PPaDS MMXXII

Matúš Jókay, C-503, matus.jokay@stuba.sk

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

konzultácie elektronicky

mail, discord

Vedomosti

- Načo je dobrá synchronizácia
- Základné synchronizačné nástroje
- Rôzne synchronizačné problémy

- Asynchrónne, konkurentné, paralelné programovanie

Literatúra

- 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 zápočet, test AIS
  - 30. III. v čase o 9.00, 24 bodov, minimum ?b
  - ??. V. v čase o 9.00, 16 bodov, minimum ?b
- 10x úlohy git repo, každé cviko za 3b
- Skúška 30 bodov, test AIS alebo prezenčne, minimum 10b

Hodnotenie

- Git repozitár: hodnotenie na týždennej báze, pri zistení plagiátorstva NZ a basta fidli ☹

- Repozitár musí obsahovať
  - Kompletnú históriu zmien
  - Označenie autora, potrebné komentáre
  - Commity vo formáte Conventional Commits 1.0.0
  - Akceptuje sa vypracovanie do času začiatku prednášky ďalšieho týždňa

Evaluácia

- Vyžadujem evaluáciu z dôvodu zvyšovania kvality vedenia predmetu
- **Nutná, nie dostatočná podmienka absolvovania predmetu**
- Odporúčam vypĺňať až po skúške ;)

Organizácia

- Prednáška 100 min týždenne (utorok 8:00)

- Dištančná výučba:
  - Spoločný seminár 50 min týždenne (streda 8:00)
  - Spoločné cvičenie 50 min týždenne (po cviku)
- Prezenčná výučba:
  - Cvičenie 100 min týždenne (utorok 13 hod)
  - Cvičenie 100 min týždenne (utorok 15 hod)
  - Cvičenie 100 min týždenne (streda 8:00)

Motivácia

1) Stolčeky

2) Povala

3) Trávnik

Úvod do PPaDS

Úvod do PPaDS

Francesco Pierfederici

Distributed Computing with Python

Packt Publishing Ltd.

April 2016

ISBN 978-1-78588-969-1

Úvod do PPaDS

- Prvý počítač – 40-te roky konca minulého tisícročia
- Odvtedy 80 rokov… terajšie smartfóny sú rýchlejšie než najrýchlejší počítač spred 20 rokov ;)
- Netreba nám obrovské miestnosti s klimatizáciou, počítač strčíme do vrecka…

Úvod do PPaDS

- Procesor dokáže v jednom momente spracovávať iba jednu úlohu
- Ilúzia paralelizmu – v krátkom čase striedanie úloh
- Na skutočné súčasné vykonávanie viacerých úloh v jednom čase potrebujeme viac procesorov/jadier
- V súčasnosti to už nie je problém…

Úvod do PPaDS

- Multiprocesorové a multijadrové systémy (osobné počítače, notebooky, netbooky, laptopy, tablety, smartfóny, minipočítače, …)
- Grafické karty (stovky až tisíce výpočtových uzlov), tzv. GPU (Graphics Processing Unit)
- Počítačové siete (Internet, siete mobilných operátorov, lokálne siete (lan, wifi), …)

PPaDS – Definície

- Paralelný výpočet

- Distribuovaný výpočet

PPaDS – Definície

- Paralelný výpočet

  Paralelný výpočet je súčasné využitie viacerých (t.j. viac než 1) výpočtových uzlov na dosiahnutie cieľa výpočtu

- Distribuovaný výpočet

  Distribuovaný výpočet je súčasné využitie viacerých (t.j. viac než 1) výpočtových uzlov na dosiahnutie cieľa výpočtu

PPaDS – Definície

- Paralelný výpočet

  Paralelný výpočet je súčasné využitie viacerých (t.j. viac než 1) výpočtových uzlov na dosiahnutie cieľa výpočtu

- Distribuovaný výpočet

  Distribuovaný výpočet je súčasné využitie viacerých (t.j. viac než 1) výpočtových uzlov na dosiahnutie cieľa výpočtu

PPaDS – Definície

- **Paralelný** výpočet (výpočtový uzol = **CPU**)

  Paralelný výpočet je súčasné využitie viacerých (t.j. viac než 1) výpočtových uzlov na dosiahnutie cieľa výpočtu

- **Distribuovaný** výpočet (výpočtový uzol = **PC**)

  Distribuovaný výpočet je súčasné využitie viacerých (t.j. viac než 1) výpočtových uzlov na dosiahnutie cieľa výpočtu

PPaDS – Definície

- **Paralelný** výpočet
  - Na procesoroch (jadrách procesora)
  - **Multivláknové** programy na komunikáciu typicky **tá istá (spoločná) pamäť**
  - **Multiprocesové** programovanie na komunikáciu typicky **zdieľaná pamäť**

- **Distribuovaný** výpočet
  - Na počítačoch (uzloch siete) (počítačové farmy) na komunikáciu typicky sieť
  - Na grafických kartách (CUDA, OpenCL) na komunikáciu zbernica počítača (napr. PCIe)

Vývoj distribuovanej aplikácie

1. Vývoj jednovláknovej (jednoprocesovej) aplikácie

Vývoj distribuovanej aplikácie

1. Vývoj jednovláknovej (jednoprocesovej) aplikácie
2. Vývoj multiprocesovej (nie multivláknovej, hoci v závislosti od implementačného prostredia to môže byť jeden z medzikrokov) aplikácie

Vývoj distribuovanej aplikácie

1. Vývoj jednovláknovej (jednoprocesovej) aplikácie
2. Vývoj multiprocesovej (nie multivláknovej, hoci v závislosti od implementačného prostredia to môže byť jeden z medzikrokov) aplikácie
3. Vývoj distribuovanej aplikácie

