PPaDS MMXX

Matus Jokay, C-503, matus.jokay@stuba.sk
Roderik Ploszek, C-512, roderik.ploszek@stuba.sk

uim.fei.stuba.sk/predmet/i-ppds

konzultacie dohodou

Vedomosti

- Naco je dobra synchronizacia
- Zakladne synchronizacne nastroje
- Rozne synchronizacne problemy

- Asynchronne, konkurentne, paralelne programovanie

Literatura

- A. B. Downey: The Little Book of Semaphores
- F. Pierfederici: Distributed Computing with Python
- A. Grama, A. Gupta. G. Karypis, V. Kumar: Introduction to Parallel Computing
- T. G. Mattson, B. A. Sanders, B. L. Massingill: Patterns for Parallel Programming

Hodnotenie

- 2x zapocet **pisomne**, nie pri PC
  - 25 bodov, 10 bodov minimum
  - 25 bodov, 10 bodov minimum
- Skuska 50 bodov, minimum 20
  - Moznost alternativneho absolvovania skusky
  - Vypracovanie brutalneho zadania
  - Jednotlivo alebo vo dvojici
  - Odovzdava sa git repozitar

Organizacia

- Prednaska 2 hod. tyzdenne (utorok 8:00)

- Cvicenia
  - Utorok 10:00 – Matus Jokay
  - Utorok 13:00 – Roderik Ploszek
  - Utorok 15:00 – Roderik Ploszek

Motivacia

Motivacia

1) Stolceky

Motivacia

1) Stolceky

2) Povala

Motivacia

1) Stolceky

2) Povala

3) Travnik

Uvod do PPaDS

Uvod do PPaDS

Francesco Pierfederici

Distributed Computing with Python

Packt Publishing Ltd.

April 2016

ISBN 978-1-78588-969-1

Uvod do PPaDS

- Prvy pocitac – 40-te roky konca minuleho tisicrocia

Uvod do PPaDS

- Prvy pocitac – 40-te roky konca minuleho tisicrocia
- Odvtedy 80 rokov… terajsie smartfony su rychlejsie nez najrychlejsi pocitac spred 20 rokov ;)

Uvod do PPaDS

- Prvy pocitac – 40-te roky konca minuleho tisicrocia
- Odvtedy 80 rokov… terajsie smartfony su rychlejsie nez najrychlejsi pocitac spred 20 rokov ;)
- Netreba nam obrovske miestnosti s klimatizaciou, pocitac strcime do vrecka…

Uvod do PPaDS

- Procesor dokaze v jednom momente spracovavat iba jednu ulohu

Uvod do PPaDS

- Procesor dokaze v jednom momente spracovavat iba jednu ulohu
- Iluzia paralelizmu – v kratkom case striedanie uloh

Uvod do PPaDS

- Procesor dokaze v jednom momente spracovavat iba jednu ulohu
- Iluzia paralelizmu – v kratkom case striedanie uloh
- Na skutocne sucasne vykonavanie viac uloh v jednom case potrebujeme viac procesorov/jadier

Uvod do PPaDS

- Procesor dokaze v jednom momente spracovavat iba jednu ulohu
- Iluzia paralelizmu – v kratkom case striedanie uloh
- Na skutocne sucasne vykonavanie viac uloh v jednom case potrebujeme viac procesorov/jadier
- V sucasnosti to uz nie je problem…

Uvod do PPaDS

- Multiprocesorove a multijadrove systemy (osobne pocitace, notebooky, netbooky, laptopy, tablety, smartfony, minipocitace, …)

Uvod do PPaDS

- Multiprocesorove a multijadrove systemy (osobne pocitace, notebooky, netbooky, laptopy, tablety, smartfony, minipocitace, …)

- Graficke karty (stovky az tisice vypoctovych uzlov), tzv. GPU (Graphics Processing Unit)

Uvod do PPaDS

- Multiprocesorove a multijadrove systemy (osobne pocitace, notebooky, netbooky, laptopy, tablety, smartfony, minipocitace, …)

- Graficke karty (stovky az tisice vypoctovych uzlov), tzv. GPU (Graphics Processing Unit)

- Pocitacove siete (Internet, siete mobilnych operatorov, lokalne siete (lan, wifi), …)

PPaDS - Definicie

- Paralelny vypocet

- Distribuovany vypocet

PPaDS - Definicie

- Paralelny vypocet

  Paralelny vypocet je sucasne vyuzitie viacerych (t.j. viac nez 1) vypoctovych uzlov na dosiahnutie ciela vypoctu

- Distribuovany vypocet

  Distribuovany vypocet je sucasne vyuzitie viacerych (t.j. viac nez 1) vypoctovych uzlov na dosiahnutie ciela vypoctu

PPaDS - Definicie

- Paralelny vypocet

  Paralelny vypocet je sucasne vyuzitie viacerych (t.j. viac nez 1) vypoctovych uzlov na dosiahnutie ciela vypoctu

- Distribuovany vypocet

  Distribuovany vypocet je sucasne vyuzitie viacerych (t.j. viac nez 1) vypoctovych uzlov na dosiahnutie ciela vypoctu

PPaDS - Definicie

- **Paralelny** vypocet (vypoctovy uzol = **CPU**)

  Paralelny vypocet je sucasne vyuzitie viacerych (t.j. viac nez 1) vypoctovych uzlov na dosiahnutie ciela vypoctu

- **Distribuovany** vypocet (vypoctovy uzol = **PC**)

  Distribuovany vypocet je sucasne vyuzitie viacerych (t.j. viac nez 1) vypoctovych uzlov na dosiahnutie ciela vypoctu

PPaDS - Definicie

- **Paralelny** vypocet
  - Na procesoroch (jadrach procesora)
  - **Multivlaknove** programy na komunikaciu typicky **ta ista (spolocna) pamat**
  - **Multiprocesove** programovanie na komunikaciu typicky **zdielana pamat**

- **Distribuovany** vypocet
  - Na pocitacoch (uzloch siete) (pocitacove farmy) na komunikaciu typicky siet
  - Na grafickych kartach (CUDA, OpenCL) na komunikaciu zbernica pocitaca (PCI)

Vyvoj distribuovanej aplikacie

1. Vyvoj jednovlaknovej (jednoprocesovej) aplikacie

Vyvoj distribuovanej aplikacie

1. Vyvoj jednovlaknovej (jednoprocesovej) aplikacie
2. Vyvoj multiprocesovej (nie multivlaknovej, hoci v zavislosti od implementacneho prostredia to moze byt jeden z medzikrokov) aplikacie

Vyvoj distribuovanej aplikacie

1. Vyvoj jednovlaknovej (jednoprocesovej) aplikacie
2. Vyvoj multiprocesovej (nie multivlaknovej, hoci v zavislosti od implementacneho prostredia to moze byt jeden z medzikrokov) aplikacie
3. Vyvoj distribuovanej aplikacie

