Programowanie równoległe i rozproszone

dr inż. Jacek Starzyński

Zakład Elektrotechniki Teoretycznej i Informatyki Stosowanej PW

Wykład na podstawie:
materiałów, które opracował *dr inż. Piotr Gronek, AGH*,
Designing and Building Parallel Programs, by *Ian Foster*

Program wszystkich wykładów

- Terminologia
- Modele programowania równoległego
- Projektowanie algorytmów równoległych
- Miary efektywności obliczeń równoległych
- Architektury maszyn równoległych
- Paradygmaty i elementy programowania równoległego
- Programowanie na maszynach z pamięcią wspólną
  - procesy, wątki, semafory, synchronizacja, IPC
- Programowanie na maszynach z pamięcią dzieloną
  - MPI, PVM, Linda
- Narzędzia do obiektowego programowania rozproszonego
  - RPC, RMI (Java), CORBA

Literatura

- Obliczenia Równoległe i Rozproszone, pod red. Andrzeja Karbowskiego i Ewy Niewiadomskiej-Szynkiewicz
- Designing and Building Parallel Programs, by Ian Foster
- Parallel Programming in C with MPI and OpenMP, by M. J. Quinn
- MPI: The Complete Reference, by M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra
- Using MPI: Portable Parallel Programming with the Message-Passing Interface, by W. Gropp, E. Lusk, and A. Skjellum
- Introduction to Parallel Computing, by A. Grama, A. Gupta, G. Karypis, V. Kumar
- The Message Passing Interface (MPI) standard
- PVM (Parallel Virtual Machine)
- Catalog of OMG CORBA®/IIOP® Specifications
- The Java Tutorial
- Numerical Recipes books in C, Fortran 77, and Fortran 90 On-Line

Laboratorium

- Procesy w systemie Unix
- Sygnały w systemie Unix
- Wątki w systemie Unix (C)
- MPI w Windows
- MPI w Linuksie
- PVM: Podstawy komunikacji
- PVM: Obliczanie liczby Pi
- Miniprojekty do wyboru: algorytm genetyczny, SA, MRS, własny (aplikacja wielowątkowa w C, aplikacja wielowątkowa w Javie, aplikacja Java RMI)

Zasady zaliczenia

- Wykład kończy się pisemnym sprawdzianem ocenianym w skali 0-60.
- Obecność na laboratorium jest obowiązkowa.
- Laboratorium składa się z zajęć wprowadzających, 7 ćwiczeń i miniprojektu.
- Ćwiczenia wykonywane są indywidualnie.
- Każde ćwiczenie laboratoryjne kończy się sprawdzianem ocenianym w skali 0-5.
- Miniprojekty wykonywane są w zespołach 3-osobowych. Każdy z członków zespołu ma ustalony zakres obowiązków.
- Ostateczny termin zakończenia (obrony) miniprojektów to ostatnie zajęcia w semestrze.
- Miniprojekty oceniane są w skali 0-25. Ocena jest indywidualna.
- Ostateczna ocena z przedmiotu wynika z sumy uzyskanych punktów: 0-60 = nzal, 61-72 = 3, 73-84 = 3.5, 85-96 = 4, 97-108 = 4.5, 109-120 = 5

Program na dziś

- Wprowadzenie, terminologia
- Zastosowanie obliczeń równoległych
- Modele OR
- Projektowanie algorytmów dla OR
- Metodyka PCAM

Pojęcia podstawowe

- Systemy wieloprocesorowe
  - możliwość wykonywania wielu procesów obliczeniowych jednocześnie
- Sieci komputerowe – klastry (grona)
  - współużytkowanie zasobów obliczeniowych

Pojęcia podstawowe

- Równoległość procesowa – zbiór złożonych współpracujących elementów, działających w zbliżony sposób
- Równoległość tablicowa – pełna synchronizacja działań
- Równoległość potokowa – jednoczesne wykonywanie kolejnych etapów złożonego procesu

Zalety obliczeń równoległych

- Przyspieszenie obliczeń
- Zwiększenie niezawodności działania lub dokładności
- Możliwość rozwiązywania zadań o większych rozmiarach

Obliczenia współbieżne

- Obliczenia wykonywane w tym samym czasie (tj. rozpoczynające się przed zakończeniem innych) – także w systemach jednoprocesorowych (z podziałem czasu)
- Pozwalają na dekompozycję funkcjonalną programu (funkcjonalność rozdzielona na poszczególne moduły)
- Jest to szersze pojęcie, niż „obliczenia równoległe”

Obliczenia rozproszone

- Obliczenia, w których dokonano dekompozycji zadania obliczeniowego, niezależnie czy jest to podział na programy, procesy czy procesory
- Brak wspólnej przestrzeni adresowej dla wszystkich procesów realizujących zadanie

Przykłady zastosowań

- Cyfrowa prognoza pogody
- Animacja 3D wysokiej rozdzielczości
- Modelowanie klimatu
- Badanie własności związków chemicznych (leki, tworzywa, półprzewodniki)
- Projektowanie złożonych układów VLSI
- Badania w medycynie i genetyce
- Zagadnienia biologii molekularnej
- Obliczenia w astronomii i kosmologii
- Modele przepływów turbulentnych
- Modelowanie eksplozji nuklearnych
- Obliczenia metodą elementów skończonych
- Analiza danych eksperymentalnych fizyki wysokich energii
- Przeszukiwanie wielkich baz danych