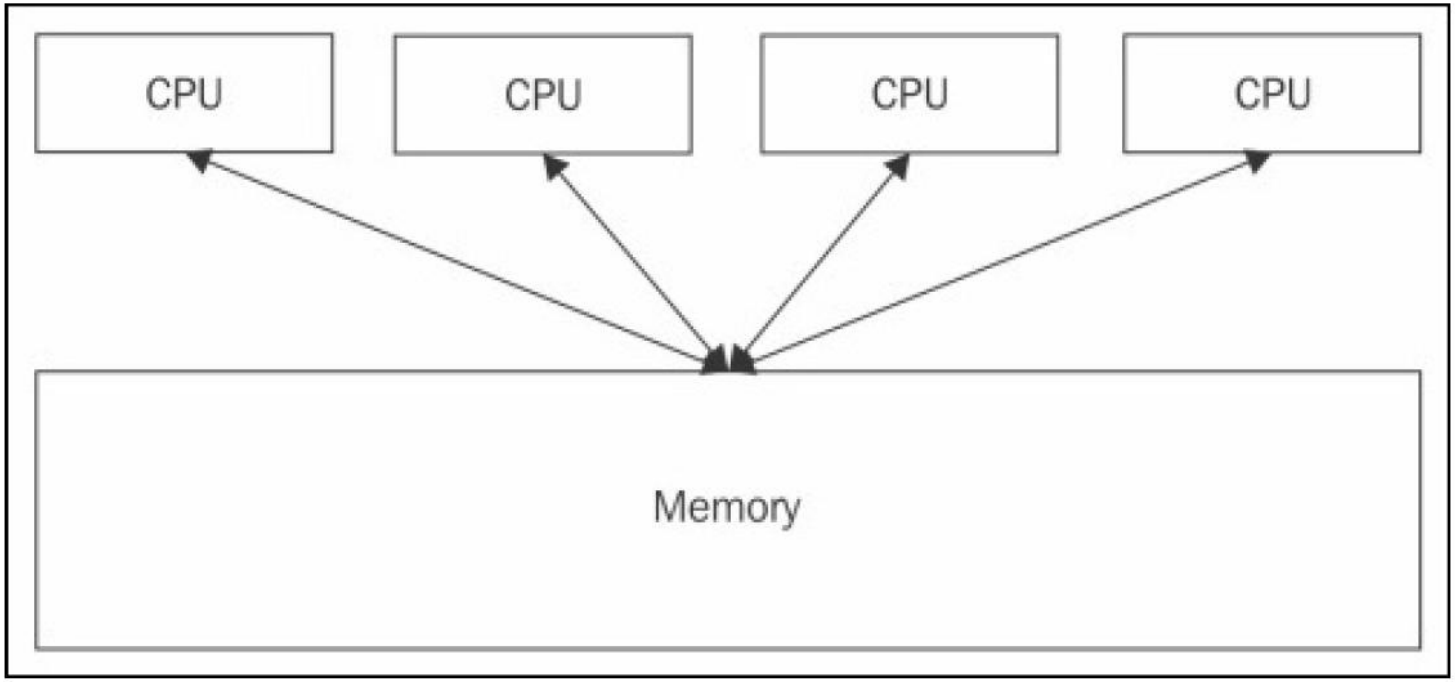
PPaDS – pozor na údaje!

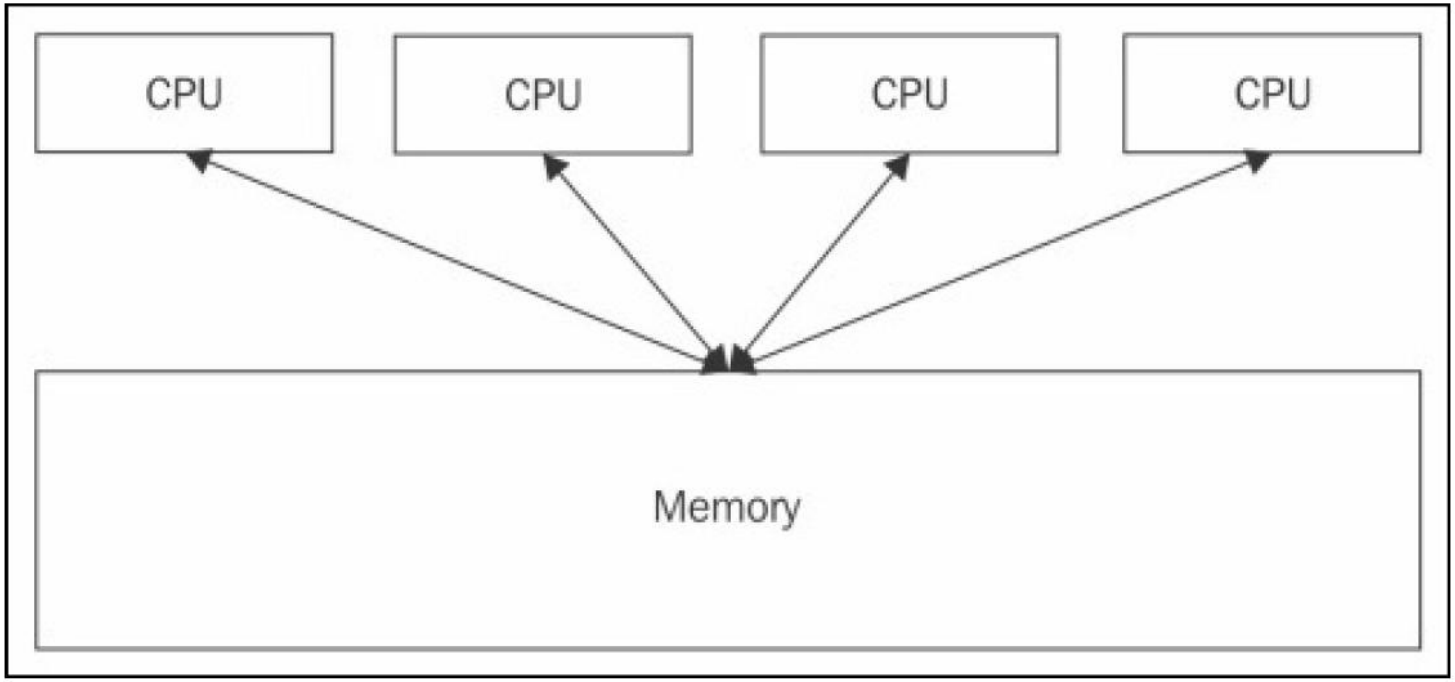
- Skutočným úzkym hrdlom výpočtov zväčša (závisí od typu aplikácie) bývajú samotné údaje, nie CPU

- Niekedy stačí zdieľaný súborový systém (napr. NFS na unixových systémoch), inokedy zdieľaná databáza alebo posielanie správ…