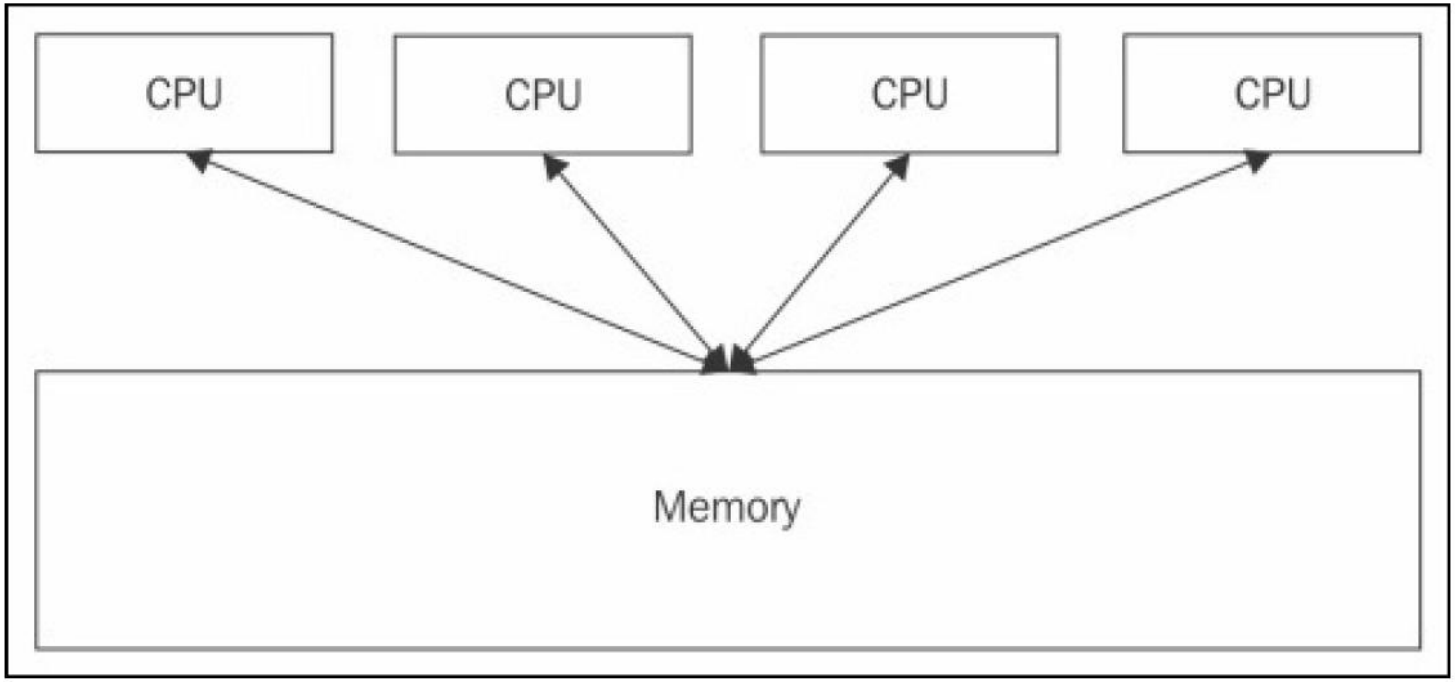
PPaDS – pozor na udaje!

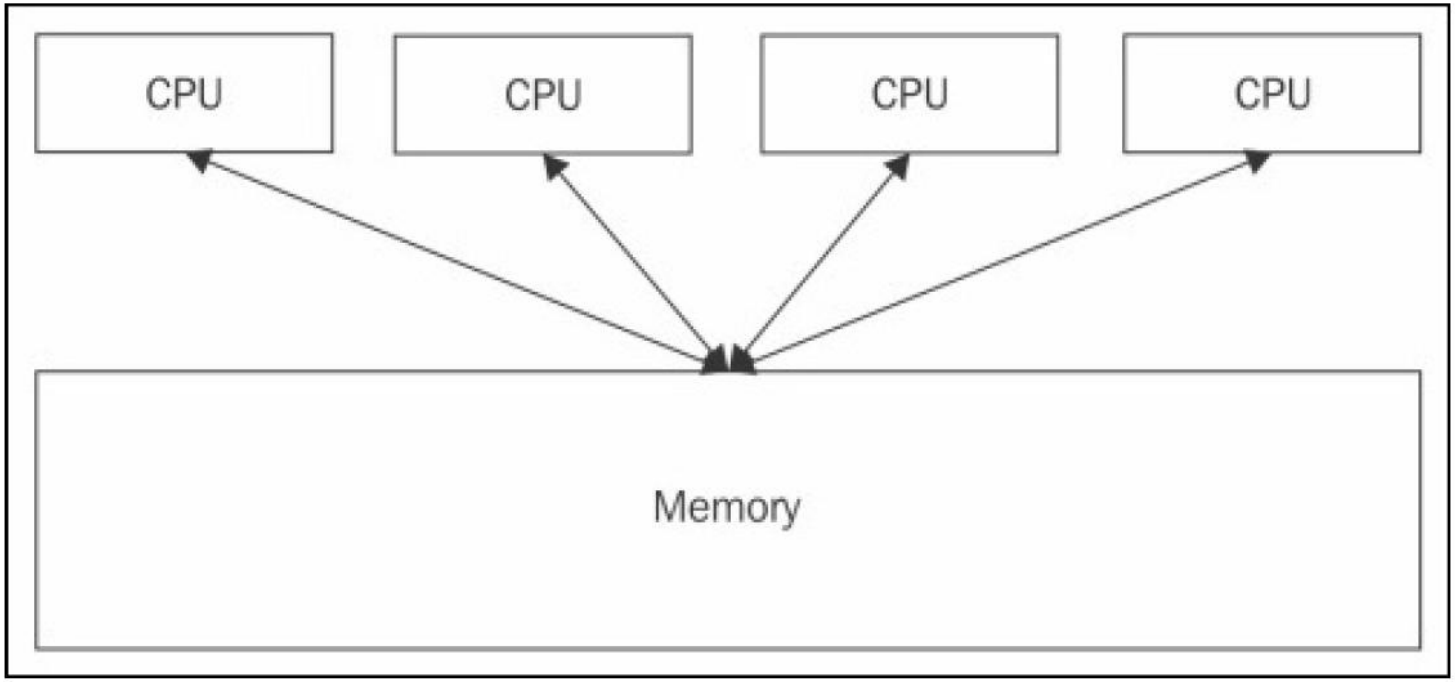
- Skutocnym uzkym hrdlom vypoctov zvacsa (zavisi od typu aplikacie) byvaju samotne udaje, nie CPU

- Niekedy staci zdielany suborovy system (napr. NFS na unixovych systemoch), inokedy zdielana databaza alebo posielanie sprav…