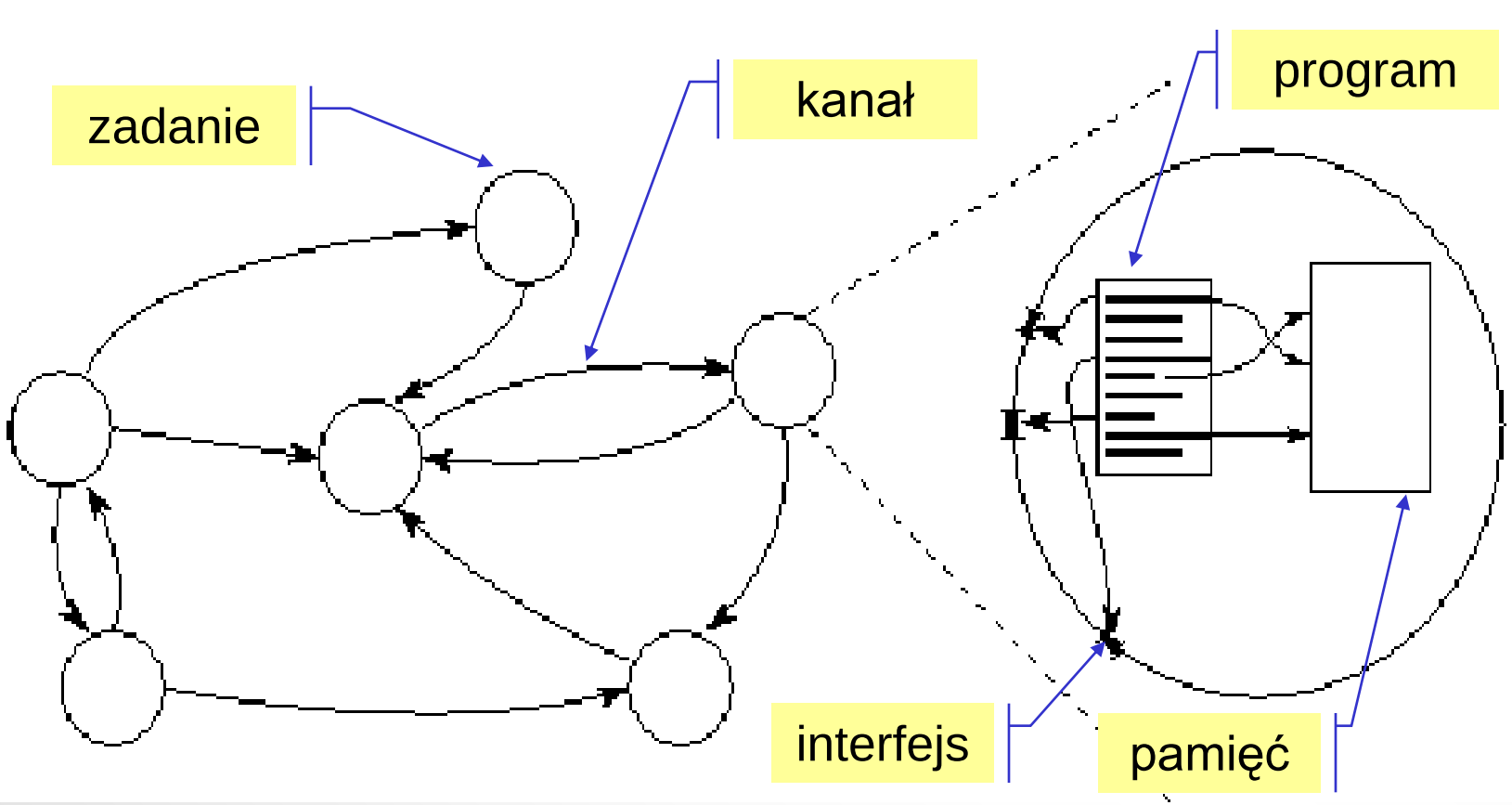
Modele programowania

- Architektura von Neumanna – model sekwencyjny (*fetch, load , execute*)
- Modularyzacja – języki wysokiego poziomu, programowanie strukturalne, obiektowe
- Równoległość – dodatkowe źródło złożoności, konieczny nowy poziom abstrakcji

Model PR zadanie-kanał

- Problem równoległy składa się z jednego lub więcej zadań, wykonywanych współbieżnie.
- Zadanie (proces) stanowi program sekwencyjny z pamięcią lokalną oraz zespołem interfejsów.
- Zadanie może (między innymi) wysyłać i odbierać komunikaty, tworzyć nowe zadania i zakończyć własne działanie.

Model zadanie-kanał



Model PR zadanie-kanał

- Operacja *send* jest asynchroniczna – wykonuje się natychmiast.
- Operacja *receive* jest synchroniczna – blokuje zadanie aż do nadejścia komunikatu.
- Porty we/wy interfejsów mogą być połączone kolejkami komunikatów – kanałami; kanały mogą być tworzone i niszczone dynamicznie.
- Zadania mogą być przydzielane procesorom na różne sposoby.

Właściwości modelu zadanie-kanał

- Wydajność – bezpośrednie odwzorowanie na architekturę równoległą
- Niezależność mapowania (lokalność) – rezultat obliczenia nie zależy od lokalizacji zadania
- Skalowalność – ilość zadań płynnie dostosowywana do ilości procesorów
- Modułowość – oddziaływania tylko poprzez jasno zdefiniowane interfejsy
- Determinizm – wyznacza go jednoznaczność nadawcy/ odbiorcy i blokowanie przy odbiorze

Model przekazywania komunikatów

***Message passing*:**

- Każde zadanie jest identyfikowane przez unikalną nazwę (liczbę).
- Zadania komunikują się poprzez wysyłanie/odbiór wiadomości.
- Zamiast kanału wskazywana jest nazwa nadawcy/odbiorcy komunikatu.

Model współdzielonej pamięci

***Shared memory*:**

- Zadania mają asynchroniczny dostęp do wspólnej przestrzeni adresowej.
- Dostęp do pamięci jest kontrolowany poprzez mechanizmy typu semaforów, zamków.
- Brak konieczności *explicite* definiowania mechanizmu komunikacji; trudności w zagwarantowaniu determinizmu.

Model równoległości danych

***Data Parallelism (loop-level parallelism)*:**

- Zdolność algorytmu do wykonywania współbieżnie takich samych operacji na poszczególnych elementach struktur danych
- Zachodzi konieczność właściwej dystrybucji danych pomiędzy zadania.
- Przykład: High Performance Fortran

Projektowanie algorytmów równoległych

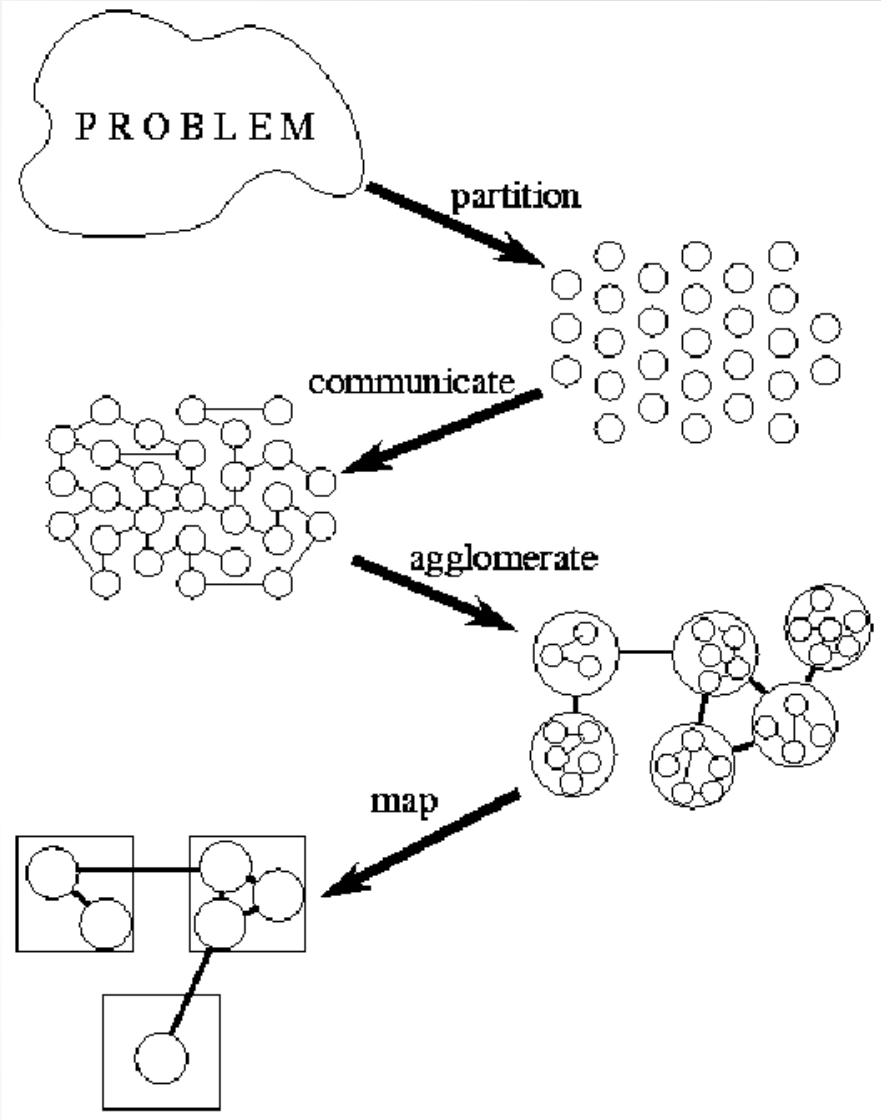
- Optymalne rozwiązanie równoległe nie musi wynikać bezpośrednio z dostępnych rozwiązań sekwencyjnych.
- Konieczne jest podejście metodologiczne analizujące w pierwszej kolejności aspekty niezależne od sprzętu takie jak współbieżność.
- Rozstrzyganie problemów zdeterminowanych sprzętowo w końcowym etapie programowania.

Projektowanie...