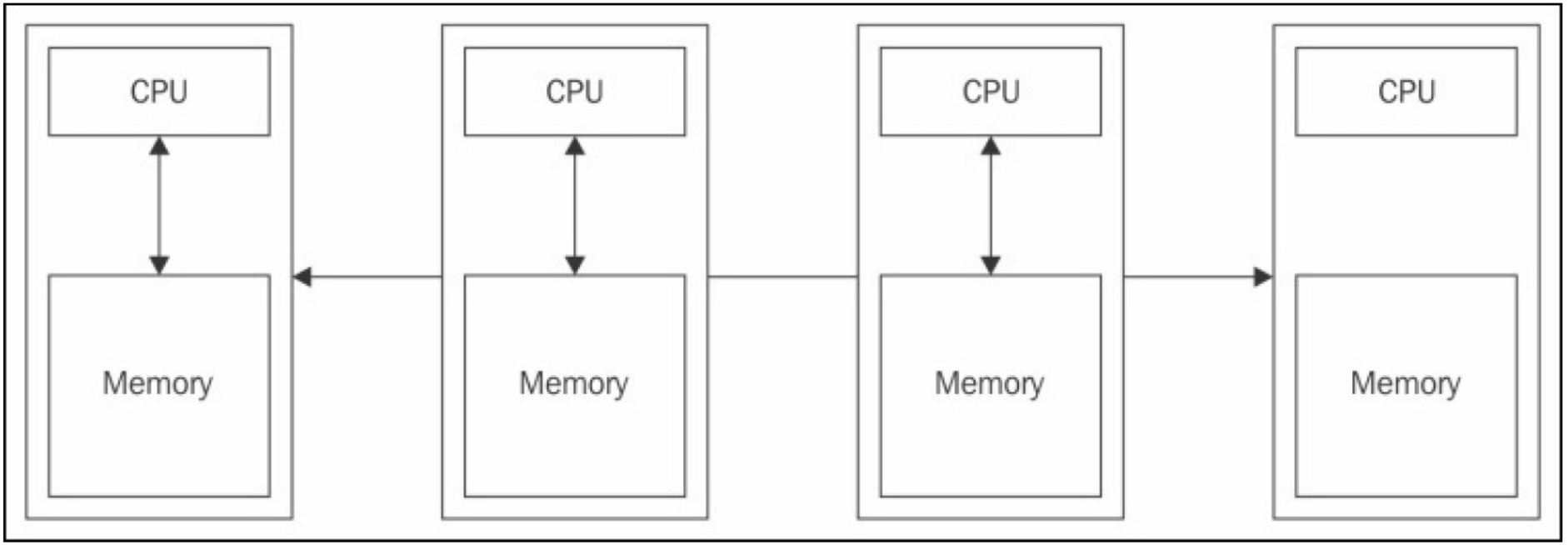
Zdieľaná versus distribuovaná pamäť

- Fyzické umiestnenie výpočtových zdrojov má veľký dopad na efektivitu výpočtu
- Najväčší rozdiel medzi PP a DP je v použitej pamäťovej architektúre a v spôsobe prístupu k dátam
  - PP zväčša využíva ten istý pamäťový priestor
  - DP zväčša využíva distribuovaný model pamäte

