Provozně ekonomická fakulta

Teoretická informatika
Tomáš Foltýnek
foltynek@pef.mendelu.cz

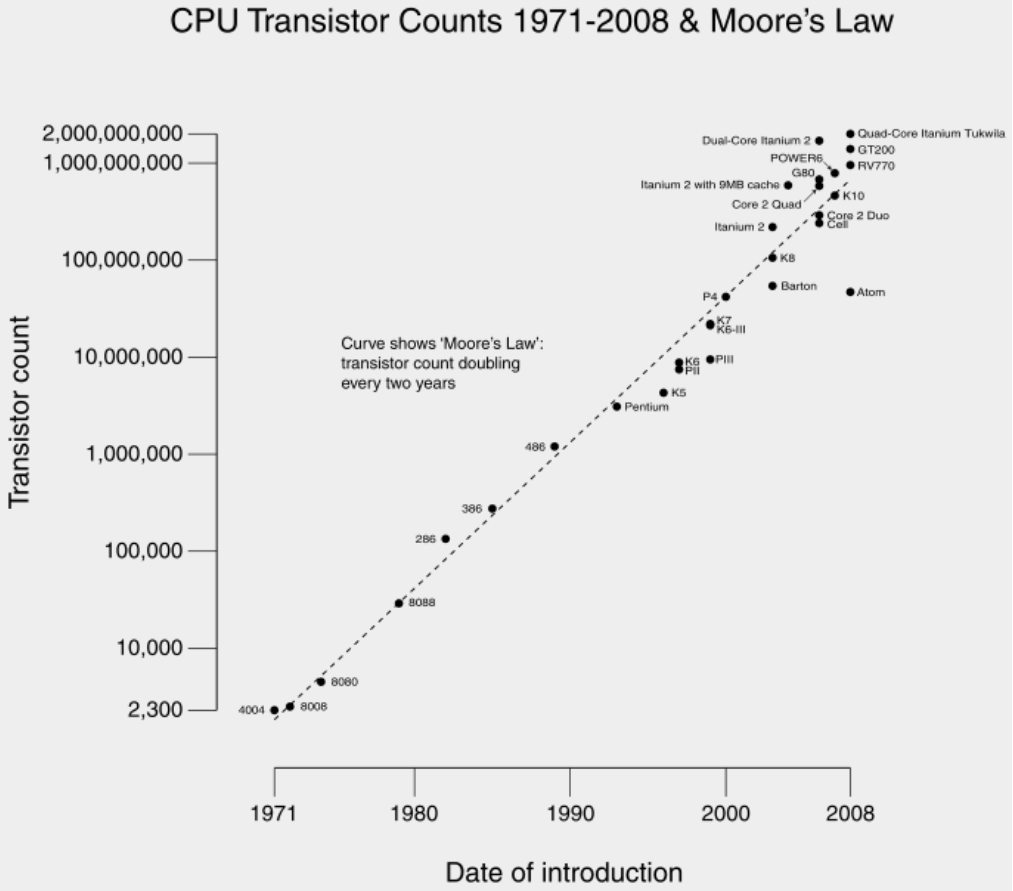
Paralelní programování

Opakování

- Co je to síť?
- Co je to tok? Co je to velikost toku?
- Co je to řez? Co je to velikost řezu?
- Jaký je vztah mezi velikostí toku a velikostí řezu?
- Jak hledat maximální tok?
- Co je to párování v grafu?
- Jak hledat největší párování v grafu?

Motivace

- Moorův zákon: *„Počet tranzistorů, které lze umístit do integrovaného obvodu při zachování ceny, se zdvojnásobuje přibližně každé dva roky“*
  - 1965 Gordon Moore, zakladatel Intelu
- I při neustálém zvyšování výkonu se vždy najdou aplikace, pro které není výkon dostatečný
- Fyzikální limity miniaturizace
- Řešení: paralelizace



Literatura

- Herlihy, M; Shavit, N.: The Art of Multiprocessor Programming
- Grama, A.; Gupta, A.; Kumar, V.: Introduction to Parallel Computing
- Foster, I.: Designing and Building Parallel Programs
- Downey, A.B.: The Little Book of Semaphores. http://greenteapress.com/semaphores/downey08semaphores.pdf
- http://en.wikipedia.org/wiki/Category:Concurrency

Nejrychlejší počítač světa

- www.top500.org
- Cray XT5-HE Opteron Six Core 2.6 GHz
  - National Center for Computational Sciences, Oak Ridge, Tennessee, USA
  - 224162 procesorových jader
  - 1759000 GFLOPS