- Rozłożenie procesu projektowania na cztery etapy (*PCAM*):
  - Dekompozycja (*Partitioning*),
  - Komunikacja (*Communication*),
  - Scalanie (*Agglomeration*),
  - Odwzorowanie (*Mapping*).
- Dwa pierwsze etapy koncentrują się na współbieżności i skalowalności; dwa pozostałe na lokalności i wydajności.

Metodologia PCAM

- *Partition*
- *Communicate*
- *Agglomerate*
- *Map*



Dekompozycja (Pcam)

- Celem dekompozycji jest ujawnienie możliwości równoległego wykonywania programu.
- Pożądane jest uzyskanie możliwie drobnoziarnistego (*fine-grained*) podziału problemu na dużą liczbę małych zadań.
- W dalszych etapach podział ten nie musi być w pełni zrealizowany.

Rodzaje dekompozycji

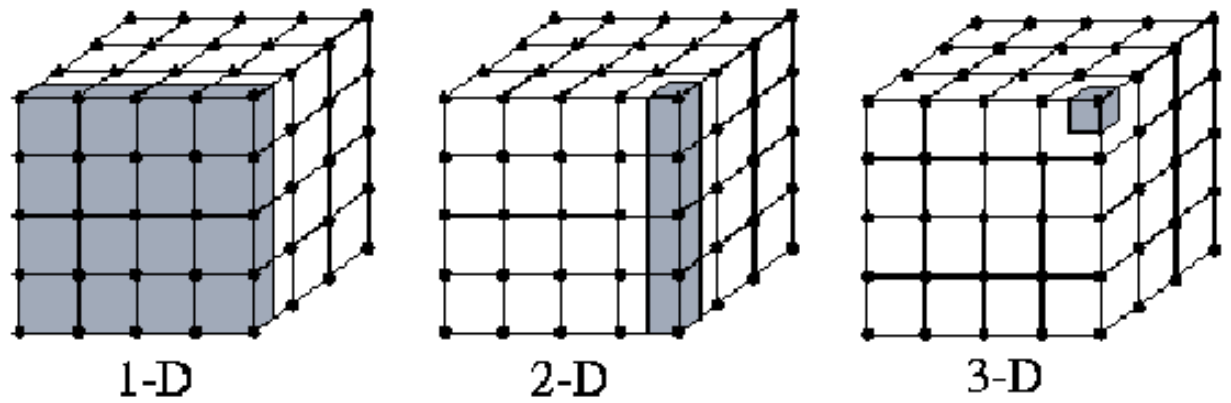
Optymalny podział dotyczy zarówno wymaganych obliczeń jak i danych.

- Zorientowanie na podział danych:
  - dekompozycja domenowa.
- Zorientowanie na podział obliczeń:
  - dekompozycja funkcjonalna.

Dekompozycja domenowa

- Podział danych na małe fragmenty
- Podział obliczeń poprzez związanie poszczególnych operacji z danymi, na których one działają
- Podzielone dane mogą stanowić dane wejściowe, rezultat bądź wyniki pośrednie działania programu
- Predestynowane do podziału są największe bądź najczęściej wykorzystywane struktury danych
- Każdy etap obliczeń może wymagać innego schematu dekompozycji danych

Przykład



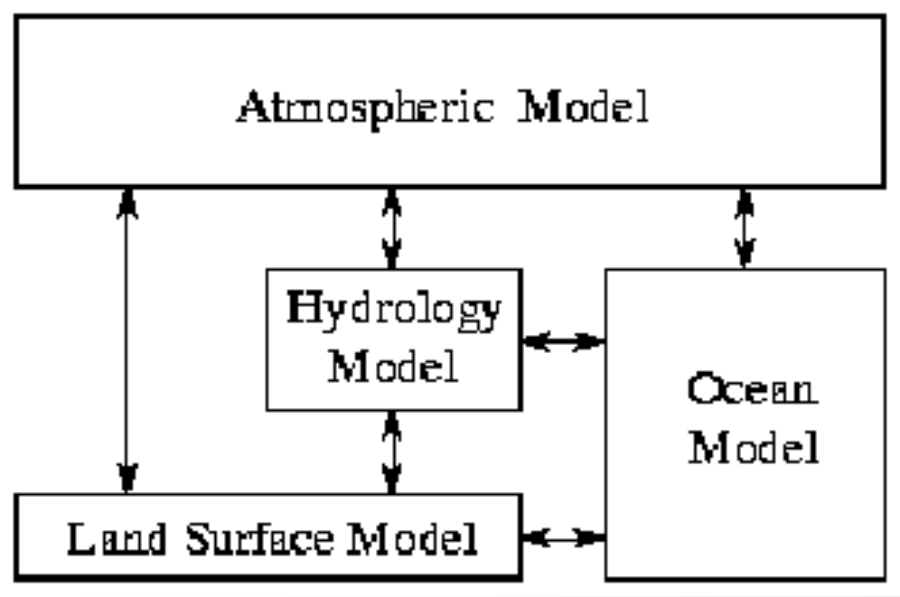
Siatka regularna 3D

- Obliczenia powtarzane dla każdego węzła
- Naturalne są 3 poziomy dekompozycji
- W najdalej posuniętym przypadku (3-D) dane i obliczenia są związane z pojedynczym węzłem
- Oferuje to największą elastyczność – pierwszy etap

Dekompozycja funkcjonalna

- Podział obliczeń na rozłączne fragmenty
- Zbadanie zależności od danych
  - rozłączność danych – pełna dekompozycja
  - duże pokrywanie się danych – sygnał sugerujący zastosowanie dekompozycji domenowej
- Pozwala uchwycić strukturę problemu nieoczywistą z punktu widzenia analizy danych
- Podział obliczeń może się łączyć z podziałem kodu – uproszczenie struktury projektu

Przykład



Modelowanie klimatu

- Podział na komponenty (zrównoleglenie funkcjonalne)
- Poszczególne z nich mogą być dalej dekomponowane np. domenowo

Kryteria dekompozycji

- Minimum o rząd wielkości więcej możliwych zadań niż procesorów
- Minimum nadmiarowych obliczeń i danych
- Porównywalny rozmiar zadań
- Proporcjonalność liczby możliwych zadań do rozmiaru problemu
- Pożądana wielowariantowość dekompozycji

Komunikacja (pCam)

- Obliczenia danego zadania (konsumenta) wymagają danych przesłanych od innego zadania (producenta).
- Dekompozycja funkcjonalna generuje zwykle proste schematy komunikacyjne.
- Dekompozycji domenowej towarzyszą najczęściej złożone struktury komunikacyjne.

Schematy komunikacyjne

- Lokalne – Globalne
- Strukturalne – Niestrukturalne
- Statyczne – Dynamiczne
- Synchroniczne – Asynchroniczne

Komunikacja lokalna

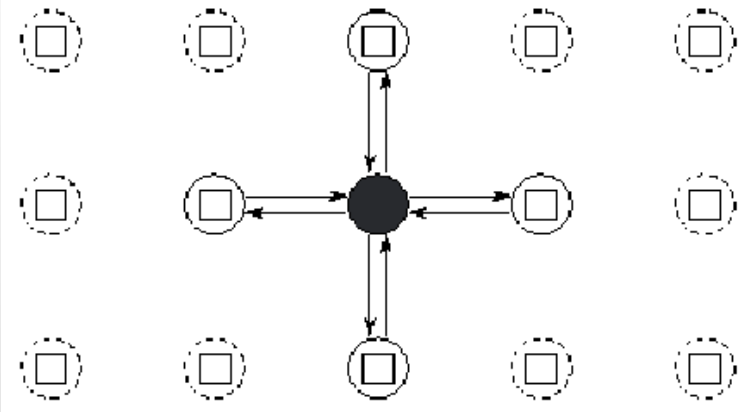
- Obliczenia danego zadania wymagają danych od niewielu innych zadań
- Nieskomplikowane określenie kanałów producent-konsument
- Przykład: aktualizacja Jacobiego kontra Gaussa-Seidela na siatce 2-wymiarowej