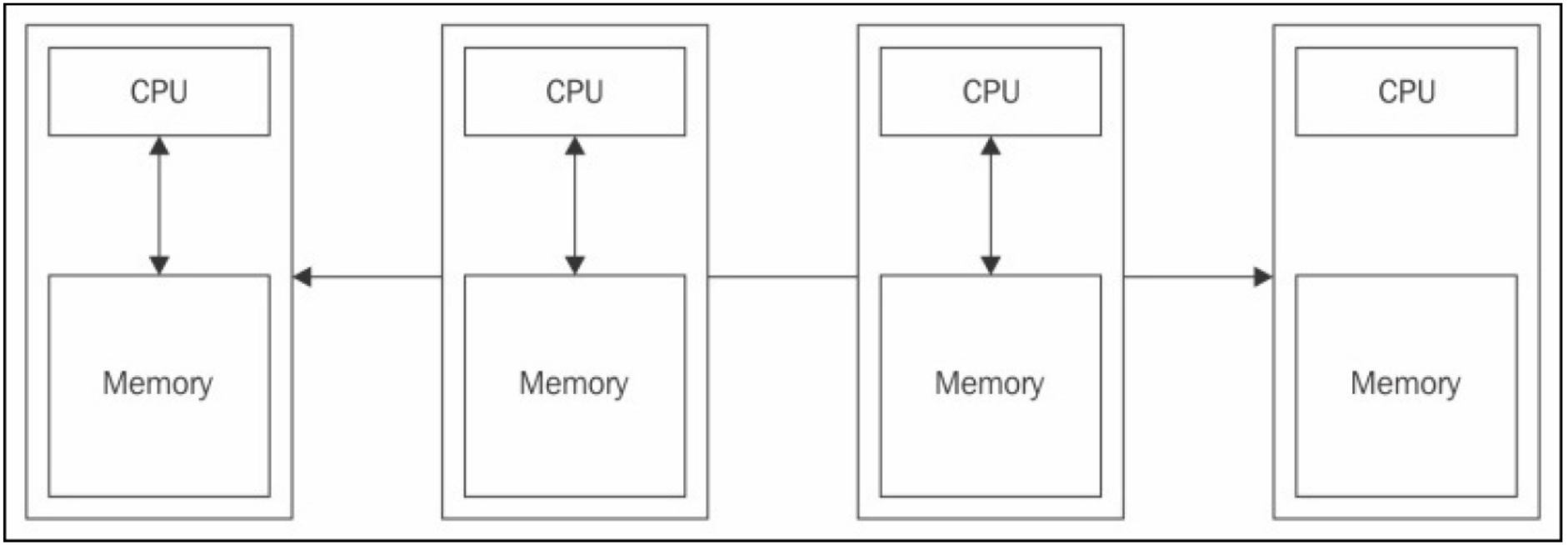

CPU
CPU
CPU
CPU
Memory

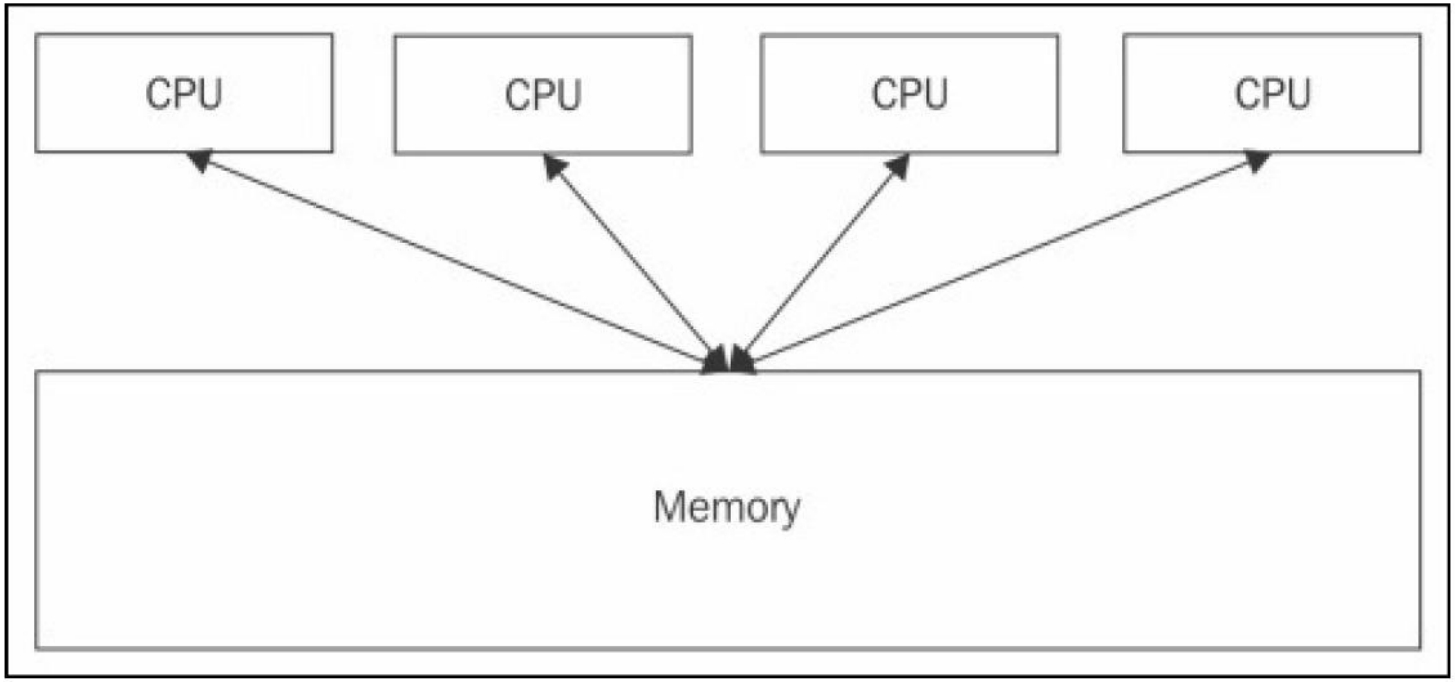


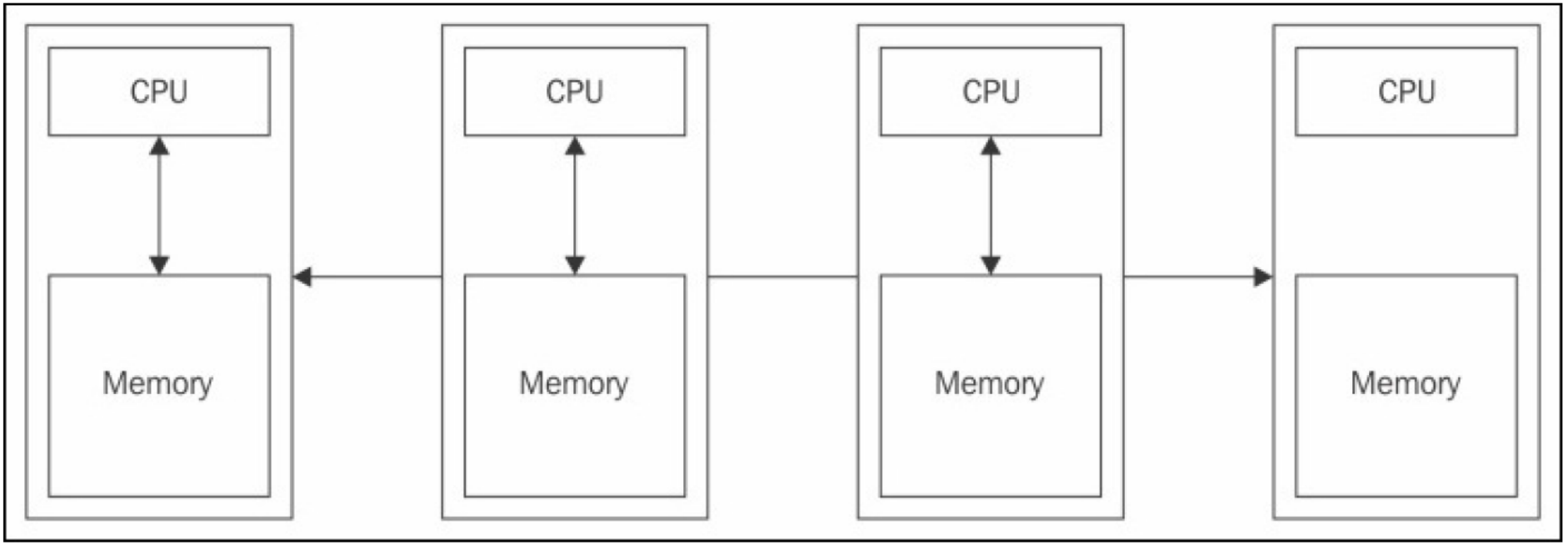
- Multivláknové programovanie natívne využíva túto architektúru (ako, **vieme z OS**?)
- Zdieľanie pamäte dosiahnuteľné aj v multiprocesovom programovaní (**ako**?)
- Problematické, ale nie nemožné aj v distribuovanom programovaní (pomocou tzv. *middleware*)

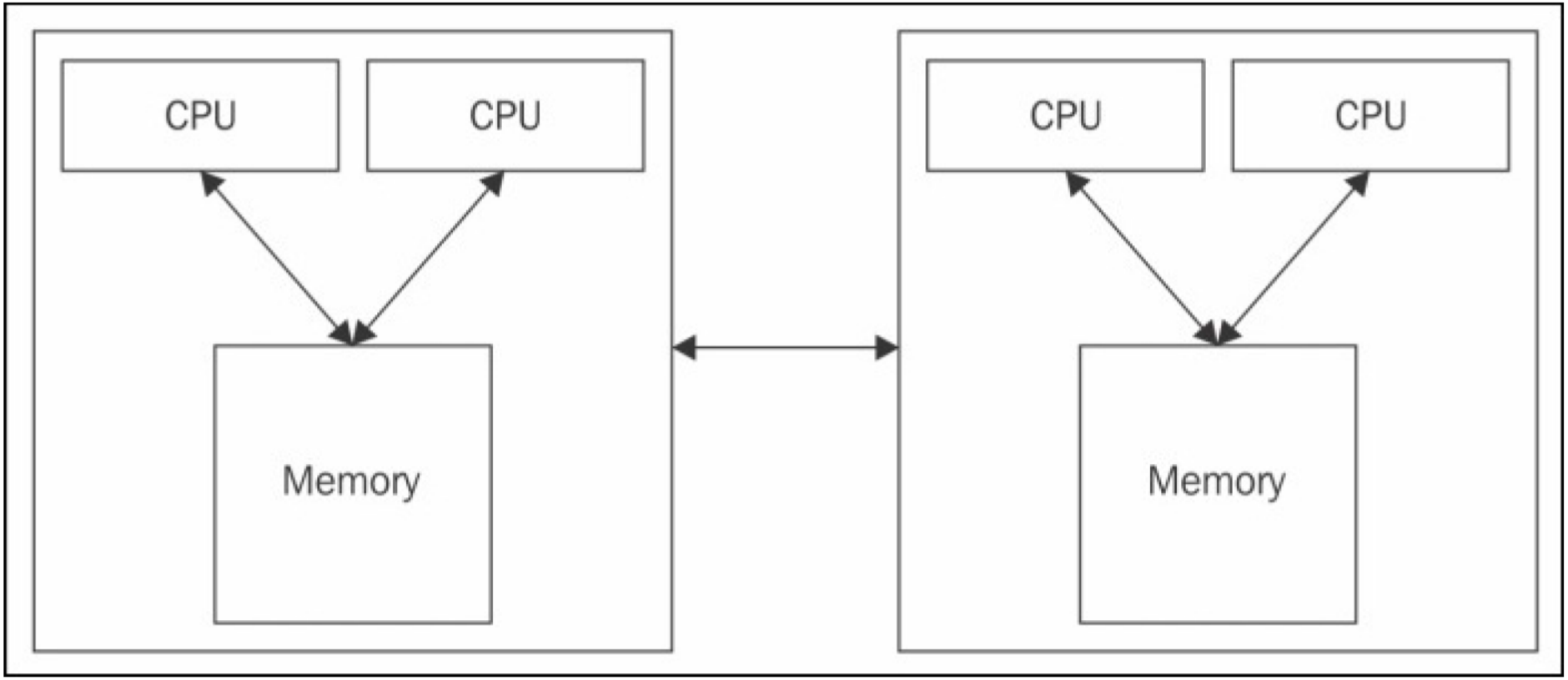

CPU
CPU
CPU
CPU
Memory
Memory
Memory
Memory

- Distribuovaný model pamäte – každý výpočtový uzol má vlastnú pamäť, ku ktorej nemá priamy prístup iný výpočtový uzol
- Typická komunikácia medzi uzlami pomocou siete