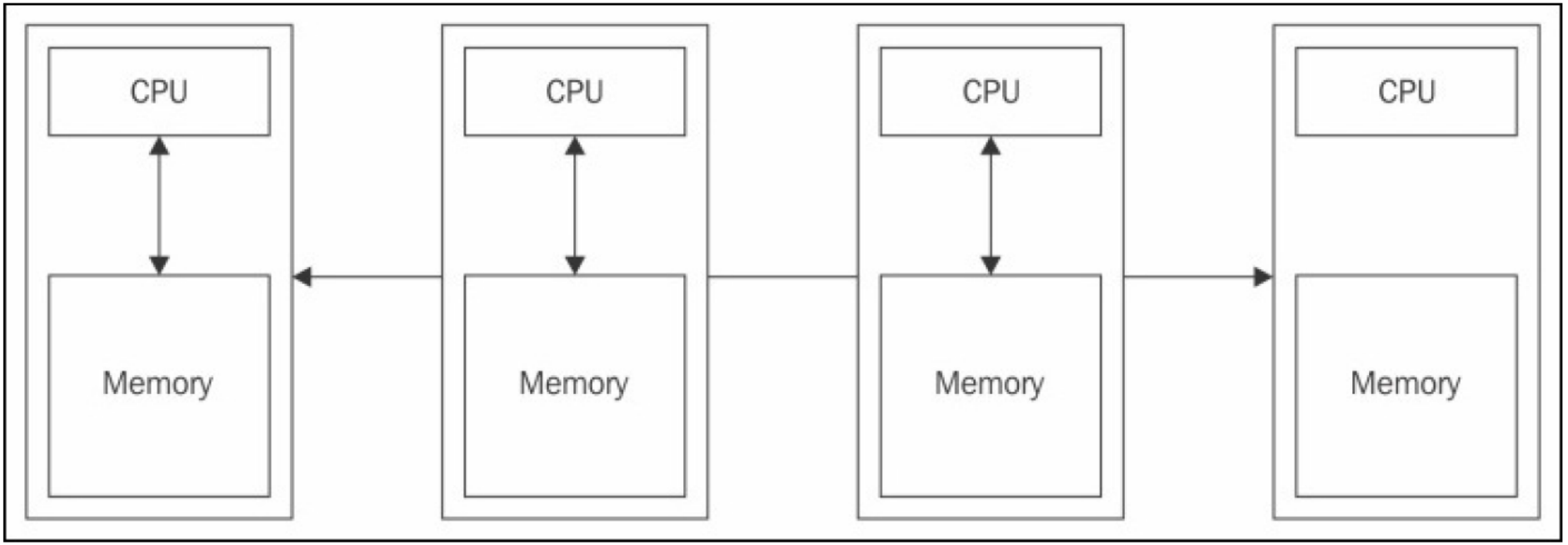
Zdielana versus distribuovana pamat

- Fyzicke umiestnenie vypoctovych zdrojov ma velky dopad na efektivitu vypoctu
- Najvacsi rozdiel medzi PP a DP je v pouzitej pamatovej architekture a v sposobe pristupu k datam
  - PP zvacsa vyuziva ten isty pamatovy priestor
  - DP zvacsa vyuziva distribuovany model pamate