Jacobi:

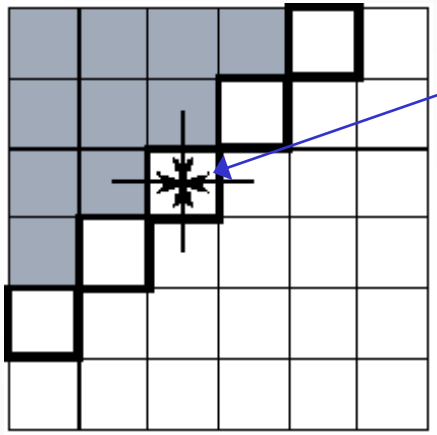
$$x_{i,j}^{(t+1)} = \frac{4x_{i,j}^{(t)} + x_{i-1,j}^{(t)} + x_{i+1,j}^{(t)} + x_{i,j-1}^{(t)} + x_{i,j+1}^{(t)}}{8}$$

Gauss -Seidel:

$$x_{i,j}^{(t+1)} = \frac{4x_{i,j}^{(t)} + x_{i-1,j}^{(t+1)} + x_{i+1,j}^{(t)} + x_{i,j-1}^{(t+1)} + x_{i,j+1}^{(t)}}{8}$$

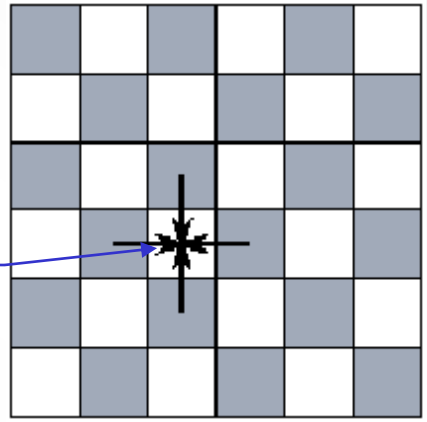


- łatwe do zrównoleglenia
- niezależna aktualizacja
- gorsza zbieżność



- tylko 5 węzłów gotowe
- podejście sekwencyjne

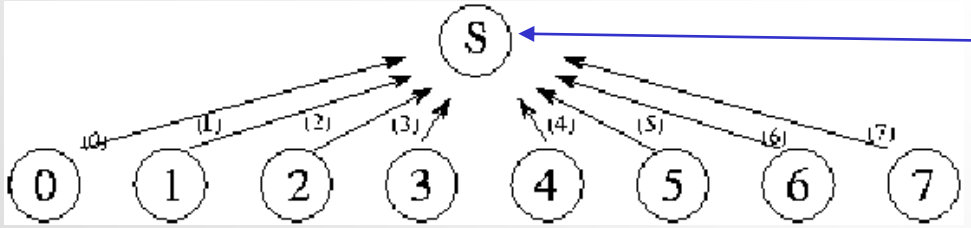
- schemat *red-black*
- możliwe zrównoleglenie



Komunikacja globalna

- Uczestnictwo wielu zadań
- Niska wydajność prostego modelu producent – konsument
- Przykład: równoległa redukcja
  - zcentralizowane (proces zarządcy uczestniczy we wszystkich komunikacjach)
  - sekwencyjne (brak współbieżności)

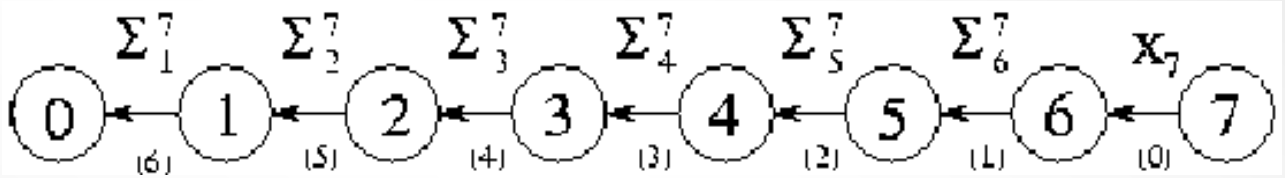
$$S = \sum_{i=0}^{N-1} X_i$$



zarządca

Strategia *Divide and Conquer*

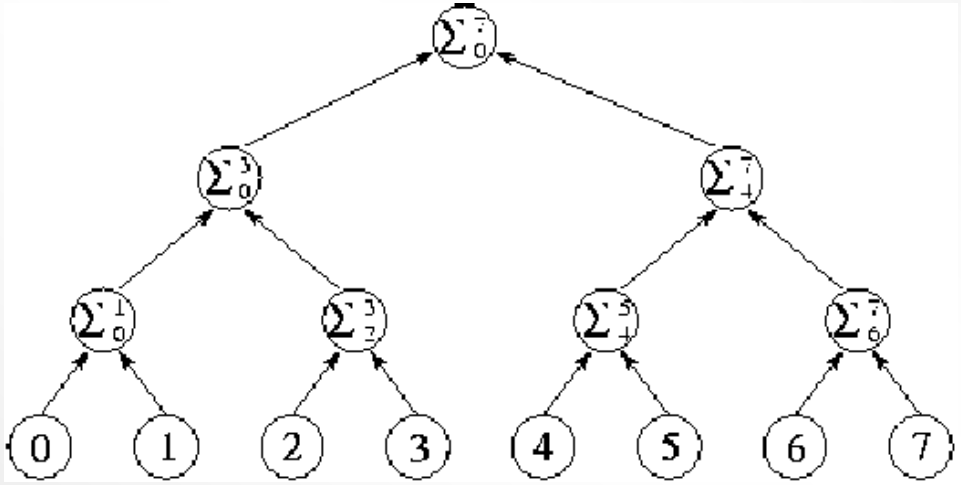
- Dystrybucja obliczeń i komunikacji



$$S_i = X_i + S_{i-1}$$

- Zastosowanie schematu rekurencyjnego:
  - podział zadania na N równych części,
  - powtarzanie schematu aż do uzyskania niepodzielnych fragmentów

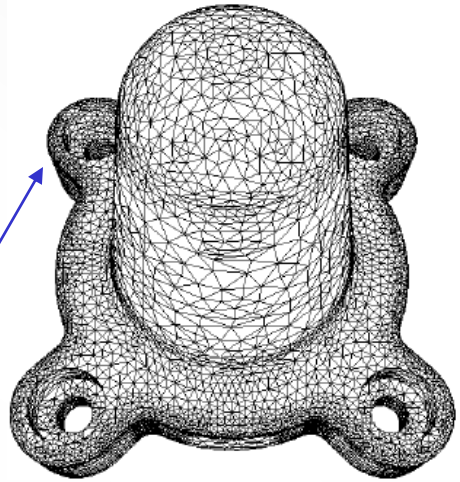
$$\sum_{i=0}^{2^n-1} = \sum_{i=0}^{2^{n-1}-1} + \sum_{i=2^{n-1}}^{2^n-1}$$



Komunikacja niestrukturalna i dynamiczna

- Gęstość komunikacji zależna od istotności danego fragmentu rozwiązania
- Prostota koncepcyjna na wczesnym etapie
- Trudności w optymalnym połączeniu dystrybucji zadań i minimalizacji komunikacji
- Przykład: metody elementów skończonych

różna lokalna gęstość siatki



Komunikacja asynchroniczna

- Producenci nie są w stanie określić kiedy konsumenci będą potrzebować danych.
- Konsumenci muszą w sposób jawny zgłaszać żądania otrzymania danych.
- Rozproszony dostęp do danych:
  - dane rozproszone między zadania obliczeniowe; zadania okresowo odpytują pozostałe (*polling*),
  - dane rozproszone między dedykowane zadania,
  - współdzielona pamięć (*shared memory*).