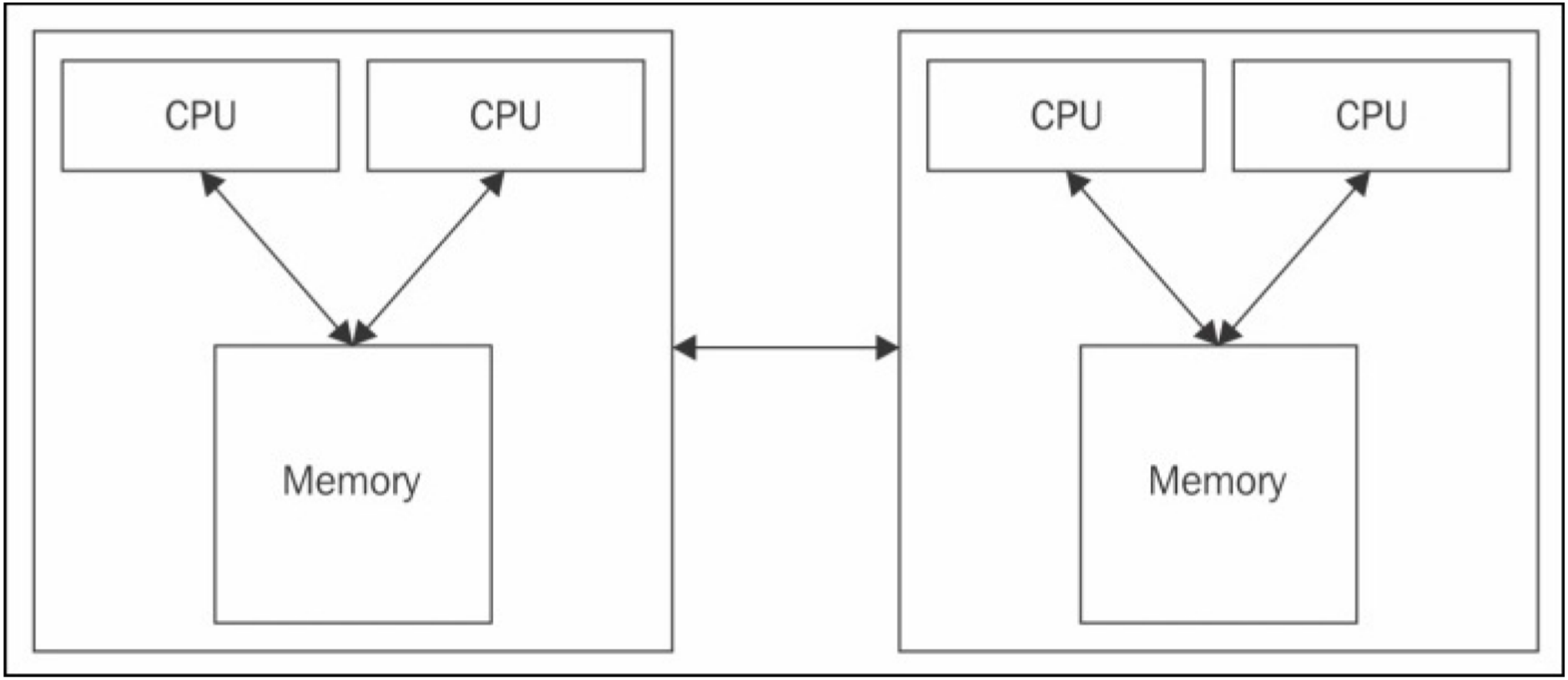



CPU
CPU
CPU
CPU
Memory


CPU
CPU
CPU
CPU
Memory
Memory
Memory
Memory



Hybridná architektúra pamäte



Hybridná architektúra pamäte

- V súčasnosti veľmi ľahko dosiahnuteľná architektúra
- Počítače pospájané pomocou siete
- Každý výpočtový uzol má viacero jadier/procesorov

Zdieľaná versus distribuovaná pamäť

- Každý model má svoje výhody i nevýhody

Pamäťové modely

- Zdieľaná pamäť
  + Programátor nerieši prístup k dátam
  + Zdieľanie údajov rádovo rýchlejšie než pri DP

- Distribuovaná pamäť
  + Jednoduché rozširovanie (pridanie PC do siete)
  + Každý uzol má vlastnú pamäť, netreba sa starať o súbehy pri prístupe k údajom

Pamäťové modely

- Zdieľaná pamäť

  + ...

  + ...

  - Súčasný prístup k údajom (!!!)

  - Drahé rozširovanie pamäte, limity (rádovo TB)

- Distribuovaná pamäť

  + …

  + …

  - Zložité mapovanie algoritmov na túto architektúru

  - Programátor sa musí sám starať o prenos dát medzi výpočtovými uzlami

Pamäťové modely - sumár

- Zdieľaná pamäť
  + Programátor nerieši prístup k dátam
  + Zdieľanie údajov rádovo rýchlejšie než pri DP
  - Súčasný prístup k údajom (!!!)
  - Drahé rozširovanie pamäte, limity (rádovo TB)
- Distribuovaná pamäť
  + Jednoduché rozširovanie (pridanie PC do siete)
  + Každý uzol má vlastnú pamäť, netreba sa starať o súbehy pri prístupe k údajom
  - Zložité mapovanie algoritmov na túto architektúru
  - Programátor sa musí sám starať o prenos dát medzi výpočtovými uzlami

Amdahlov zákon

Amdahlov zákon

- Vyjadruje mieru možného zrýchlenia výpočtu pri použití paralelizmu

Amdahlov zákon

- Vyjadruje mieru možného zrýchlenia výpočtu pri použití paralelizmu
- Zahŕňa v sebe dve časti
  - Časti, ktoré sa vykonávajú sériovo (a nedajú sa paralelizovať)
  - Časti, ktoré sa vykonávajú paralelne

Amdahlov zákon

- Vyjadruje mieru možného zrýchlenia výpočtu pri použití paralelizmu
- Zahŕňa v sebe dve časti
  - Časti, ktoré sa vykonávajú sériovo (a nedajú sa paralelizovať)
  - Časti, ktoré sa vykonávajú paralelne
- Výsledný program nemôže byť rýchlejší než suma sériovo vykonávaných častí na jednom jadre procesora

Amdahlov zákon

- Majme algoritmus, ktorého paralelne vykonateľnú časť označíme P a sériovo vykonateľnú časť označíme S. Platí, že S+P = 100%.

Amdahlov zákon

- Majme algoritmus, ktorého paralelne vykonateľnú časť označíme P a sériovo vykonateľnú časť označíme S. Platí, že S+P = 100%.
- Nech T(n) označuje čas (v sekundách), ktorý je potrebný na beh algoritmu pri použití n výpočtových vlákien (procesov).