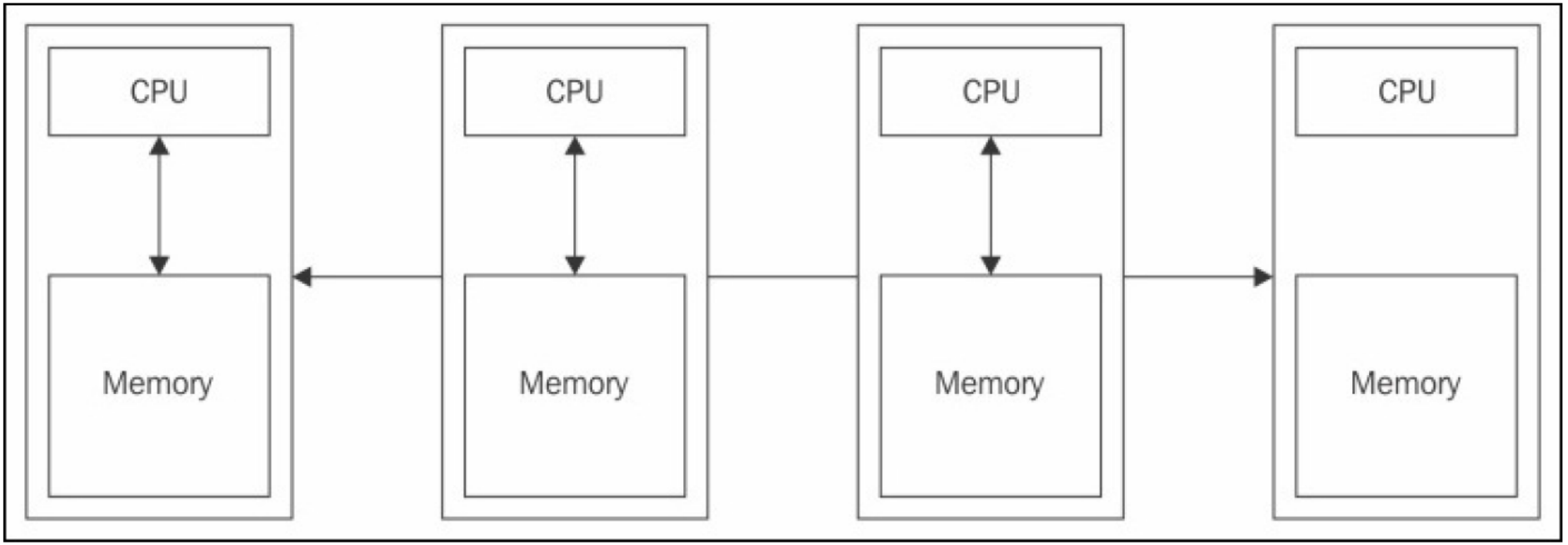

CPU
CPU
CPU
CPU
Memory

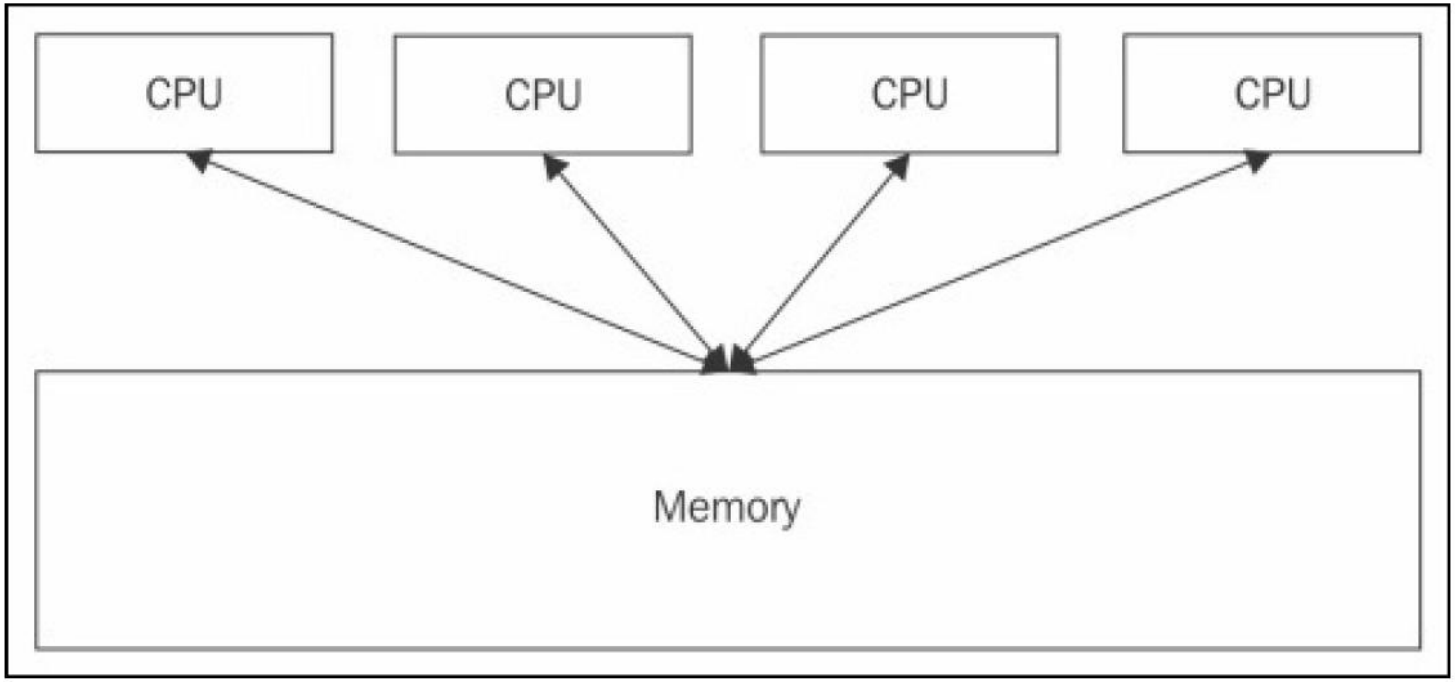


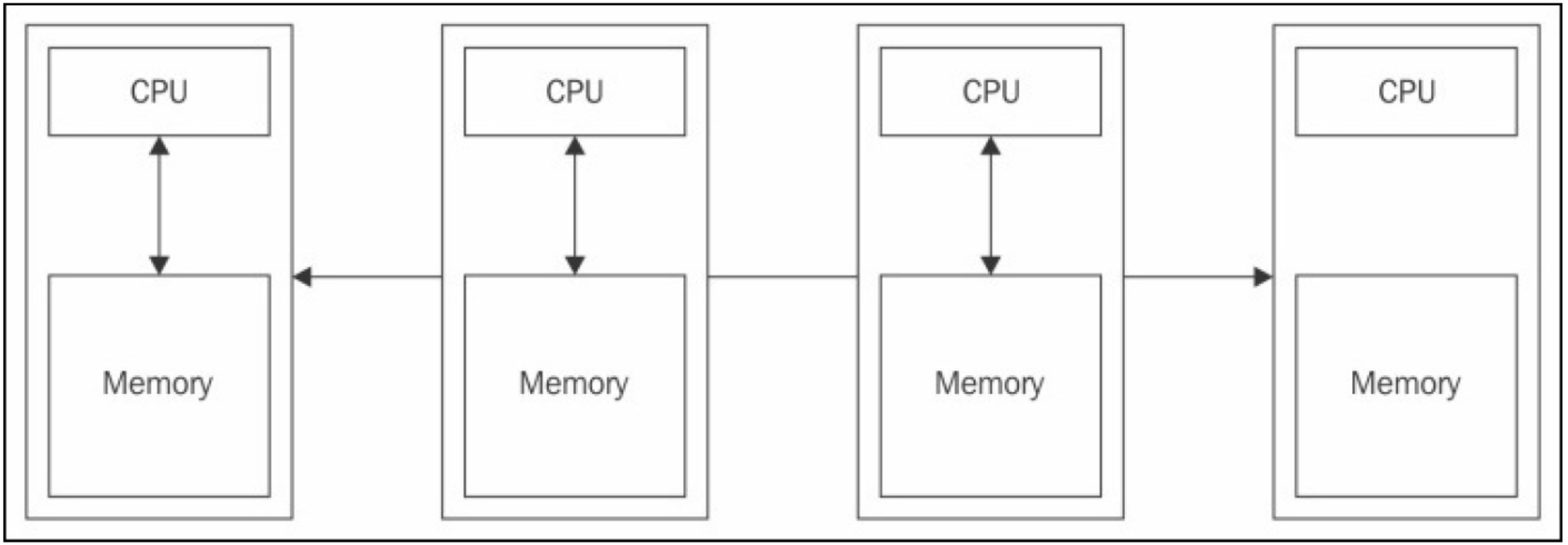
- Multivlaknove programovanie nativne vyuziva tuto architekturu
- Zdielanie pamate dosiahnutelne aj v multiprocesovom programovani
- Problematicke, ale nie nemozne aj v distribuovanom programovani (pomocou tzv. middleware)

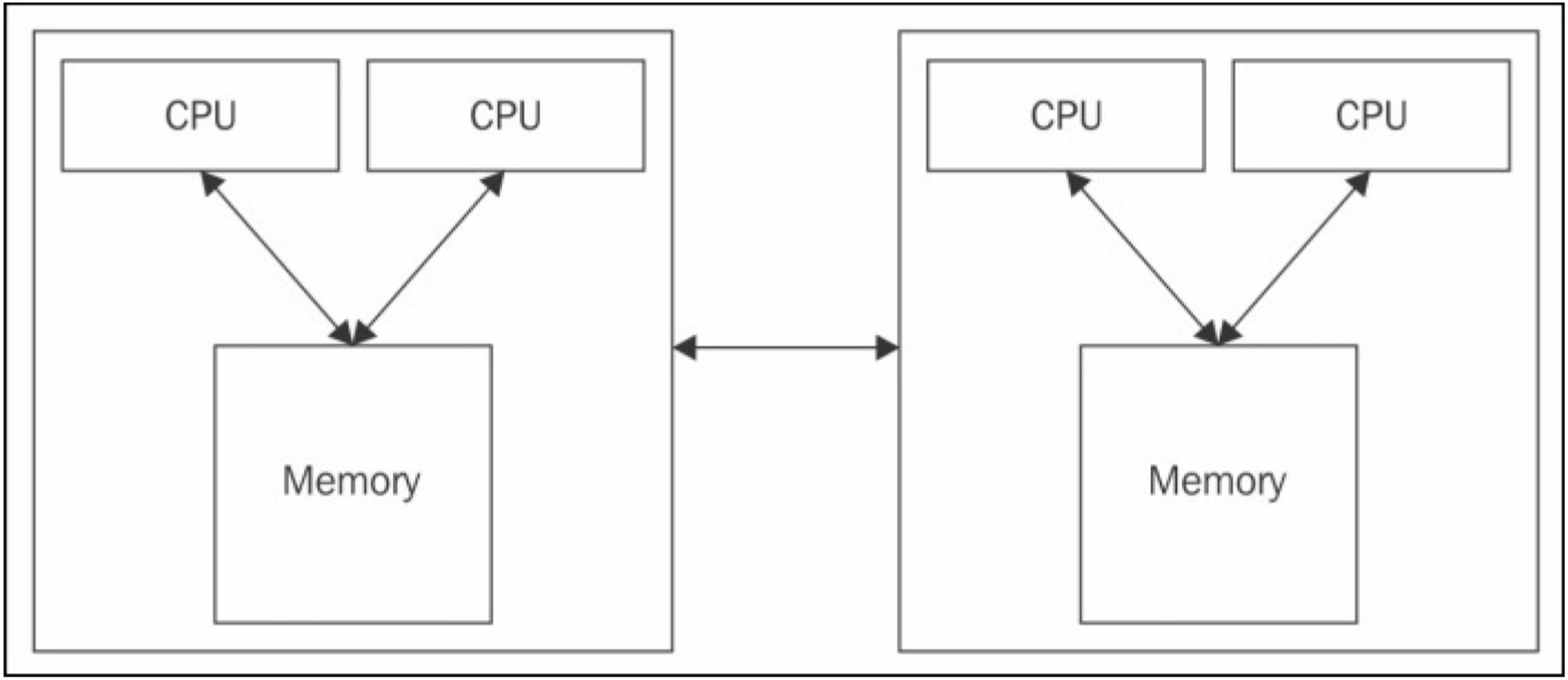

CPU
CPU
CPU
CPU
Memory
Memory
Memory
Memory

- Distribuovany model pamate – kazdy vypoctovy uzol ma vlastnu pamat, ku ktorej nema priamy pristup iny vypoctovy uzol
- Typicka komunikacia medzi uzlami pomocou siete