Kryteria komunikacji

- Pożądane zrównoważenie obciążenia komunikacyjnego między zadaniami
- Minimalizacja ilości partnerów
- Współbieżność komunikacji determinuje skalowalność
- Celem pozostaje współbieżność obliczeń

Scalanie (pcAm)

- Etap mający na celu dopasowanie ilości i rozkładu zadań do specyfiki sprzętowej konkretnego systemu komputerowego.
- Zastąpienie potencjalnie dużej liczby małych zadań mniejszą liczbą zadań większych.
- Określenie zasadności powielania danych i obliczeń pomiędzy zadaniami.
- Liczba zadań nadal *może być* większa od liczby dostępnych procesorów.

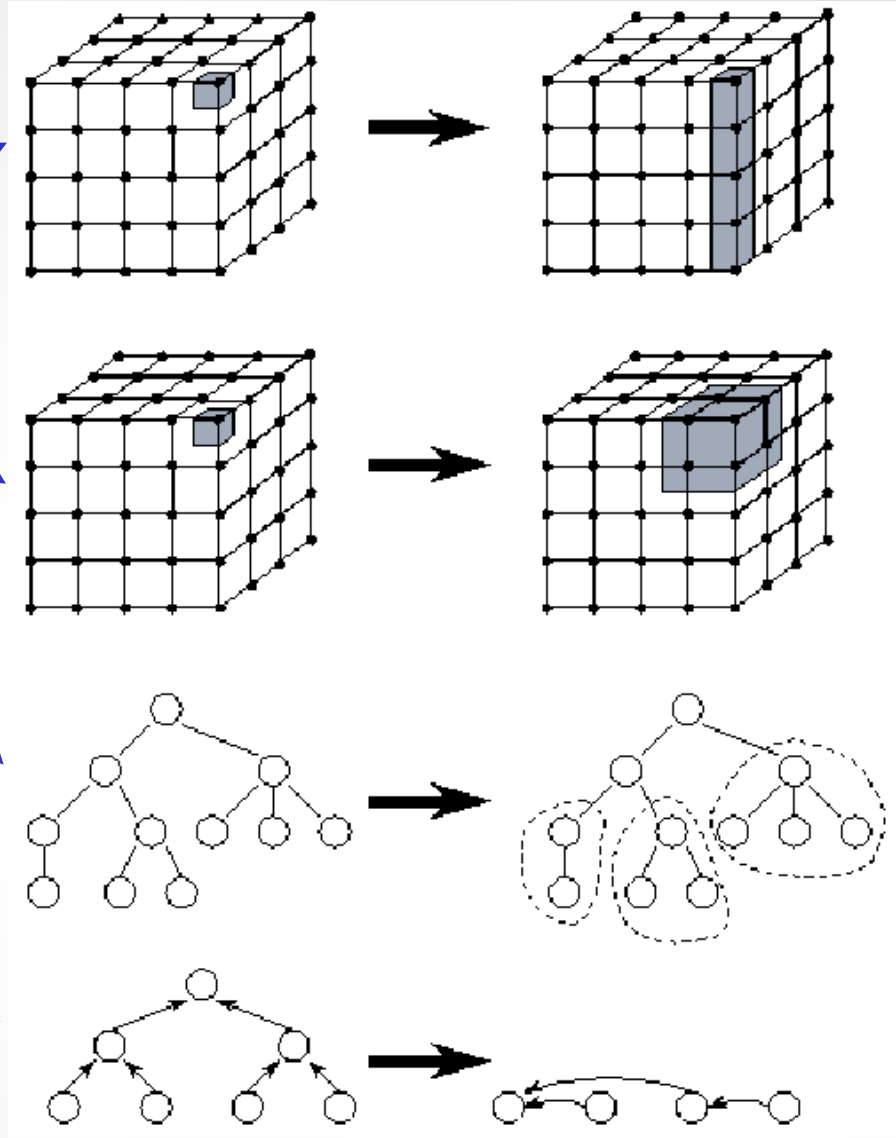
Przykłady:

Wzrost rozmiaru zadania dzięki zmniejszeniu wymiaru dekompozycji

Zwiększona ziarnistość dzięki połączeniu sąsiednich zadań

Połączenie gałęzi w schemacie *divide-and-conquer*

Połączenie węzłów np. w schemacie drzewa poszukiwań



Cele scalania i powielania

- Zmniejszenie kosztów komunikacji poprzez zwiększenie ziarnistości obliczeń i komunikacji
- Zachowanie elastyczności w odniesieniu do skalowania i mapowania
- Zmniejszenie kosztów inżynierii oprogramowania (*trudności problemu programistycznego*)

Zwiększanie ziarnistości

- Im mniej przesyłanych danych tym lepiej – więcej czasu na obliczenia
- Im mniej komunikatów (większe porcje danych) tym lepiej – stały koszt otwarcia kanału
- Dynamiczne tworzenie zadań także pociąga koszty czasowe

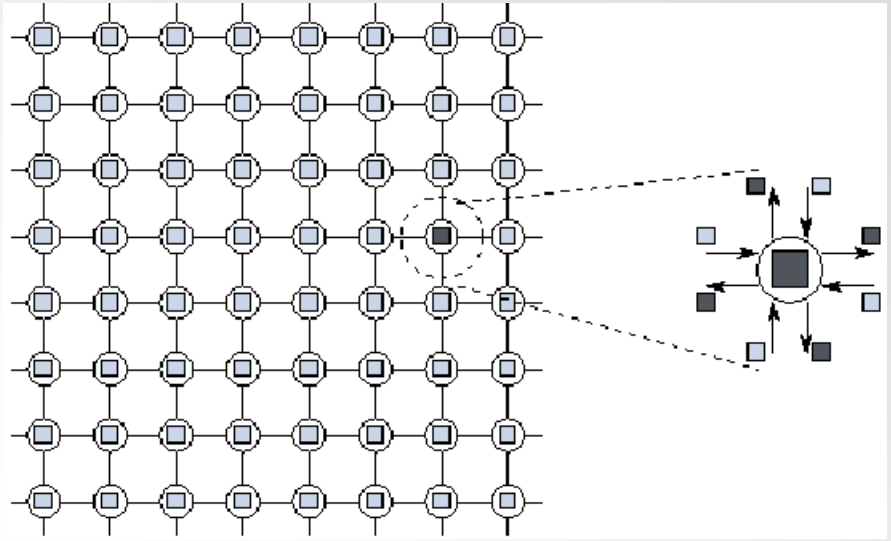
Efekty powierzchni do objętości

Surface-to-Volume Effects

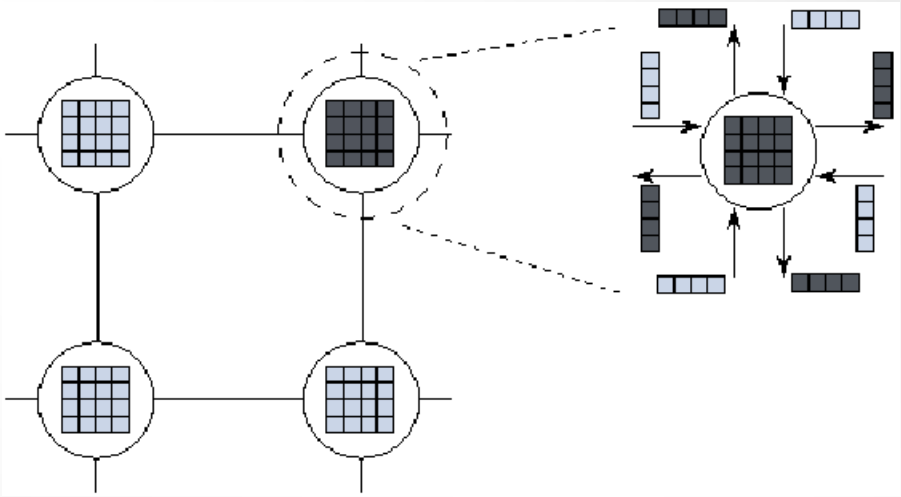
- Ilość komunikacji zadania jest proporcjonalna do powierzchni domeny
- Ilość obliczeń jest proporcjonalna do objętości domeny (węzła)
- W problemach wielowymiarowych stosunek komunikacja/obliczenia maleje ze wzrostem pojedynczego zadania
- Wyżej wymiarowe dekompozycje są zwykle bardziej wydajne – scalanie lepiej wykonywać we wszystkich wymiarach jednocześnie