Základní myšlenka paralelismu

- Sdružíme několik počítačů (procesorů, jader) a rozdělíme výpočetní zátěž mezi ně
- **Paralelní program** je charakterizován dvěma a více souběžně prováděnými a kooperujícími výpočetními aktivitami
  - proces (task), vlákno (thread)
- Kooperace je založena na komunikaci
  - předávání zpráv
  - sdílené proměnné
- Komunikace: nutné zlo zdržující výpočet
  - efektivní řešení komunikace je klíčem k úspěchu

Příklad: skalární součin

- Paralelizace výpočtu skalárního součinu *n*-rozměrných vektorů na *p* procesorech
- Řídicí proces rozdělí vektory na *p* bloků
  - každý s přibližně *n/p* prvky
- Na každém z p procesorů proběhne dílčí výpočet skalárního součinu
- Řídicí proces všechny dílčí výsledky sečte

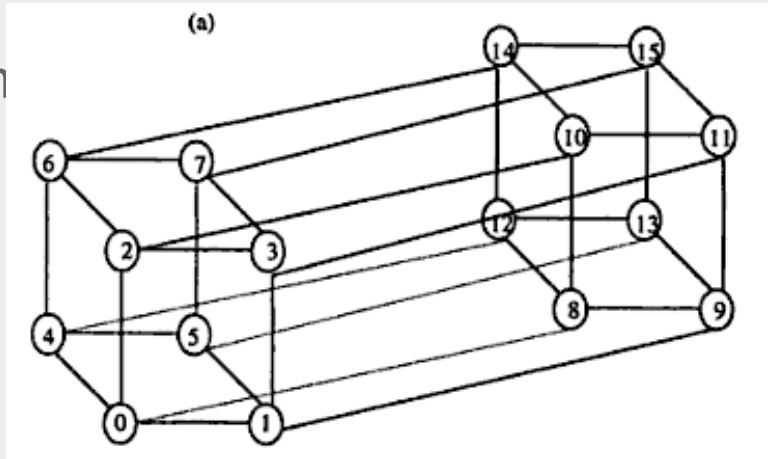
Flynnova klasifikace paralelismu

- Single/Multiple instruction
- Single/Multiple data

- MIMD = Multiple Instruction Multiple Data
  - nejobecnější architektura
  - více instrukčních proudů
  - každý pracuje nad jinými daty
- SIMD = Single Instruction Multiple Data
  - tatáž instrukční sada je provedena na rozdělených datech
  - viz příklad výpočtu skalárního součinu
- SISD = Single Instruction Single Data
  - sekvenční zpracování = bez paralelismu
- MISD = Multiple Instruction Single Data
  - prakticky nepoužitelné

Uspořádání paralelních systémů

- Uspořádání paměti
  - Sdílená paměť
    - všechny procesy přistupují ke stejnému paměťovému prostoru
  - Distribuovaná paměť
    - každý proces má svoji paměť, pro ostatní nedostupnou
- Komunikační topologie
  - Klika – úplný graf
  - Sběrnice – sdílené médium
  - Kruh
  - Strom
  - Hvězdice
    - hvězdice hvězdic
  - Hyperkrychle



Metodika tvorby paralelních aplikací

- **Dekompozice**
  - rozdělení úlohy na podúlohy
  - datová dekompozice (SIMD)
  - funkční dekompozice (MIMD)
- **Komunikace**
  - analýza komunikace mezi podúlohami
  - komunikační schéma (orientovaný acyklický graf/síť)
- **Aglomerace**
  - slučování úloh do větších celků
  - **granularita** (zrnitost) = míra jemnosti podúloh
  - stupeň souběžnosti = maximální počet paralelně běžících úloh
- **Mapování**
  - přidělování úloh procesorům
  - algoritmy vyvažování zátěže