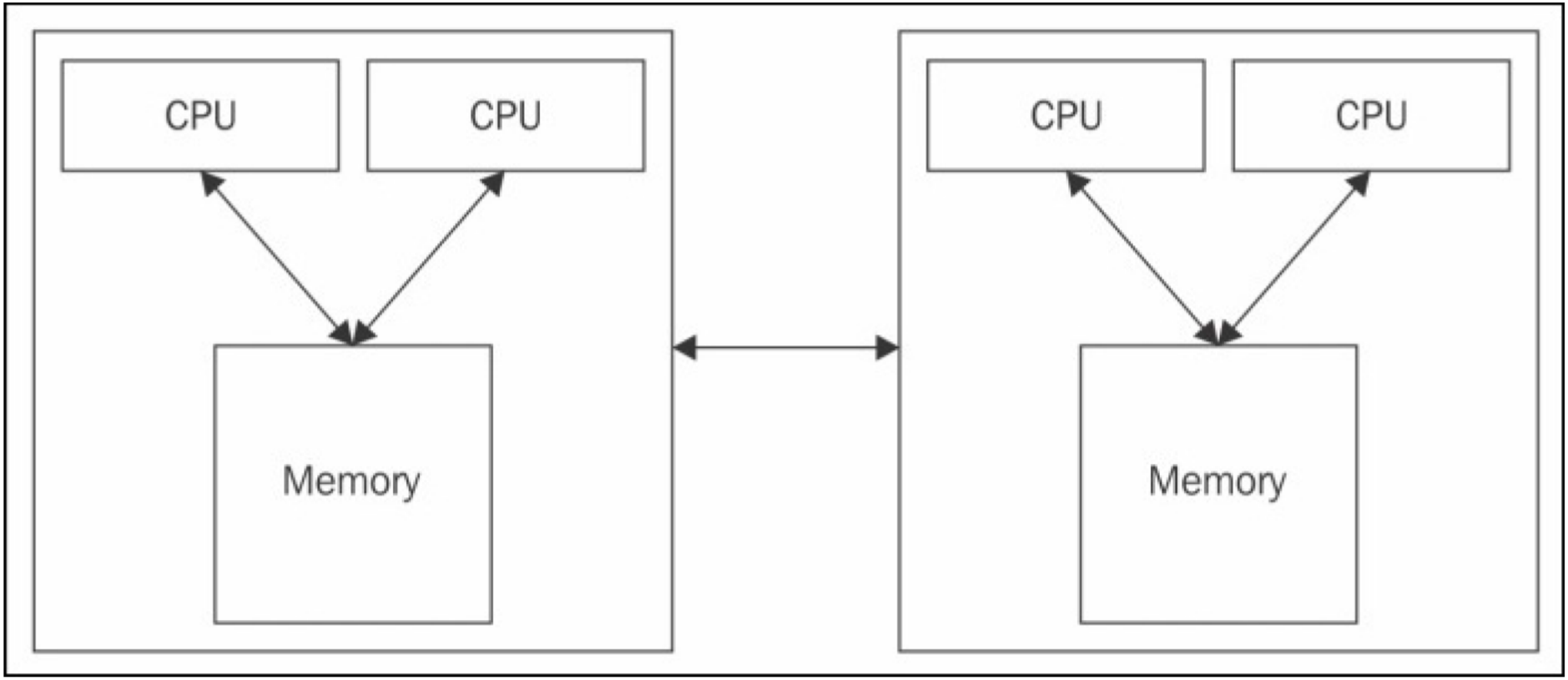



CPU
CPU
CPU
CPU
Memory


CPU
CPU
CPU
CPU
Memory
Memory
Memory
Memory



Hybridna architektura pamate



Hybridna architektura pamate

- V sucasnosti velmi lahko dosiahnutelna architektura
- Pocitace pospajane pomocou siete
- Kazdy vypoctovy uzol ma viacero jadier/procesorov

Zdielana versus distribuovana pamat

- Kazdy model ma svoje vyhody i nevyhody

Pamatove modely

- Zdielana pamat
  + Programator neriesi pristup k datam
  + Zdielania udajov radovo rychlejsie nez pri DP

- Distribuovana pamat
  + Jednoduche rozsirovanie (pridanie PC do siete)
  + Kazdy uzol ma vlastnu pamat, netreba sa starat o subehy pri pristupe k udajom

Pamatove modely

- Zdielana pamat

  + ...

  + ...

  - Sucasny pristup k udajom

  - Drahe rozsirovanie pamate, limity (radovo TB)

- Distribuovana pamat

  + …

  + …

  - Zlozite mapovanie algoritmov na tuto architekturu

  - Programator sa musi sam starat o prenos dat medzi vypoctovymi uzlami

Pamatove modely

- Zdielana pamat
  + Programator neriesi pristup k datam
  + Zdielania udajov radovo rychlejsie nez pri DP
  - Sucasny pristup k udajom
  - Drahe rozsirovanie pamate, limity (radovo TB)
- Distribuovana pamat
  + Jednoduche rozsirovanie (pridanie PC do siete)
  + Kazdy uzol ma vlastnu pamat, netreba sa starat o subehy pri pristupe k udajom
  - Zlozite mapovanie algoritmov na tuto architekturu
  - Programator sa musi sam starat o prenos dat medzi vypoctovymi uzlami

Amdahlov zakon

Amdahlov zakon

- Vyjadruje mieru mozneho zrychlenia vypoctu pri pouziti paralelizmu

Amdahlov zakon

- Vyjadruje mieru mozneho zrychlenia vypoctu pri pouziti paralelizmu
- Zahrna v sebe dve casti

Amdahlov zakon

- Vyjadruje mieru mozneho zrychlenia vypoctu pri pouziti paralelizmu
- Zahrna v sebe dve casti
  - Casti, ktore sa vykonavaju seriovo (a nedaju sa paralelizovat)
  - Casti, ktore sa vykonavaju paralelne

Amdahlov zakon

- Vyjadruje mieru mozneho zrychlenia vypoctu pri pouziti paralelizmu
- Zahrna v sebe dve casti
  - Casti, ktore sa vykonavaju seriovo (a nedaju sa paralelizovat)
  - Casti, ktore sa vykonavaju paralelne
- Vysledny program nemoze byt rychlejsi nez suma seriovo vykonavanych casti na jednom jadre procesora

Amdahlov zakon

- Majme algoritmus, ktoreho paralelne vykonatelnu cast oznacime P a seriovo vykonatelnu cast oznacime S. Plati, ze S+P = 100%.

Amdahlov zakon

- Majme algoritmus, ktoreho paralelne vykonatelnu cast oznacime P a seriovo vykonatelnu cast oznacime S. Plati, ze S+P = 100%.
- Nech T(n) oznacuje cas (v sekundach), ktory je potrebny na beh algoritmu pri pouziti n vypoctovych vlakien (procesov).

Amdahlov zakon

- Majme algoritmus, ktoreho paralelne vykonatelnu cast oznacime P a seriovo vykonatelnu cast oznacime S. Plati, ze S+P = 100%.
- Nech T(n) oznacuje cas (v sekundach), ktory je potrebny na beh algoritmu pri pouziti n vypoctovych vlakien (procesov).
- Potom plati nasledovny vztah

$$T(n) \geq S * T(1) + \frac{P * T(1)}{n}$$

Amdahlov zakon

$$T(n) \geq S * T(1) + \frac{P * T(1)}{n}$$

Doba vykonania algoritmu na n vypoctovych uzloch (jadrach) je rovna (a zvacsa vacsia) ako doba vykonania seriovej casti na jednom jadre plus doba vykonania paralelne vykonatelnej casti na jednom procesore deleno n (pocet jadier).