Przykład

Siatka 8x8 podzielona na 64 zadania generuje 64x4=256 komunikacji; wymiana 256 danych



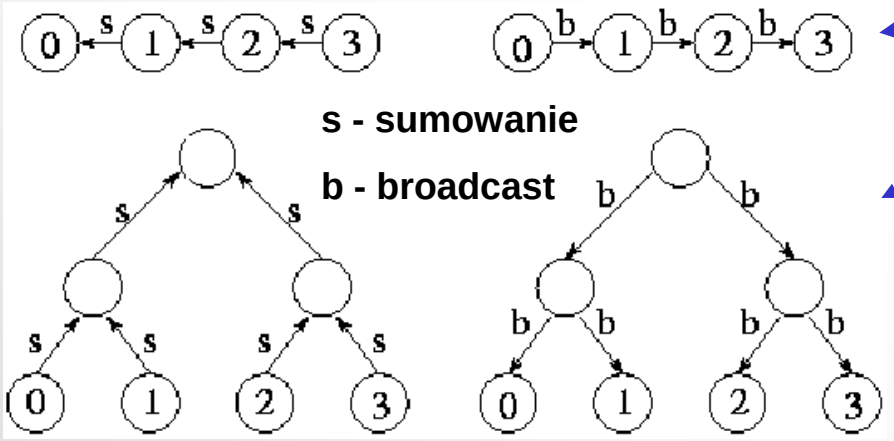
Siatka 8x8 podzielona na 4 zadania o większej ziarnistości generuje 4x4=16 komunikacji; wymiana 16x4=64 danych



Dwuwymiarowy problem różnicowy – węzły z 4 sąsiadami

Powielanie obliczeń

- Zmniejszenie ilości komunikacji okupione powieleniem obliczeń
- Przykład: równoległa redukcja
  - redystrybucja sumy do wszystkich zadań



Szereg: *2(N-1)* kroków

Drzewo: *2 log N* kroków

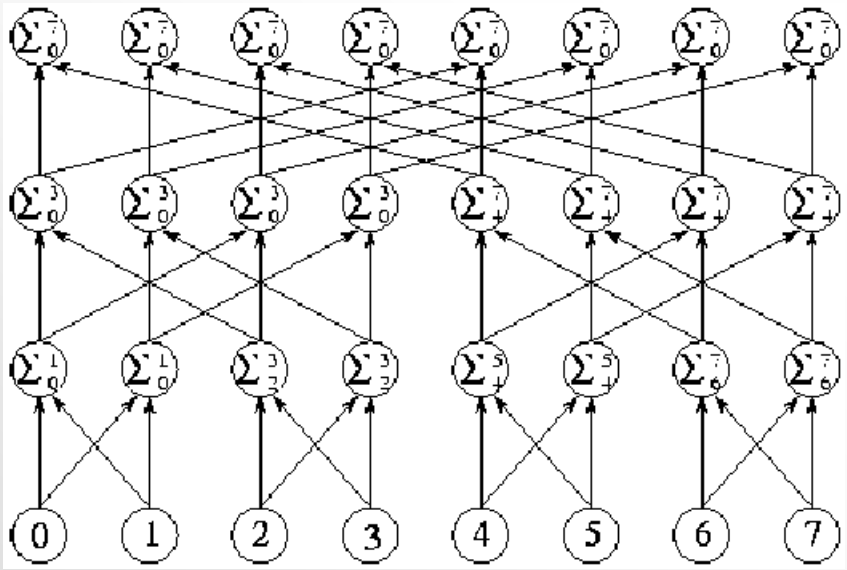
- brak nadmiarowych obliczeń
- dużo komunikacji

Przykład

- Współbieżne obliczanie sumy

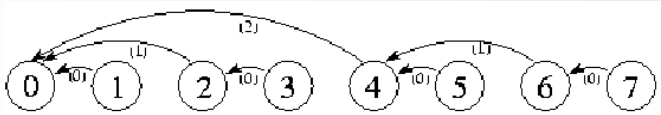
Pierścień: modyfikacja szeregu – wszystkie zadania wykonują ten sam algorytm: *N-1* kroków kosztem $(N-1)^2$ nadmiarowych dodawań i $(N-1)^2$ nadmiarowych komunikacji.

Motyl: wiele drzew sumujących współbieżnie: *log N* kroków – sumowanie w *N log N* operacjach

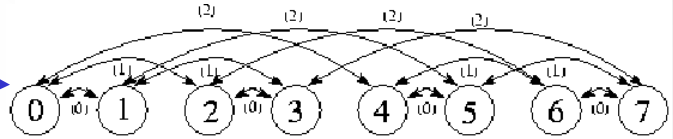


Schematy komunikacyjne:

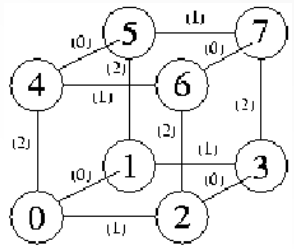
drzewo



motyl



hipersześcian (*hypercube*)
odpowiednik motyla



Unikanie komunikacji

- Scalanie jest zawsze pożądane gdy analiza wskazuje, że komunikacja uniemożliwia współbieżne działanie.
- Jeśli zadania w danym kroku nie wykonują się współbieżnie to ich scalenie generuje uproszczone schematy komunikacyjne.

Zachowywanie elastyczności

- Możliwość podziału problemu na zmienną w dużym zakresie liczbę zadań jest warunkiem skalowalności.
- Dopuszczalne jest nakładanie się obliczeń i komunikacji kilku zadań na jednym procesorze.
- Elastyczność mapowania na procesory.

Ograniczanie kosztów inżynierii oprogramowania

- Strategia dekompozycji i scalania wpływa znacząco na nakłady niezbędne do zaadaptowania istniejących programów sekwencyjnych do obliczeń równoległych.
- Współpraca różnych modułów dużego systemu obliczeniowego może wymuszać uproszczenie struktur dekompozycji.

Kryteria scalania

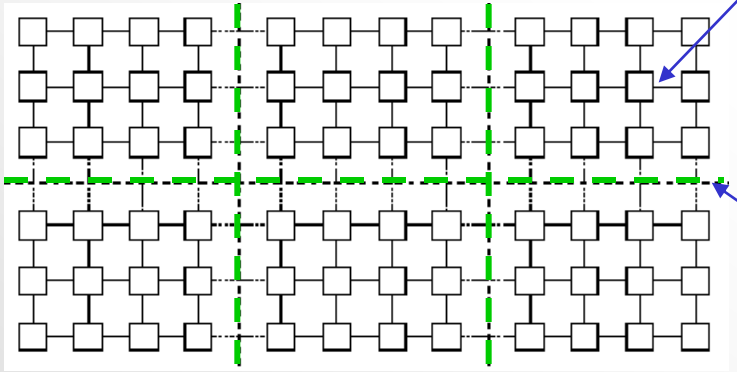
- Redukcja komunikacji przez zwiększenie lokalności
- Przewaga zysków nad stratami wynikającymi z nadmiarowych obliczeń
- Powielanie danych nie ograniczające skalowalności (rozmiaru problemu)
- Porównywalność rozmiaru zadań bardziej istotna wraz ze zmniejszaniem się ich liczby
- Zachowanie skalowania liczby zadań z rozmiarem problemów
- Utrzymanie wystarczającej współbieżności na docelowym systemie komputerowym
- Zachowanie możliwości dalszego zmniejszenia liczby zadań bez znaczących kosztów
- Wpływ nakładów niezbędnych do zrównoleglenia istniejącego kodu sekwencyjnego

Odwzorowanie (pcaM)

- Stanowi sposób przypisania zadań do zasobów (procesorów) systemu komputerowego.
- Nie dotyczy maszyn 1-procesorowych oraz tych ze współdzieloną pamięcią i automatycznym systemem zarządzania procesami.
- Możliwe dwa podejścia do problemu minimalizacji czasu obliczeń:
  - umieszczanie współbieżnych zadań na oddzielnych procesorach,
  - umieszczanie zadań często komunikujących się ze sobą na tym samym procesorze.
- Mapowanie pozostaje trudnym zadaniem (klasa NP-zupełna).