Varianty přístupu k paměti

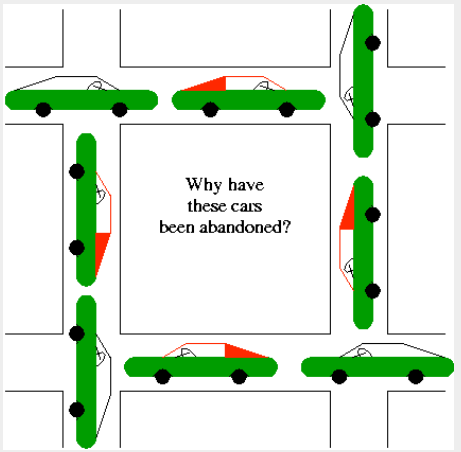
- Souběžné čtení
  - dva nebo více procesů čte z jednoho místa paměti
- Výlučné čtení
  - více procesů čte zároveň z více míst paměti, z každého místa čte jen jeden proces
- Souběžný (soupeřící) zápis
  - dva nebo více procesů zapisuje do stejného paměťového místa
- Výlučný zápis
  - více procesů zapisuje zároveň do více paměťových míst, do každého místa zapisuje jen jeden proces

Problém vzájemného vyloučení

- Proces manipuluje se sdílenými daty
  - Nachází se vůči těmto datům v **kritické sekci**
  - Provádění kritických sekcí musí být **vzájemně výlučné**
- Synchronizační primitiva
  - Bariéra
  - Semafor
  - Monitor
- Viz znalosti z předmětu „Operační systémy“

Požadavky na řešení kritické sekce

- **Bezpečnost** – vzájemné vyloučení
  - v kritické sekci sdružené s daným prostředkem se smí v 1 okamžiku nacházet nejvýše 1 proces
- **Živost** – trvalost postupu
  - vybrat procesu na vstup do KS v konečném čase
  - požadavky každého procesu uspokojit v konečném čase
  - nesmí dojít k **uváznutí** a ke **stárnutí**
- Předpoklady
  - o vzájemné rychlosti procesů nic nevíme
  - o počtu procesů nic nevíme
  - proces bude v KS jen konečnou dobu
  - procesy lze přerušit kdykoliv mimo atomické instrukce



Bariéra

- Všechny procesy jsou zablokovány do chvíle, než dosáhne bariéry poslední
  - Sraz na určitém místě
- Paměťová bariéra
  - instrukce mfence
  - efekt instrukcí provedených před bariérou bude globálně viditelný pro všechny instrukce za bariérou

Instrukce „TEST-AND-SET“

- Algoritmus

```
bool testset(int& i){
  if (i==0) {
    i=1;
    return true
  } else {
    return false
  }
}
```

- Použití

```
while(!testset(b));
CRITICAL_SECTION;
b=0;
```

- Je-li instrukce testset atomická, může být v KS jen jeden proces
  - bezpečné
- Jestliže proces vystoupí z KS, volba dalšího procesu je náhodná
  - procesy mohou stárnout

- Jiná varianta implementace

```
bool TestAndSet(bool
  lock) {
  bool initial = lock;
  lock = 1;
  return initial;
}
```

Instrukce „COMPARE-AND-SWAP“

- Algoritmus

```
bool CAS(int* addr,
  int exp, int val) {
  if(*addr == exp) {
    *addr = val;
    return true;
  }
  return false;
}
```

- Použití

```
do {
  oldval = shared_var;
  newval = NĚCO;
while (CAS(shared_var,
  oldval, newval));
```

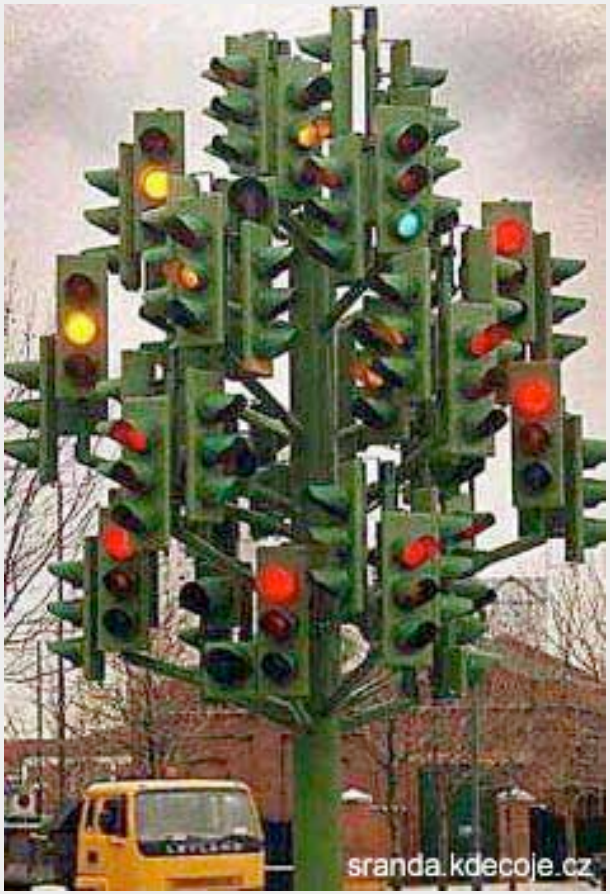
- Použití pro změnu hodnoty sdílených dat
  - přečtu stávající hodnotu
  - připravím novou hodnotu
  - aplikuji funkci CAS
- Návratová hodnota
  - TRUE = objekt nebyl v mezičase změněn, nová hodnota je platná a je uložená
  - FALSE = objekt byl v mezičase modifikován, nová hodnota je neplatná, postup je třeba opakovat

