglania.

### Zrównoleglanie

Proces automatycznego zrównoleglania ma również znaczące ograniczenia:

- wyniki obliczeń mogą nie być poprawne
- program może być wolniejszy ze względu na narzuty związane pracą równoległą
- kompilatory potrafią zrównoleglać głównie pętle

Znacząco lepsze wyniki można uzyskać poprzez ręczną identyfikację fragmentów programu, które mogą być uruchamiane współbieżnie i odpowiednie ich oprogramowanie.

## Zrównoleglanie ręczne

- Sprawdź, czy dany problem może być zrównoleglony.
  - Pewne problemy nie nadają się do zrównoleglenia, np. obliczenie $n$-tej potęgi liczby.
  - Inne zadania mogą być zrównoleglane, jednak wymagają częstej synchronizacji w celu wymiany danych, np. algorytm wyznaczania szybkiej transformaty Fouriera.
  - Istnieją również zadania, które ze względu na swoją strukturę zrównoleglają się bardzo łatwo, np. wyznaczenie wartości funkcji dla dużego zbioru danych.
- Zidentyfikuj kluczowe fragmenty programu, której zajmują najwięcej czasu procesora.
- Zidentyfikuj wąskie gardła programu wpływające na wydajność algorytmu, np. operacje wejścia-wyjścia.
- Zidentyfikuj elementy uniemożliwiające przeprowadzenie obliczeń równoległych, np. zależność od danych jak w przypadku z podnoszeniem liczby do potęgi.
- Rozważ wykorzystanie innego algorytmu.

## Dekompozycja

Jednym z pierwszych kroków projektowania programów współbieżnych jest podział programu na skończoną liczbę zadań, które mogą być wykonane w tym samym czasie.

- **Dekompozycja dziedzinowa**

  Podział związany z danymi. Każde z wykonywanych równolegle zadań wykonuje ten sam podprogram wykorzystując inną porcję danych.

- **Dekompozycja funkcjonalna**

  Podział związany jest z rodzajem obliczeń jakie należy wykonać. Poszczególne zadania mogą tworzyć określoną sekwencję.

- **Dekompozycja dziedzinowa i funkcjonalna**

  Dzielenie zadań ze względu na fragment danych jak i rodzaj obliczeń jest naturalne i często spotykane.

### Komunikacja

Istnieją problemy, które mogą być zdekomponowane na zadania niewymagające wzajemnej komunikacji, np. przetwarzanie zdjęć piksel po pikselu lub przeszukiwanie zdjęć satelitarnych w celu dopasowania do wzorca.

Większość programów współbieżnych wykorzystuje bardziej złożone algorytmy i wymaga współdzielenia informacji pomiędzy poszczególnymi zadaniami, np. symulacja fali elektromagnetycznej metodą elementów skończonych wymaga wiedzy na temat stanu sąsiadów danego węzła aby prawidłowo wyznaczyć jego stan.

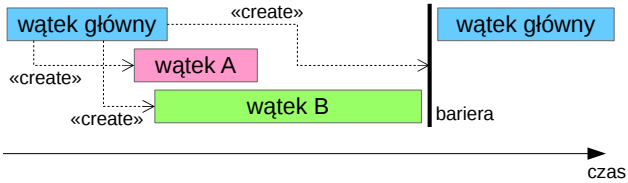
## Zagadnienia związane z komunikacją

Planując komunikację pomiędzy poszczególnymi zadaniami, należy wziąć pod uwagę między innymi:

- Koszt komunikacji – cykle zegara zużywane są na obsługę komunikacji zamiast na wykonywania zadań.
- Przepustowość łącza i jego opóźnienie.
- Świadomość istnienia komunikacji – w przypadku współdzielenia pamięci istnieje możliwość niezauważenia komunikacji i wynikania z tego błędów.
- Komunikacja synchroniczna i asynchroniczna – w przypadku komunikacji synchronicznej oba zadania uczestniczące są blokowane na czas przesyłania komunikatów.
- Zasięg komunikacji.
- Efektywność komunikacji – niektóre sposoby komunikacji są efektywniejsze na danej platformie obliczeniowej.

# Podstawowe techniki synchronizacji – bariera

- **Bariera** zatrzymuje wykonanie programu do momentu, w którym wszystkie "zainteresowane" procesy/wątki robocze zakończą wykonywanie swoich zadań.