Amdahlov zakon

$$T(n) \geq S * T(1) + \frac{P * T(1)}{n}$$

S rastucim n (poctom jadier zapojenych do vypoctu) sa zmensuje druhy clen suctu. S n iducim do nekonecna ide tento vyraz k nule, takze

$$T(\infty) \approx S * T(1)$$

Doba vykonania algoritmu na znacne velkom pocte jadier je priblizne rovna dobe vykonania jeho seriovej casti na jednom jadre.

Dosledky Amdahlovho zakona

- Pozorovanie: casto neviem plne paralelizovat algoritmus, ostava seriovo vykonatelna cast
  - Priprava dat (kopirovanie medzi uzlami)
  - Rozdelenie udajov a ich prenos sietou
  - Zber udajov, postprocessing
  - …
- Casto sa stava, ze doba behu algoritmu je dana jeho seriovo vykonatelnou castou

Dosledky Amdahlovho zakona

- Dokonca sa velmi casto stava, ze zavedenim paralelneho vypoctu sa celkova doba vypoctu algoritmu zhorsi! (napr. vzhladom na narast nutnej komunikacie medzi uzlami pri vymene udajov)
- Ak je mozne seriovu cast kodu limitne znizit k nule, paralelna cast nam dava moznost velkej skalovatelnosti algoritmu: linearny narast zrychlenia vzhladom na pocet procesorov! (toto je, zial, velmi zriedkavy pripad ☹)

Amdahlov zakon - priklad

- Majme algoritmus, ktoreho doba behu na jednom jadre je 100 sekund
- Dajme tomu, ze dokazeme paralelizovat 99% algoritmu

$$T(1) = 100s$$

$$T(10) \approx 0.01 * 100s + \frac{0.99 * 100s}{10} = 10.9s \Rightarrow 9.2X \ speedup$$

$$T(100) \approx 1s + 0.99s = 1.99s \Rightarrow 50.2X \ speedup$$

$$T(1000) \approx 1s + 0.099s = 1.099s \Rightarrow 91X \ speedup$$

$$T(1) = 100s$$

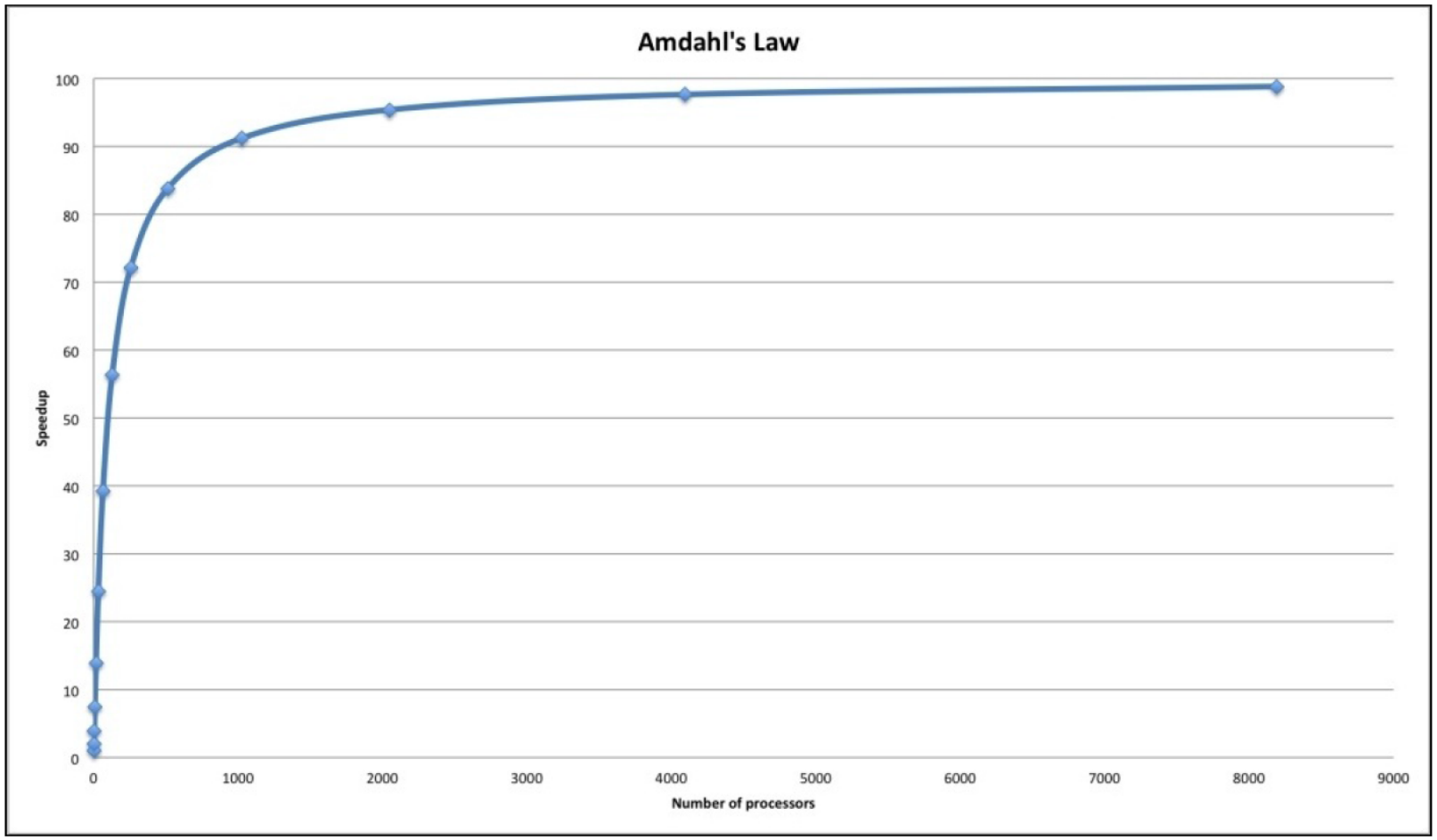
$$T(10) \approx 0.01 * 100s + \frac{0.99 * 100s}{10} = 10.9s \Rightarrow 9.2X \ speedup$$