Mapowanie...

- W przypadku regularnych problemów o określonej liczbie jednakowych zadań mapowanie jest stosunkowo proste.
  - Minimalizowana jest komunikacja między procesorami



Zdania scalone do obsługi przez jeden procesor

Granice dekompozycji między procesory

Algorytmy równoważenia obciążeń

Load-Balancing

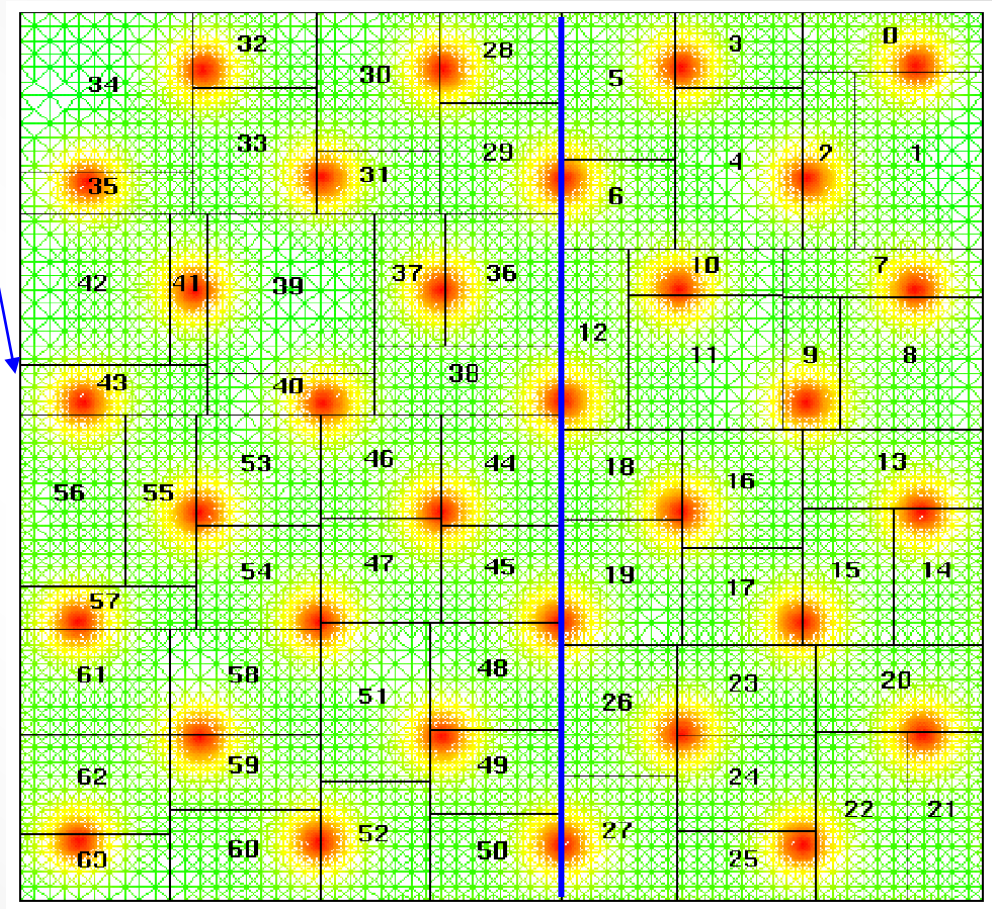
- Stosowane są do optymalizacji algorytmów dekompozycji domenowej o zróżnicowanym obciążeniu zadań i/lub niestrukturalnym charakterze komunikacji.
- Wykorzystują metody heurystyczne.
- Czas wykonania algorytmu równoważenia nie powinien przewyższać spodziewanych zysków z optymalizacji;
  - podejście probabilistyczne kontra analiza struktury.

Rekursywna bisekcja

- Podział domeny minimalizujący ilość kanałów komunikacyjnych przecinających granice poddomen.
  - Podejście *Divide-and-Conquer;*
  - *Recursive coordinate bisection* – cięcie zawsze wzdłuż dłuższej osi współrzędnych;
  - *Unbalanced recursive bisection* – redukcja komunikacji poprzez zachowanie stosunku rozmiarów poddomen (cięcia na *P-1* poddomen o obciążeniach *n/P* i *(P-n)/P* n=1,2,... z wyborem minimalizującym stosunek rozmiarów.

Przykład: *unbalanced recursive bisection*

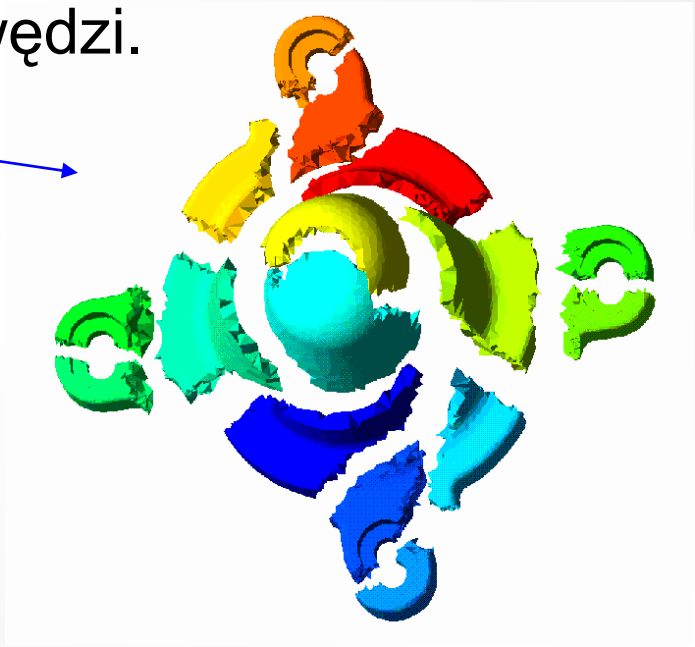
Nieregularna siatka
Mapowanie na 64 procesory problemu symulacji nadprzewodnictwa. Zwiększone obciążenie obliczeniowe uwidocznione barwą. Pierwszy podział wykonany pionową niebieską linią.



Algorytmy oparte na teorii grafów (siatki MES)

- *Recursive graph bisection* – siatka traktowana jako graf; wybierane 2 punkty najbardziej odległe; podział pozostałych na 2 części wg położenia względem krańcowych;
- *Recursive spectral bisection* – podział minimalizujący liczbę cietych krawędzi.