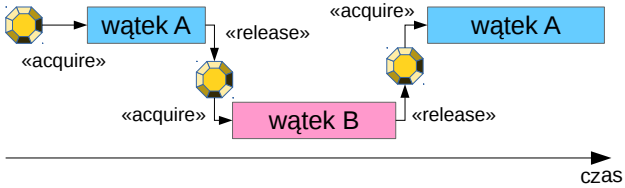



- Z reguły jest tworzona przez wątek główny programu dla wybranych wątków.
- W większości języków programowania (C++, Java, Python):

```
wątek_roboczy.join()
```

# Podstawowe techniki synchronizacji – muteks

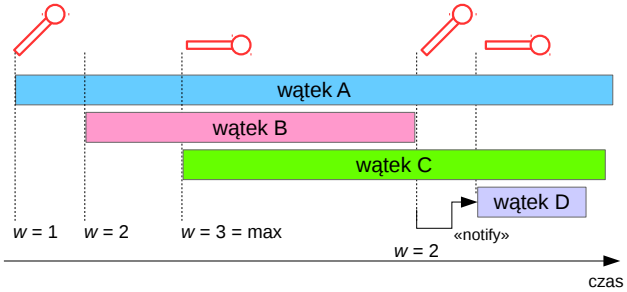
- **Muteks** (od *mutual exclusion*) reprezentuje prawo dostępu do wspólnego zasobu (np. pliku, struktury danych).




- W danej chwili tylko jeden wątek może korzystać z zasobu.
- Pierwszy wątek, któremu uda się pozyskać (ang. *acquire*) muteks, nabywa tym samym prawa do korzystania z zasobu.
- Inne wątki muszą czekać aż ten, któremu udało się zająć zasób, przestanie na nim operować i zwolni (ang. *release*) muteks.

# Podstawowe techniki synchronizacji – semafor

- **Semafor** (ang. *semaphore*) jest uogólnioną wersją muteksu.
- Określa maksymalną liczbę wątków mogących w danej chwili korzystać z zasobu.
- Często pozwala na powiadamianie (ang. *notify*) wątków o tym, że zasób jest już dostępny.




Muteksy i semafory są także używane do synchronizacji **procesów, nie tylko wątków**.

### Zależność danych

W kontekście przetwarzania równoległego zależność danych oznacza, że poszczególne zadania zależą od wyników przetwarzania innych zadań. Zależność ta może być cykliczna i może mieć wpływ na wyniki przetwarzania (np. w metodzie elementów skończonych).

Zależności danych powstają między innymi, gdy wiele wątków korzysta wielokrotnie z tych samych danych.

Zależność danych jest jednym z głównych przyczyn ograniczania możliwości zrównoleglania programów.

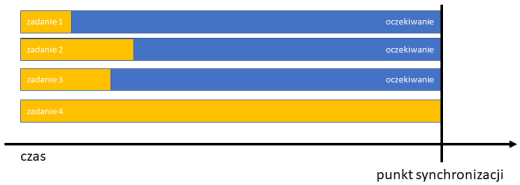
W przypadku pamięci rozproszonej konieczne jest komunikacja w punktach synchronizacji, w przypadku pamięci współdzielonej – synchronizowanie operacji odczytu i zapisu.

## Równoważenie obciążenia

Równoważenie obciążenia odnosi się do praktyki rozdzielania zadań między wątki, tak aby wszystkie one były przez cały czas zajęte. Można to uznać za minimalizację czasu bezczynności wątków.

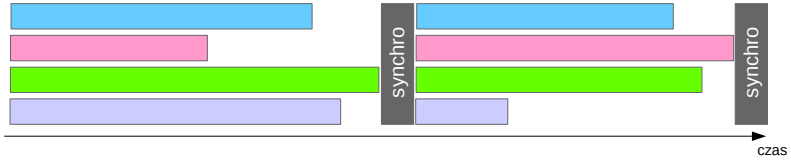
Równoważenie obciążenia jest ważne dla programów równoległych ze względu na wydajność. Na przykład, jeśli wszystkie wątki podlegają punktowi synchronizacji bariery, najwolniejszy watek określi ogólną wydajność.

W przypadku podziału programu na zadania wykonywane równolegle należy zadbać, aby czas wykonania wszystkich zadań był porównywalny. Konkretne maszyny mogą być mniej lub bardziej wydajne.

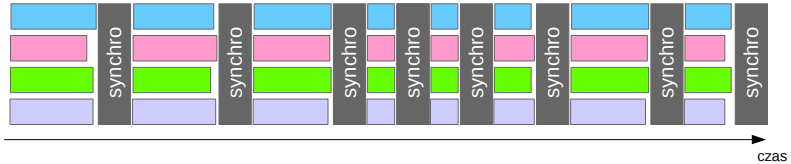


# Projektowanie programów współbieżnych – ziarnistość

- W obliczeniach równoległych **ziarnistość** jest jakościową miarą stosunku obliczeń do komunikacji. Okresy obliczeń są zwykle oddzielone od okresów komunikacji przez zdarzenia synchronizacji.
- Podział **gruboziarnisty** zadań:




- Podział **drobnoziarnisty** zadań:



## Równoległość drobnoziarnista

- Pomiędzy zdarzeniami komunikacyjnymi wykonywana jest stosunkowo niewielka ilość pracy obliczeniowej.
- Niski stosunek czasu obliczeń do czasu poświęconego na komunikację.
- Ułatwia równoważenie obciążenia.
- Jeśli poziom szczegółowości jest zbyt wysoki, możliwe, że narzut wymagany do komunikacji i synchronizacji między zadaniami zajmie więcej czasu niż obliczenia.