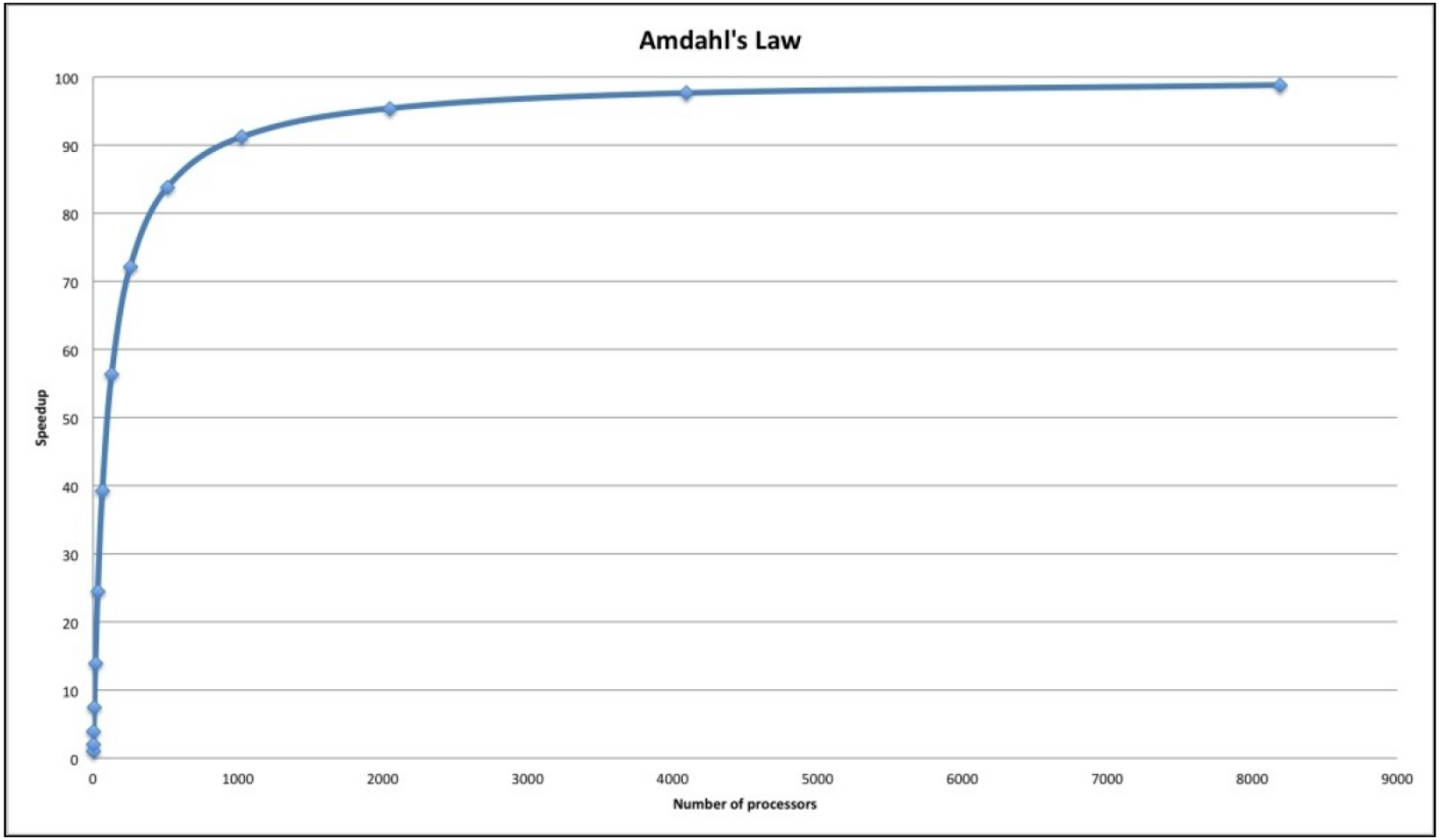
$$T(100) \approx 1s + 0.99s = 1.99s \Rightarrow 50.2X \ speedup$$

$$T(1000) \approx 1s + 0.099s = 1.099s \Rightarrow 91X \ speedup$$

- Vysledky prikladu su dost znechucujuce…
  - Pri 10 jadrach zrychlenie 9.2-krat
  - Pri 100 jadrach zrychlenie 50-krat
  - Pri 1000 jadrach zrychlenie iba 91-krat!
- 100x zvysime pocet jadier, a iba 10x sa nam zrychli vypocet!!!

Amdahlov zakon - priklad





- Bez ohladu na to, kolko jadier pouzijeme, neziskame lepsie nez 100 nasobne zrychlenie
- Trvanie vypoctu bude vzdy aspon 1 sekundu! (doba behu seriovej casti)

Amdahlov zakon

- Dva dolezite poznatky z tohto zakona:
  1. Kolko nasobne zrychlenie vieme ocakavat v tom najlepsom pripade

Amdahlov zakon

- Dva dolezite poznatky z tohto zakona:
  1. Kolko nasobne zrychlenie vieme ocakavat v tom najlepsom pripade
  2. Kedy sa uz neoplati pridavat dalsie vypoctove uzly do vypoctu (t.j. investovat do hardveru), pretoze prinos zrychlenia sa vzhladom na investiciu neoplati

Amdahlov zakon

- Zakon plati rovnako pre PP aj pre DP!

Amdahlov zakon

- Zakon plati rovnako pre PP aj pre DP!
  - Pri PP zvacsa netreba zohladnovat dobu pristupu do pamate, ale synchronizaciu ano!

Amdahlov zakon

- Zakon plati rovnako pre PP aj pre DP!
  - Pri PP zvacsa netreba zohladnovat dobu pristupu do pamate, ale synchronizaciu ano!
  - Pri DP sa zvacsa markantne zvysuje doba seriovo vykonatelnej casti (kvoli dobe pristupu k udajom)

Druhe dejstvo

paralelny / konkurentny

- Konkurentne udalosti: mozu byt realizovane sucasne (ale NEMUSIA)
- Hovorime, ze udalosti v programe sa vykonavaju konkurentne, ak vzhladom na zdrojovy kod nevieme urcit, v akom poradi nastavaju

- Paralelne udalosti: su realizovane sucasne

Synchronizacia

- Nielen sucasne 2 veci, ale lubovolny pocet udalosti v roznych (casovych) vazbach (pred, sucasne, po)
- Synchronizacne obmedzenia mozu byt rozne
  - Serializacia (A < B)
  - Vzajomne vylucenie (A <>B)

Kto ranajkoval skor?

- Mame moznost pre lubovolnych 2 ludi spätne overit, kto zacal ranajkovat skor?

Kto ranajkoval skor?

- Mame moznost pre lubovolnych 2 ludi spätne overit, kto zacal ranajkovat skor?

- Mame moznost zabezpecit (na zajtra), aby jeden z dvojice zacal ranajkovat skor nez ten druhy?

Serializacia spravami

Jano

1. Spanie
2. Ranna hygiena
3. Ranajkovanie
4. Zavolanie

Fero

1. Spanie
2. Prijatie hovoru
3. Ranajkovanie

Formalny zapis

$J1 < J2 < J3 < J4$

$F1 < F2 < F3$

Predpoklad: $J4 < F2$

$J1 < J2 < J3 < J4 < F2 < F3$

$J3 < F3$

Konkurentne programovanie

- Nedeterministicke spravanie
  - V akom poradi nastanu udalosti?
- Zdielane premenne versus integrita
  - zapis
  - citanie + zapis
  - Su operacie nad udajmi atomicke?

Nedeterministicke

Vlakno A:

print('yes')

Vlakno B:

print('no')

Poradie vykonavania

Vlakno A:

X = 5

print(X)

Vlakno B:

X = 7

Poradie vykonavania

A1 < A2 < B1

A1 < B1 < A2

B1 < A1 < A2

??? moze nastat A2 < A1 ???

Neatomicka aktualizacia

Vlakno A:

X += 1

Vlakno B:

X += 1

Vlakno A:

Tmp = X

X = Tmp + 1

Vlakno B:

Tmp = X

X = Tmp + 1

Vylucenie pomocou sprav

- Jano a Fero operatori v Mochovciach
- Vzdy aspon jeden musi pozerat na kontrolky
- Maju moznost atomickej signalizacie (telefon)

- Akym sposobom sa da vyriesit problem naobedovania sa? (bez hodin, bez obmedzenia na cas obeda a jeho dlzku; je jedno, kto bude jest ako prvy)
- Aky najmensi pocet sprav riesi tuto ulohu?

- Jano a Fero sa dohodnu, kto pojde prvy na obed… 1 sprava

- Jano a Fero sa dohodnu, kto pojde prvy na obed… 1 sprava... staci?

- Jano a Fero sa dohodnu, kto pojde prvy na obed… 1 sprava... staci?

- Da sa vyriesit tento problem tak, ze nebudu vopred dohodnuti, kto ide prvy na obed?