Amdahlov zákon

- Majme algoritmus, ktorého paralelne vykonateľnú časť označíme P a sériovo vykonateľnú časť označíme S. Platí, že S+P = 100%.
- Nech T(n) označuje čas (v sekundách), ktorý je potrebný na beh algoritmu pri použití n výpočtových vlákien (procesov).
- Potom platí nasledovný vzťah

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

Amdahlov zákon

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

Doba vykonania algoritmu na n výpočtových uzloch (jadrách) je rovná (a zväčša väčšia) ako doba vykonania sériovej časti na jednom jadre plus doba vykonania paralelne vykonateľnej časti na jednom procesore deleno n (počet jadier).

Amdahlov zákon

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

S rastúcim n (počtom jadier zapojených do výpočtu) sa zmenšuje druhý člen súčtu. S n idúcim do nekonečna ide tento výraz k nule, takže

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

Doba vykonania algoritmu na značne veľkom počte jadier je približne rovná dobe vykonania jeho sériovej časti na jednom jadre.

Dôsledky Amdahlovho zákona

- Pozorovanie: často nevieme plne paralelizovať algoritmus, ostáva sériovo vykonateľná časť
  - Príprava dát (kopírovanie medzi uzlami)
  - Rozdelenie údajov a ich prenos sieťou
  - Zber údajov, postprocessing
  - …
- Často sa stáva, že doba behu algoritmu je daná jeho sériovo vykonateľnou časťou

Dôsledky Amdahlovho zákona

- Dokonca sa veľmi často stáva, že zavedením paralelného výpočtu sa celková doba výpočtu algoritmu zhorší! (napr. vzhľadom na nárast nutnej komunikácie medzi uzlami pri výmene údajov)
- Ak je možné sériovú časť kódu limitne znížit k nule, paralelná časť nám dáva možnosť veľkej škálovateľnosti algoritmu: lineárny nárast zrýchlenia vzhľadom na počet procesorov! (toto je, žiaľ, veľmi zriedkavý prípad ☹)

Amdahlov zákon – príklad

- Majme algoritmus, ktorého doba behu na jednom jadre je 100 sekúnd
- Dajme tomu, že dokážeme paralelizovať 99% algoritmu

$$T(1) = 100s$$

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

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

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

$$T(1) = 100s$$

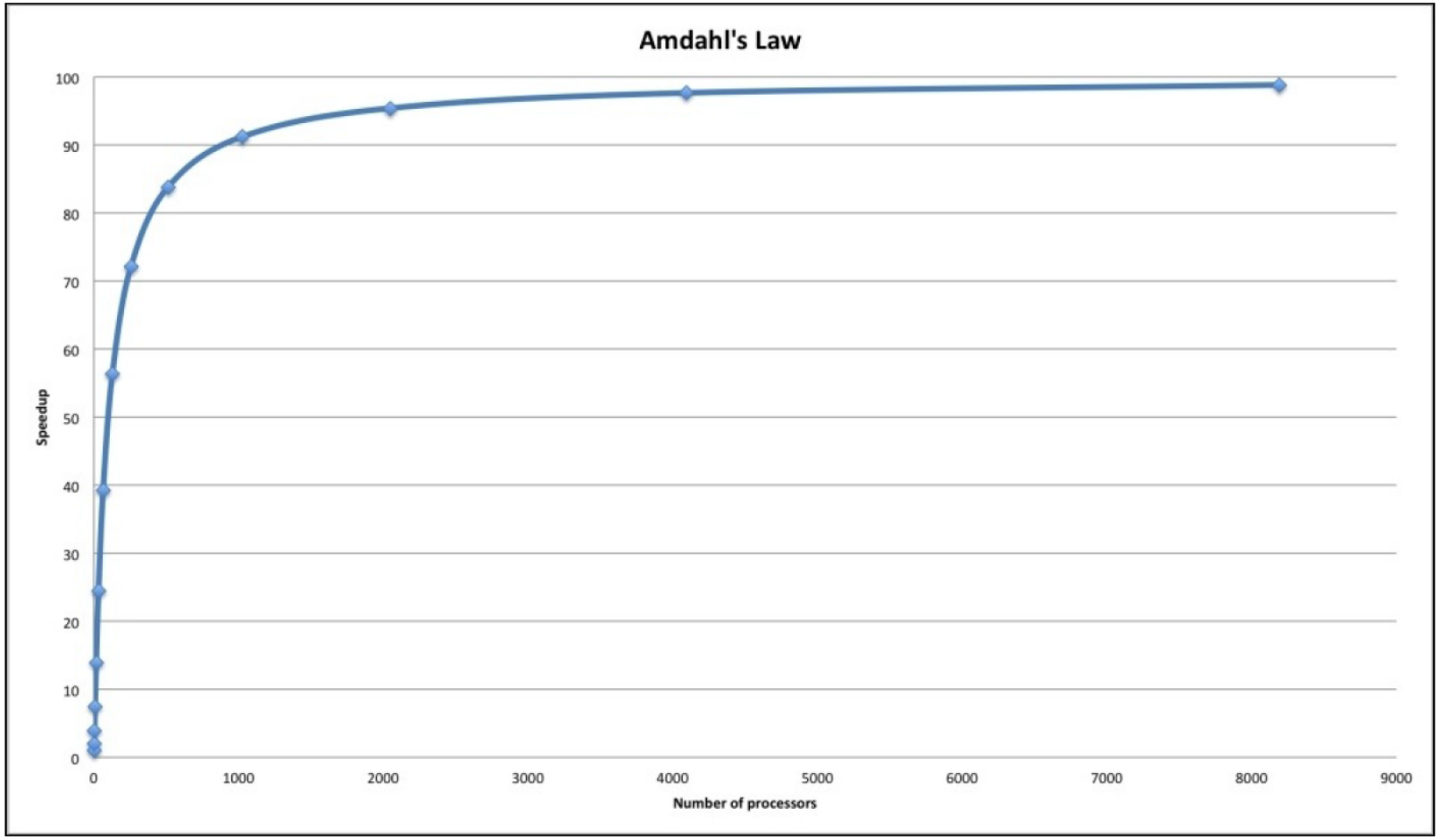
$$T(10) \approx 0.01 * 100s + \frac{0.99 * 100s}{10} = 10.9s \Rightarrow 9.2X \ speedup$$