## Równoległość gruboziarnista

- Stosunkowo duże ilości pracy obliczeniowej są wykonywane między zdarzeniami komunikacyjnymi / synchronizacyjnymi
- Wysoki współczynnik obliczeń do komunikacji.
- Trudniej zrównoważyć obciążenie.

- Najbardziej wydajna ziarnistość zależy od algorytmu i środowiska sprzętowego, w którym działa.
- W większości przypadków narzut związany z komunikacją i synchronizacją jest wysoki w stosunku do szybkości wykonywania, więc korzystne jest, aby uzyskać grubą ziarnistość.
- Równoległość drobnoziarnista może pomóc zmniejszyć koszty ogólne z powodu nierównowagi obciążenia.

## Wejście/wyjście

- Operacje wejścia-wyjścia są uważane za elementy utrudniające zrównoleglanie.
- W środowisku, w którym wszystkie zadania mają ten sam obszar przestrzeni dyskowej, operacje zapisu spowodują nadpisanie pliku.
- Na operacje odczytu będzie miała wpływ zdolność serwera plików do obsługi wielu żądań odczytu w tym samym czasie.
- Operacje wejścia-wyjścia wykorzystujące sieć mogą stanowić wąskie gardła.

- Zredukuj ogólne operacje wejścia-wyjścia tak bardzo, jak to możliwe.
- Ogranicz operacje wejścia-wyjścia do określonych części szeregowych zadania, a następnie korzystaj z komunikacji równoległej w celu dystrybucji danych do zadań równoległych.
- W przypadku systemów z pamięcią rozproszoną ze współużytkowanym obszarem plików operacje wejścia-wyjścia należy wykonywać w lokalnym, niewspółużytkowanym obszarze plików.
- Twórz unikatowe nazwy dla plików wejściowych-wyjściowych każdego zadania.

### Prawo Amdahla

Maksymalne przyspieszenie $S_{max}$ programu jest ograniczone niezależnie od liczby procesorów i opisuje je wzór:

$$S_{max} = \frac{1}{1 - P} \tag{1}$$

gdzie $P$ to stosunek części programu, która może być zrównoleglona do całości programu.
Przyspieszenie $S$ wynikające z wykorzystania $N$ procesorów jest dane wzorem:

$$S = \frac{1}{\frac{P}{N} + 1 - P} \tag{2}$$

Programy, których jedynie niewielka część może być zrównoleglona, charakteryzują się niewielkim przyspieszeniem, niezależnie od liczby wykorzystanych procesorów.

# Podsumowanie

- Aplikacje równoległe są znacznie bardziej złożone niż odpowiadające im aplikacje szeregowe, prawdopodobnie o rząd wielkości. Występuje nie tylko wiele strumieni instrukcji wykonywanych w tym samym czasie, ale także przepływają między nimi dane .
- Koszty złożoności mierzone są czasem programisty praktycznie w każdym aspekcie cyklu tworzenia oprogramowania: projekt, kodowanie, debugowanie i konserwacja.
- Przestrzeganie dobrych praktyk tworzenia oprogramowania jest niezbędne podczas pracy z aplikacjami równoległymi.
- Wszystkie typowe problemy z przenośnością związane z programami szeregowymi dotyczą również (jeżeli nie bardziej) programów równoległych.

# Podsumowanie

- Podstawowym celem programowania równoległego jest zmniejszenie czasu wykonywania programu ("*wallclock time*"), jednak aby to osiągnąć, potrzeba więcej czasu procesora.
- Ilość wymaganej pamięci może być większa w przypadku programów równoległych niż szeregowych, ze względu na konieczność replikacji danych i narzuty związane z równoległymi bibliotekami i podsystemami.
- Koszty ogólne związane z konfiguracją środowiska równoległego, tworzeniem zadań, komunikacją i zakończeniem zadań mogą stanowić znaczną część całkowitego czasu operowania na małych zbiorach danych.
- Możliwość skalowania wydajności programu równoległego jest wynikiem wielu powiązanych ze sobą czynników. Algorytm może mieć immanentne ograniczenia skalowalności. W pewnym momencie dodanie większej liczby zasobów powoduje spadek wydajności.
- Biblioteki obsługi równoległej i oprogramowanie podsystemów może ograniczać skalowalność niezależnie od aplikacji.