1. Semafor

- Abstraktny udajovy typ (ADT)

- Interny stav: cislo
- Metody na modifikaciu stavu:
  - Inicializacia
  - Inkrementacia
  - Dekrementacia

1. Semafor

- Inicializacia
  - Lubovolne cislo (z podporovaneho rozsahu)
  - *Nie je mozne citat hodnotu semaforu*
  - Po inicializacii pouzivat iba inc() a dec()
- Dekrementacia
  - Ak je po dekrementacii vysledok zaporny, volajuci musi cakat, kym niekto nezavola nad semaforom inkementaciu

1. Semafor

- Inkrementacia
  - Ak v case inkrementacie jestvuje niekto, kto caka na uvolnenie (v metode dekrementacie), bude mu umoznene pokracovat
  - Zvysi sa hodnota semaforu o 1
  - !!! *Lubovolny z cakatelov moze pokracovat* !!!
  - Ak mame tzv. silny semafor, ten implementuje FIFO frontu cakajucich, takze cakanie bude realizovane pomocou nej

1. Semafor - obmedzenia

- Kym nezavolame dec(), nevieme, ci ostaneme zablokovani a ci nie
- V inc() po uvolneni cakajuceho nevieme, kto bude pokracovat a v akom poradi; aj uvolneny v dec(), aj ten, kto vyvolal inc() moze pokracovat vo svojej cinnosti dalej (konkurentne)
- Pri volani inc() nevieme, ci niekto caka; pocet uvolnenych moze byt 0 alebo 1

1. Semafor – stav hodnoty

- > 0
  - Pocet vlakien, ktore mozu zavolat dec() bez toho, aby sa zablokovali
- < 0
  - Pocet vlakien, ktore su zablokovane v dec() a cakaju na uvolnenie pomocou inc()
- = 0
  - Ziadne vlakno necaka; az najblizsie volanie dec() bude blokujuce

1. Semafor - syntax

- inc() / dec()
- signal() / wait()
- V() / P()
- acquire() / release()
- increment_and_wake_a_waiting_process_if_any() / decrement_and_block_if_the_result_is_negative()
- ...

2. Zamok (Mutex)

- Abstraktny udajovy typ (ADT)

- Interny stav: 0 alebo 1
- Metody na modifikaciu stavu:
  - Inicializacia
  - Uzamknutie
  - Odomknutie

2. Zamok (Mutex)

- Inicializacia
  - Na stav „odomknuty“ (zvacsa hodnota 0)
  - *Nie je mozne citat hodnotu zamku*
  - Po inicializacii pouzivat iba lock() a unlock()
- Uzamknutie (metoda lock())
  - Ak je v case vyvolania metody hodnota semaforu „uzamknuty“ (zvacsa 1), volajuci musi cakat na odomknutie

2. Zamok (Mutex)

- Odomknutie (metoda unlock())
  - Hodnota zamku sa nastavi na „odomknuty“
  - !!! *Lubovolny z cakatelov moze pokracovat* !!!
  - Ale iba jeden!

2. Zamok (Mutex)

- Zamok je zjednodusenou verziou Semaforu
- Nie je vsak uplne totozny s binarnym Semaforom

2. Zamok (Mutex)

- Rozdiel oproti Semaforu:
  - Vacsina implementacii
  - Kto zamkne, musi odomknut!

2. Zamok (Mutex) – implement.

- Spinlock
  - Neustale vytazuje CPU
  - Nemoze prist ku preplanovaniu procesu!
- Sleeplock
  - Nevytazuje CPU, vyzaduje sa vsak kooperacia s planovacom OS!
  - Vyzaduje viac zdrojov (fronta)

2. Zamok (Mutex) - Spinlock

```
def lock(m):
    while swap(m, 1) == 1:
        pass

def unlock(m):
    swap(m, 0)
```

2. Zamok (Mutex) - Sleeplock

```
def lock(m):
    if swap(m.status, 1) == 1:
        add_m_to_queue(m, m.queue)
        yield()
def unlock(m):
    if not is_empty(m.queue):
        resume(pop(m.queue))
    else:
        swap(m.status, 0)
```

2. Zamok (Mutex)

- Implementacia vyzaduje atomicku operaciu pre otestovanie a nastavenie pamatoveho miesta

- swap(where, what)
- test_and_set(where)
  - Nastavi pamatove mieto na 1
  - Vrati povodnu hodnotu, ktora bola pred zmenou

3. Zakladne synchronizacne vzory

- Signalizacia
- Vzajomna signalizacia (rendezvous)
- Mutex (mutual exclusion)
- Multiplex
- Bariera
- Znovu pouzitelna bariera (v cykle)
- Fronta

3. Zakladne synchronizacne objekty

- Semafor / Zamok
- Turniket (turnstile)
- Vypinac (lightswitch)
- Skore (score board)

Tretie dejstvo

Python

verzia 3.x

Python

- Verzia 3.x
- Prostredie Idle

- print
- komentare
- Operatory +,-,*,**,/,//,%; and, or, not
- premenne

Python - premenne

- Objekt je instanciou triedy
  - Identita - id()
  - Typ - type()
  - Hodnota
- Premenna
  - Priradenie
  - Porovnanie

Python – typy objektov

- Numericke
  - NoneType: None
  - bool: True, False
  - int, float
- Sekvencne
  - str
  - list, tuple, range
- Kolekcie
  - dict
  - set

Python – pretypovanie

- novy_typ(hodnota)

- int('32')
- int('ahoj')
- int(3.00023)
- str(32)
- str(True)

Python – nejake funkcie

- Nacitanie vstupu: input()
  - cislo_a = input(“zadaj cislo:”)
  - type(cislo_a)
  - cislo_a = int(cislo_a)
  - cislo_b = int(input(“zadaj druhe cislo:”))
- Dlzka: len(“ahoj svjete!”)

Python – funkcie

```
def meno_funkcie(parametre):
    prikazy

def novy_riadok():
    print()

def tri_riadky():
    novy_riadok()
    novy_riadok()
    novy_riadok()
```

Python – argumenty

- Pozicne
- Klucove

```
def mocnina(m, n=2):
        print(m**n)

mocnina(4)      # 4 ** 2 → 16
mocnina(4,3)    # 4 ** 3 → 64
```

Python – import a return

```
from time import sleep
import random

def sucet(a, b):
    sleep(random.randint(1,10)/10)
    return a+b
```

Python – PEP8

- Python – vynútené odsadzovanie
- Čitateľnosť – PEP8
  - Odsadenie (indent): 4 medzery!!!
  - Import na začiatku skriptu
  - Definície funkcií oddelené 2 prázdnymi riadkami
  - Najprv definície funkcií, na konci skriptu ich volania (tj. vykonávaný kód na konci skriptu)

- https://www.python.org/dev/peps/pep-0008

Cvicenie

- https://uim.fei.stuba.sk/i-ppds/priprava-prostredia

- https://uim.fei.stuba.sk/i-ppds/1-cvicenie-oboznamenie-sa-s-prostredim-%f0%9f%90%8d

Cvicenie

- Implementujte dve vlákna, ktoré budú používať spoločný index do spoločného poľa (inicializovaného na hodnoty 0) istej veľkosti.
- Každé vlákno nech inkrementuje ten prvok poľa, kam práve ukazuje spoločný index. Následne nech index zvýši.
- Ak už index ukazuje mimo poľa, vlákno svoju činnosť skončí.
- Po skončení vlákien spočítajte, koľko prvkov poľa má hodnotu 1.

Cvicenie

- Ak zistíte, že nie každý prvok poľa má hodnotu 1, modifikujte program tak, aby na koniec (po skončení behu vlákien) zistil početnosti (histogram) hodnôt, ktoré sa nachádzajú v poli.

- Potom program „opravte“.