Nevýhoda CAS: ABA problém

- Proces A načte hodnotu x
- Proces B hodnotu změní
- Proces C hodnotu změní opět na x
- Proces A aplikuje CAS
  - ta uspěje, proces nepoznal, že se hodnota měnila
- Nebezpečí vzniku nekonzistentních dat

Semafor I.

- Synchronizační prostředek poskytovaný OS
- Pasivní čekání
- Semafor je objekt
  - celočíselná proměnná
  - fronta čekajících procesů
  - operace inicializace
  - 2 atomické a vzájemně výlučné operace
    - wait
    - signal



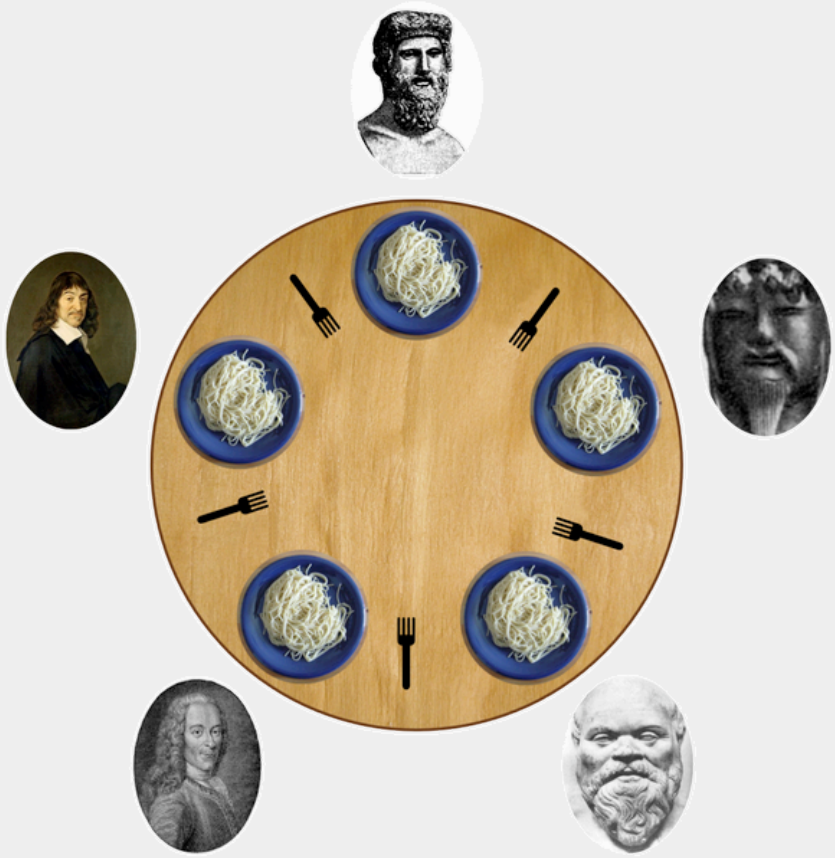
Semafor II.

- S.init
  - S.count=1 //nezáporná hodnota
- S.wait
  - S.count--;
  - if (S.count<0)
    - zablokuj proces
    - vlož jej do fronty čekajících procesů
- S.signal
  - S.count++;
  - if(S.count<=0)
    - vyjmi proces z fronty čekajících procesů
    - odblokuj jej

- Použití semaforu pro vzájemné vyloučení
  - wait(S);
  - CRITICAL_SECTION;
  - signal(S);
- Inicializační hodnota count určuje počet proecsů, které mohou být v kritické sekci
- Atomicita a vzájemná výlučnost wait a signal je zodpovědnost OS
- Použití pro synchronizaci
  - P0::=… signal(s); …
  - P1::=… wait(s); …