# Słownik pojęć

**Zadanie**

Logicznie dyskretna sekcja pracy obliczeniowej. Zadanie to zwykle program lub podobny do programu zestaw instrukcji wykonywanych przez procesor.

**Zadanie równoległe**

Zadanie, które może być bezpiecznie wykonywane przez wiele procesorów (daje prawidłowe wyniki).

# Słownik pojęć

**Wykonanie szeregowe**

Wykonywanie programu sekwencyjnie, jedna instrukcja na raz. Mówiąc najprościej, dzieje się tak na komputerze z jednym procesorem. Jednak praktycznie wszystkie zadania równoległe będą miały sekcje programu równoległego, które muszą być wykonywane szeregowo.

**Wykonanie równoległe**

Wykonanie programu przez więcej niż jedno zadanie, przy czym każde zadanie może wykonać tę samą lub inną instrukcję w tym samym momencie.

# Słownik pojęć

## Pamięć współdzielona

Ze ściśle sprzętowego punktu widzenia opisuje architekturę komputera, w której wszystkie procesory mają bezpośredni (zwykle oparty na magistrali) dostęp do wspólnej pamięci fizycznej. W sensie programowania opisuje model, w którym zadania równoległe mają ten sam „obraz" pamięci i mogą bezpośrednio adresować te same logiczne lokalizacje pamięci i uzyskiwać do nich dostęp, niezależnie od tego, gdzie faktycznie istnieje pamięć fizyczna.

## Pamięć rozproszona

W przypadku sprzętu odnosi się do dostępu sieciowego do pamięci fizycznej, który nie jest powszechny. Jako model programowania, zadania mogą logicznie „widzieć" lokalną pamięć komputera i muszą używać komunikacji, aby uzyskać dostęp do pamięci na innych komputerach, na których są wykonywane inne zadania.

# Słownik pojęć

## Komunikacja

Równoległe zadania zwykle wymagają wymiany danych. Można to osiągnąć na kilka sposobów, na przykład za pośrednictwem współdzielonej magistrali pamięci lub sieci, jednak faktyczne zdarzenie wymiany danych jest powszechnie określane jako komunikacja, niezależnie od zastosowanej metody.

## Synchronizacja

Koordynacja równoległych zadań w czasie rzeczywistym, bardzo często związanych z komunikacją. Często wdrażane przez ustanowienie punktu synchronizacji w aplikacji, w którym zadanie nie może być kontynuowane, dopóki inne zadanie(a) nie osiągnie tego samego lub logicznie równoważnego punktu.
Synchronizacja zwykle wiąże się z czekaniem przez co najmniej jedno zadanie i dlatego może spowodować wydłużenie czasu wykonywania aplikacji równoległej.

# Słownik pojęć

## Ziarnistość

W obliczeniach równoległych ziarnistość jest jakościową miarą stosunku obliczeń do komunikacji. Gruba: stosunkowo duże ilości pracy obliczeniowej są wykonywane między zdarzeniami komunikacyjnymi Drobna: stosunkowo niewielkie ilości pracy obliczeniowej są wykonywane między zdarzeniami komunikacyjnymi

## Zaobserwowane przyspieszenie

Zaobserwowane przyspieszenie kodu, który został zrównoleglony, zdefiniowane jako stosunek czasu wykonania seryjnego do czasu wykonywania równoległego.
Jeden z najprostszych i najczęściej stosowanych wskaźników wydajności równoległego programu.

## Koszty zrównoleglania

Ilość czasu wymagana do koordynowania zadań równoległych, w przeciwieństwie do wykonywania pożytecznej pracy. Koszty te mogą obejmować takie czynniki, jak:

- czas rozpoczęcia zadania:
  - uruchomienie potrzebnej liczby procesów lub wątków,
  - dystrybucję danych pomiędzy procesory (zwłaszcza w modelu rozproszonym),
- synchronizację,
- komunikację,
- narzut oprogramowania spowodowany przez biblioteki, narzędzia, system operacyjny itp.
- czas zakończenia zadania:
  - zakończenie pracy procesów lub wątków,
  - łączenie wyników cząstkowych (zwłaszcza w modelu rozproszonym).

# Słownik pojęć

## Skalowalność

Odnosi się do zdolności systemu równoległego (sprzętu i/lub oprogramowania) do wykazania proporcjonalnego wzrostu szybkości po dodaniu większej liczby procesorów. Czynniki wpływające na skalowalność obejmują:

- sprzęt – w szczególności przepustowości procesora, pamięci i komunikacji sieciowej,
- algorytm aplikacji,
- równoległe koszty ogólne,
- charakterystykę konkretnej aplikacji i kodowania.