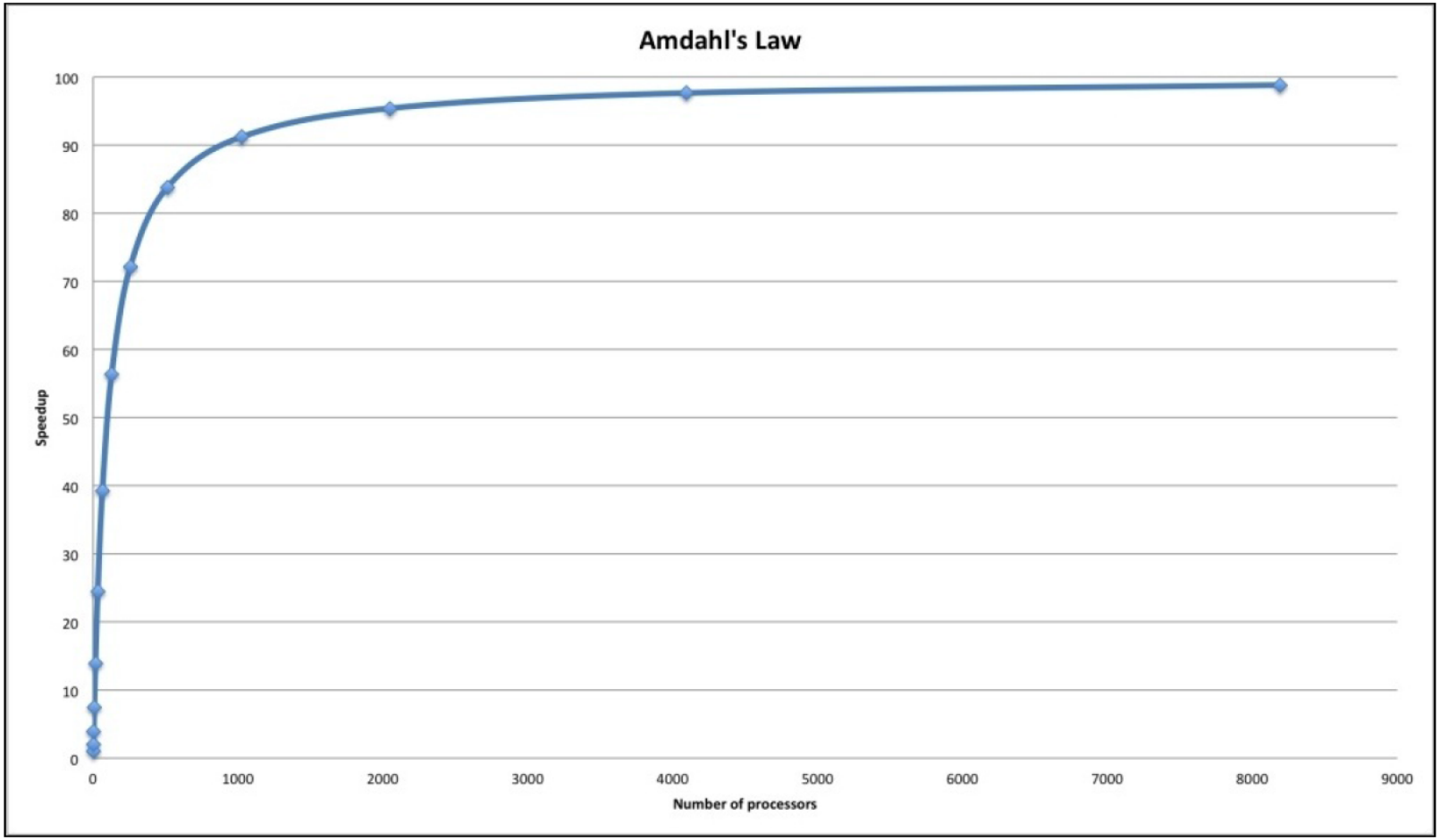
$$T(100) \approx 1s + 0.99s = 1.99s \Rightarrow 50.2X \ speedup$$

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

- Výsledky príkladu sú dosť pesimistické…
  - Pri 10 jadrách zrýchlenie 9.2-krát
  - Pri 100 jadrách zrýchlenie 50-krát
  - Pri 1000 jadrách zrýchlenie iba 91-krát!
- 100x zvýšime počet jadier, a iba 10x sa nám zrýchli výpočet!!! (pri 10 a 1000 jadrách)

Amdahlov zákon – príklad





- Bez ohľadu na to, koľko jadier použijeme, nezískame lepšie než 100-násobné zrýchlenie
- Trvanie výpočtu bude vždy aspoň 1 sekundu! (logické – doba behu sériovej časti)

Amdahlov zákon

- Dva dôležité poznatky z tohto zákona:
  1. Koľko násobné zrýchlenie vieme očakávať v tom najlepšom prípade

Amdahlov zákon

- Dva dôležité poznatky z tohto zákona:
  1. Koľko násobné zrýchlenie vieme očakávať v tom najlepšom prípade
  2. Kedy sa už neoplatí pridávať ďalšie výpočtové uzly do výpočtu (t.j. investovať do hardvéru), pretože prínos zrýchlenia sa vzhľadom na investíciu neoplatí

Amdahlov zákon

- Zákon platí rovnako pre PP aj pre DP!

- Pri PP zväčša netreba zohľadňovať dobu prístupu do pamäte, ale synchronizáciu áno!
- Pri DP sa zväčša markantne zvyšuje doba sériovo vykonateľnej časti (kvôli dobe prístupu k údajom)

Druhé dejstvo

paralelný / konkurentný

- Konkurentné udalosti: môžu byť realizované súčasne (ale NEMUSIA)
- Hovoríme, že udalosti v programe sa vykonávajú konkurentne, ak vzhľadom na zdrojový kód nevieme určiť, v akom poradí nastávajú

- Paralelné udalosti: sú realizované súčasne

Synchronizácia

- Nielen súčasne 2 veci, ale ľubovoľný počet udalostí v rôznych (časových) väzbách (pred, súčasne, po)
- Synchronizačné obmedzenia môžu byť rôzne
  - Serializácia (A < B)
  - Vzájomné vylúčenie (A <>B)

Kto raňajkoval skôr?

- Máme možnosť pre ľubovoľných 2 ľudí spätne overiť, kto začal raňajkovať skôr?

Kto raňajkoval skôr?

- Máme možnosť pre ľubovoľných 2 ľudí spätne overiť, kto začal raňajkovať skôr?

- Máme možnosť zabezpečiť (na zajtra), aby jeden z dvojice začal raňajkovať skôr než ten druhý?

Serializácia správami

Jano

1. Spanie
2. Ranná hygiena
3. Raňajkovanie
4. Zavolanie

Fero

1. Spanie
2. Prijatie hovoru
3. Raňajkovanie

Formálny zápis

$J1 < J2 < J3 < J4$

$F1 < F2 < F3$

Predpoklad: $J4 < F2$

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

$J3 < F3$

Konkurentné programovanie

- Nedeterministické správanie
  - V akom poradí nastanú udalosti?
- Zdieľané premenné versus integrita
  - zápis
  - čítanie + zápis
  - Sú operácie nad údajmi atomické?

1) Nedeterministické

Vlákno A:

print('yes')

Vlákno B:

print('no')

2) Poradie vykonávania

Vlákno A:

X = 5

print(X)

Vlákno B:

X = 7

Poradie vykonávania

A1 < A2 < B1

A1 < B1 < A2

B1 < A1 < A2

??? môže nastať A2 < A1 ???

3) Integrita/konzistentnosť dát

Vlákno A:

X += 1

Vlákno B:

X += 1

Vlákno A:

Tmp = X

X = Tmp + 1

Vlákno B:

Tmp = X

X = Tmp + 1

Vylúčenie pomocou správ

- Jano a Fero operátori v Mochovciach
- Vždy aspoň jeden musí pozerať na kontrolky
- Majú možnosť atomickej signalizácie (SMS)

- Akým spôsobom sa dá vyriešiť problém naobedovania sa? (bez hodín, bez obmedzenia na čas obeda a jeho dĺžku; kto bude jesť prvý, sa musí dohodnúť algoritmom)
- Aký najmenší počet správ rieši túto úlohu?