Problém večeřících filozofů

- Klasický synchronizační problém
- Přidělování prostředků bez uváznutí a stárnutí
- Máme 5 filozofů, kteří pouze jí a myslí
- Každý filozof umí jíst jen 2 vidličkami
- Máme k dispozici 5 vidliček
- Jak řešit problém?
- Zobecnění pro *n* filozofů, *m* vidliček
  - a *k*-ruké filozofy ☺



Úloha producent / konzument

- Producent produkuje informaci, konzument informaci zpracovává
  - překladač generující moduly zpracovávané sestavovacím programem
- Buffer pro ukládání vyprodukovaných nezpracovaných hodnot
  - neomezené kapacity
  - délky k
- Podmínky
  - k bufferu smí přistupovat jen jediný proces
  - nelze konzumovat, je-li buffer prázdný
  - nelze produkovat, je-li buffer plný

Problém tří kuřáků

- Aby mohla být cigareta vykouřena, vyžaduje tři složky
  - tabák
  - papír
  - zápalky
- Kolem stolu jsou tři kuřáci
  - každý má neomezené zásoby právě jedné složky
- Rozhodčí nekuřák
  - nedeterministicky vybere 2 kuřáky, ti položí 1 položku ze svých zásob na stůl
  - upozorní třetího kuřáka; ten odebere věci ze stolu, ubalí cigaretu a bude chvíli kouřit
- Kuřáci suroviny neshromažďují
  - nejprve dokouřit, teprve potom balit další cigaretu
  - ale během kouření mohou pokládat své věci na stůl

Monitor I.

- Konstrukce na úrovni programovacího jazyka
- Funkčně ekvivalentní semaforu, implementovatelný semaforem
- Monitor je objekt
  - data, k nimž řídíme přístup
  - funkce manipulující s těmito daty
- Podmínky
  - v monitoru se může nacházet jen jeden proces
    - implicitní vstupní semafor
  - data monitoru jsou přístupná jen z jeho funkcí
  - funkce monitoru nepoužívají externí data
  - po skončení funkce jsou data v konzistentním stavu

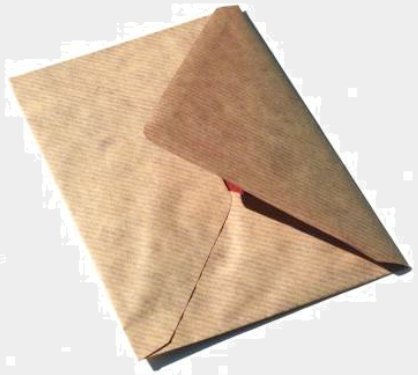
Monitor II.

- Někdy je třeba čekat na událost způsobenou jiným procesem
  - tj. na hodnotu **podmínkové proměnné**
- S každou podmínkovou proměnnou je svázán semafor (a fronta čekajících procesů)
  - wait pouští do monitoru další proces; aktuální proces je odstaven do fronty
  - signal (notify) budí čekající procesy
    - až poté, co proces volající signal opustí monitor

Komunikace

- Dvoustranná komunikace
  - komunikace z bodu do bodu
  - synchronní odesílání
    - ukončeno ve chvíli, kdy příjemce obdrží zprávu
    - odesilatel má jistotu, že příjemce zprávu obdržel
  - asynchronní odesílání
    - ukončeno v okamžiku, kdy je zpráva odeslána
    - odesilatel nemá informaci o doručení zprávy
- Kolektivní komunikace
  - bariéra – synchronizace procesů
  - vysílání (broadcast) – 1 odesilatel, 1 zpráva, více příjemců
  - rozdělení (scatter) – 1 odesilatel, více zpráv, více příjemců
  - sesbírání (gather) – více odesilatelů, více zpráv, 1 příjemce
  - redukce (reduce) – sesbírání dat, vytvoření jediné hodnoty a její rozeslání všem

Zasílání zpráv



- MPI = Message Passing Interface
  - Protokol relační vrstvy
  - Množina funkcí poskytovaná v různých jazycích
- Nejdůležitější funkce
  - MPI_Send, MPI_Recv – blokující odeslání a příjem
  - MPI_Isend, MPI_Irecv – neblokující odeslání a příjem
  - MPI_Bcast, MPI_Scatter, MPI_Gather, MPI_Reduce
  - MPI_Init, MPI_Finalize, MPI_Comm_size, MPI_Comm_rank
  - MPI_Barrier