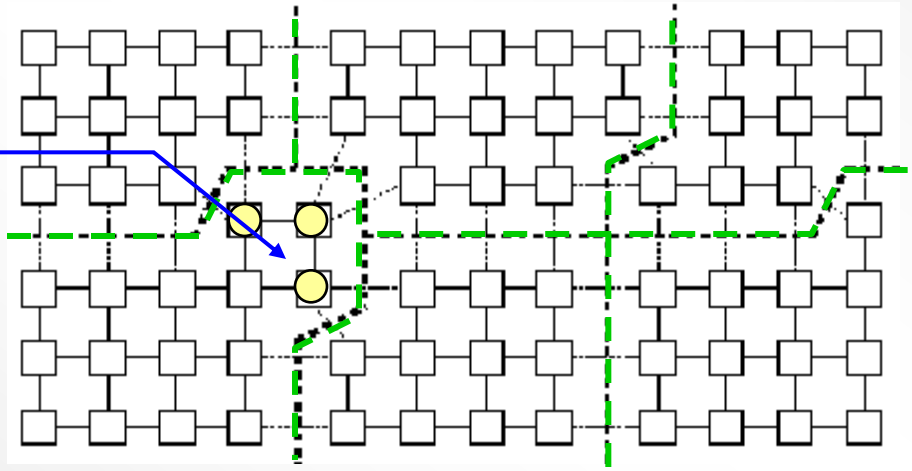
Algorytm *recursive spectral bisection* zastosowany do siatki metody elementu skończonego



Algorytmy lokalne

- Równoważenie wykonywane w oparciu o informację pochodzącą tylko od sąsiednich procesorów
  - Porównywanie obciążenia przez sąsiadów na siatce
  - Przekazywanie nadmiaru obliczeń mniej obciążonemu sąsiadowi

Węzły przekazane od sąsiednich procesorów



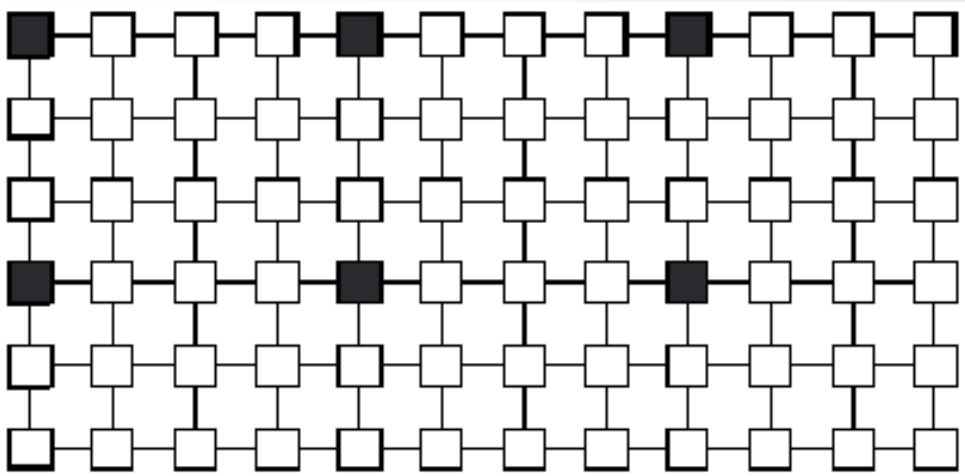
Metody probabilistyczne

- Losowe alokowanie zadań z dużej puli pomiędzy procesory
- Zaleta: skalowalność i niski koszt
- Wada: nielokalna komunikacja
- Efektywne gdy jest niewiele komunikacji między zadaniami lub są one z założenia nielokalne.

Mapowanie cykliczne

- Stosowane jeżeli obciążenie węzła zmienia się i zachodzi skokowa lokalna zmiana poziomu obciążenia.
- Każdy z *P* procesorów otrzymuje co *P*-te zadanie wg określonego schematu.

Odwzorowanie na 12 procesorów:
- zaznaczone węzły przydzielone do danego procesora,
- komunikacja z sąsiednimi węzłami obsługiwanymi przez inne procesory.



Algorytmy kolejkowania zadań

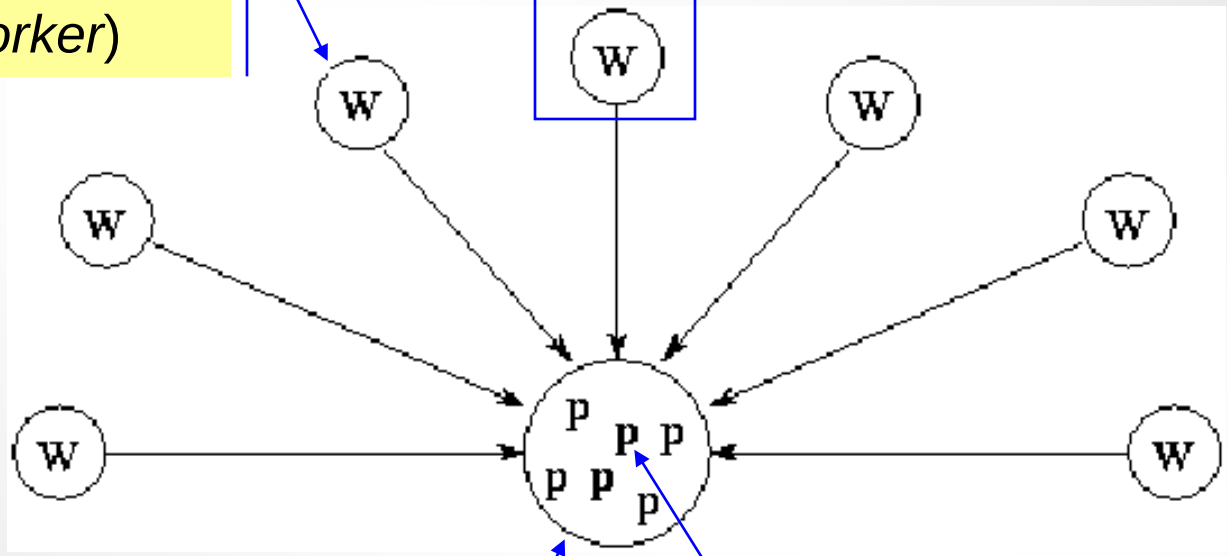
Task-scheduling

- W przypadku dekompozycji funkcjonalnej pojawia się problem koordynacji krótkich zadań kontaktujących się tylko przy starcie i zakończeniu:
  - przydział zadań do oczekujących procesorów.
- Interpretacja zadań (problemów) jako struktur rozwiązywanych przez zadania wykonawcze przypisane do procesorów.
- Krytyczną pozostaje strategia przydziału problemów do wykonawców.

Zarządca – Wykonawca

- Centralne zadanie zarządca przydziela problemy.
- Wykonawcy cyklicznie zgłaszają się do zarządcy, a następnie wykonują zlecone problemy.
- Wykonawcy mogą także przekazywać zarządcy nowe problemy do zlecenia.
- Efektywność zależy od ilości wykonawców i względnych kosztów otrzymania i wykonania problemu.

Przykład



procesor

wykonawca
(*worker*)

zarządca
(*manager*)

problem oczekujący
na przydział

Hierarchiczny
Zarządca – Wykonawca

- Wariant poprzedniego schematu
- Podział wykonawców na rozłączne podzbiory, każdy z własnym podrzędnym zarządcą.
- Podrzędni zarządcy komunikują się okresowo z zarządcą głównym i między sobą w celu zrównoważenia obciążenia podlegających im zbiorów procesorów.

Schematy zdecentralizowane

- Brak centralnego zarządcy
- Każdy procesor obsługuje oddzielną pulę zadań.
- Wolni wykonawcy żądają zadań od innych procesorów.
- Asynchroniczny dostęp do rozproszonej struktury oczekujących zadań

Wykrywanie zakończenia

Termination detection

- Trywialne w systemach zcentralizowanych
- W wersjach zdecentralizowanych problemy z synchronizacją stanu – wiadomości w przelocie (o oczekujących na wykonanie zadaniach) mogą zostać nie uwzględnione.

Kryteria odwzorowania

- Algorytm z dynamicznym tworzeniem zadań może być prostszy, ale nie bardziej wydajny.
- Centralny zarządca może podlegać przeciążeniu.
- Koszty różnych strategii dynamicznego równoważenia obciążenia
- Metody probabilistyczne i cykliczne wymagają dużej liczby zadań do osiągnięcia dobrego zrównoważenia (rząd więcej niż dostępna liczba procesorów).