- Jano a Fero sa dohodnú, kto pôjde prvý na obed… 1 správa

- Jano a Fero sa dohodnú, kto pôjde prvý na obed… 1 správa... stačí?

- Jano a Fero sa dohodnú, kto pôjde prvý na obed… 1 správa... stačí?

- Dá sa vyriešiť tento problém tak, že nebudú vopred dohodnutí, kto ide prvý na obed a rozhodne sa to až ich komunikáciou?

Synchronizačné nástroje

1. Semafor

- Abstraktný údajový typ (ADT)

- Interný stav: číslo
- Metódy na modifikáciu stavu:
  - Inicializácia
  - Inkrementácia
  - Dekrementácia

1. Semafor

- Inicializácia
  - Ľubovoľné číslo (z podporovaného rozsahu)
  - *Nie je možné použiť hodnotu semaforu na synchronizáciu (na základe prečítanej hodnoty vykonať nejakú časť kódu)*
  - Po inicializácii sa používa iba inc() a dec()
- Dekrementácia
  - Ak je po dekrementácii výsledok záporný, volajúci musí čakať, kým niekto nezavolá nad semaforom inkrementáciu

1. Semafor

- Inkrementácia
  - Ak v čase inkrementácie jestvuje niekto, kto čaká na pokračovanie (v metóde dekrementácie), bude mu umožnené pokračovať
  - Zvýši sa hodnota semaforu o 1
  - !!! *Ľubovoľný z čakateľov môže pokračovať* !!!
  - Ak máme tzv. silný semafor, ten implementuje FIFO frontu čakajúcich, takže čakanie bude realizované pomocou nej (pokračovať nebude ľubovoľný čakajúci, ale ten, ktorý sa prvý do fronty dostal)

1. Semafor – obmedzenia

- Kým nezavoláme dec(), nevieme, či ostaneme zablokovaní alebo nie
- V inc() po uvoľnení čakajúceho nevieme, kto bude pokračovať a v akom poradí; aj uvoľnený v dec(), aj ten, kto vyvolal inc() môže pokračovať vo svojej činnosti ďalej (**konkurentne**)
- Pri volaní inc() nevieme, či niekto čaká; počet uvoľnených môže byť 0 alebo 1

1. Semafor – stav hodnoty

- > 0
  - Počet vlákien, ktoré môžu zavolať dec() bez toho, aby sa zablokovali
- < 0
  - Počet vlákien, ktoré sú zablokované v dec() a čakajú na uvoľnenie pomocou inc()
- = 0
  - Žiadne vlákno nečaká; až najbližšie volanie dec() bude blokujúce

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. Zámok (Mutex)

- Abstraktný údajový typ (ADT)

- Interný stav: 0 alebo 1
- Metódy na modifikáciu stavu:
  - Inicializácia
  - Uzamknutie
  - Odomknutie

2. Zámok (Mutex)

- Inicializácia
  - Na stav „odomknutý“ (zväčša hodnota 0)
  - *Nie je možné použiť hodnotu zámku na synchronizáciu (na základe prečítanej hodnoty vykonať nejakú časť kódu)*
  - Po inicializácii sa používa iba lock() a unlock()
- Uzamknutie (metóda lock())
  - Ak je v čase vyvolania metódy hodnota semaforu „uzamknutý“ (zväčša 1), volajúci musí čakať na odomknutie

2. Zámok (Mutex)

- Odomknutie (metóda unlock())
  - Hodnota zámku sa nastaví na „odomknutý“
  - !!! *Ľubovoľný z čakateľov môže pokračovat* !!!
  - Ale iba jeden!

  - Ak máme tzv. silný mutex typu Sleeplock (viď ďalej), ten implementuje FIFO frontu čakajúcich, takže čakanie bude realizované pomocou nej (pokračovať nebude ľubovoľný čakajúci, ale ten, ktorý sa prvý do fronty dostal)

2. Zámok (Mutex)

- Zámok je zjednodušenou verziou Semaforu
- Nie je však úplne totožný s binárnym Semaforom

2. Zámok (Mutex)

- Rozdiel oproti Semaforu:
  - Väčšina implementácií
  - Kto zamkne, musí odomknúť!

2. Zámok (Mutex) – implement.

- Spinlock (slabý zámok)
  - Neustále vyťažuje CPU
  - Nemôže prísť ku preplánovaniu procesu!

- Sleeplock (zväčša silný zámok)
  - Nevyťažuje CPU, vyžaduje sa však kooperácia s plánovačom OS!
  - Vyžaduje viac zdrojov (fronta čakajúcich)

2. Zámok (Mutex) – Spinlock

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

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

2. Zámok (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. Zámok (Mutex)

- Implementácia vyžaduje atomickú operáciu pre otestovanie a nastavenie pamäťového miesta

- swap(where, what)
- test_and_set(where)
  - Nastaví pamäťové miesto na 1
  - Vráti pôvodnú hodnotu, ktorá bola pred zmenou

3. Základné synchronizačné vzory

- Signalizácia
- Vzájomná signalizácia (rendezvous)
- Mutex (mutual exclusion)
- Multiplex
- Bariéra
- Znovu použiteľná bariéra (napr. v cykle)
- Fronta

3. Základné synchronizačné objekty

- Semafor / Zámok
- Turniket (turnstile)
- Vypínač (lightswitch)
- Skóre (score board)

Tretie dejstvo

Python

verzia 3.x

Python

- Verzia 3.x
- Prostredie Idle

- print
- komentáre
- operátory +,-,*,**,/,//,%; and, or, not
- premenné

Python – premenné

- Objekt je inštanciou triedy
  - Identita - id()
  - Typ - type()
  - Hodnota
- Premenná
  - Priradenie
  - Porovnanie

Python – typy objektov

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

Python – pretypovanie

- novy_typ(hodnota)

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

Python – nejaké funkcie

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

Python – funkcie

```
def meno_funkcie(parametre):
    príkazy

def novy_riadok():
    print()

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

Python – argumenty

- Pozičné
- Kľúčové

```
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

Cvičenie

- 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

Cvičenie

- 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.

Cvičenie

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

- Potom program „opravte“. Zároveň urobte analýzu, nakoľko váš program dokáže využiť potenciál paralelného behu (bez ohľadu na GIL).

Cvičenie

- Git repozitár (poslať mi pozvánku do zajtrajšieho cvičenia)
- Štruktúra repozitára nasledovná:
  - Každé cvičenie zvlášť vo vetve
  - Označenie vetiev: 01, 02, 03, 04, 05, 06, ..., 10
  - Každá vetva readme.md súbor popisujúci cvičenie (zadanie, riešenie, odpovede na otázky atď)
- PEP8

Cvičenie

- https://www.conventionalcommits.org/en/about/

- https://github.com/thi-ng/umbrella
- https://github.com/danielduarte/diffparse