Praktické paralelní programování

http://www.fit.vutbr.cz/study/courses/public/PPP/
Další informace → Zadání projektů
Garant: Prof. Václav Dvořák
Instruktor: Otto Fučík
fucik@fit.vutbr.cz
http:// www.fit.vutbr.cz/~fucik/
Tel: 4114 1212
Kancelář: 042, Božetěchova 2, Brno
Ústav počítačových systémů FIT VUT v Brně
Konzultační hodiny: Po: 12:30-14:30, St: 11:30-13:30
Přednášky: středa 9:45 – 11:25

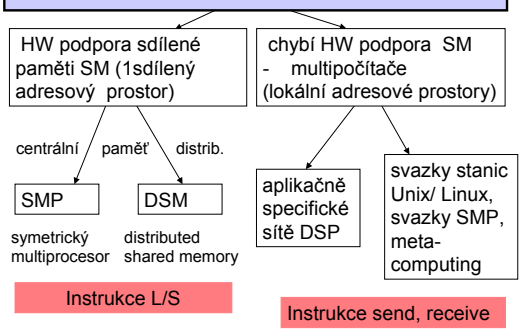
Obsah

1. Paralelní počítače a programování, model PRAM
2. Programování zasíláním zpráv, MPI
3. Programování se sdílenými proměnnými, OpenMP
4. Základní komponenty a vzory programů
5. Případové studie
   (řazení, numerické výpočty, zpracování signálů a obrazů, diskrétní optimalizace)
6. Datově paralelní programování, HPF

Paralelní systémy

- "Paralelní počítač je kolekce procesorů, které kooperují a komunikují, aby řešily velké problémy rychle."
- Pokud procesory sdílejí paměť (shared memory, SM), je komunikace možná tak, že jeden proces zapíše data, druhý (druzí) je čtou. To vyžaduje synchronizaci.
- Při absenci sdílené paměti musí procesy komunikovat zasíláním zpráv (message passing, MP). Synchronizace nastává při komunikaci nebo dodatečně.

Architektury paralelních systémů



Nejvýkonnější současné superpočítače

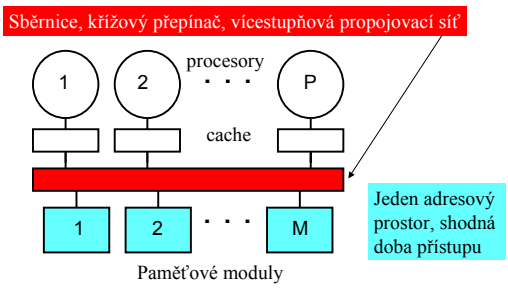
lze nalézt v seznamu TOP 500 na

http://www.top500.org/

Obsahuje i přehledovou zprávu (Overview of Recent Supercomputers, ORSC) nejen o posledních superpočítačích, ale i o mikroprocesorech, aktualizovanou každý rok!

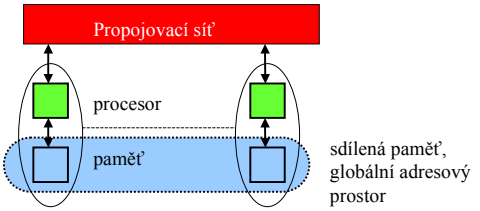
http://www.top500.org/ORSC/2001/

Multiprocesor se sdílenou pamětí (SM)



Distribuovaná sdílená paměť (DSM)

Každý procesor má přístup do celého adresového prostoru. Vzdálený přístup vyžaduje *automatické* zaslání zprávy tam a zpět.



Programování multiprocesoru se SM

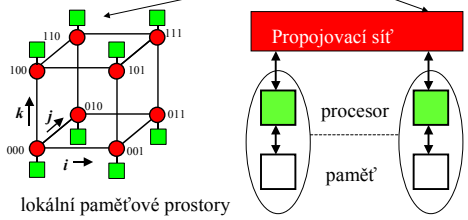
Vlákna obsahují kódové sekvence pro jednotlivé procesory a mohou přistupovat ke sdíleným proměnným. Paralelní programování musí mít příkazy

- pro paralelní běh vláken na jednotlivých procesorech a pro správu vláken (vytváření, rušení, přepínání)
- pro synchronizaci vláken

OpenMP je standardní API obsahující direktivy pro kompilátor, volání knihovních funkcí a proměnné prostředí.

Multiprocesor se zasíláním zpráv (MP)

Kompletní počítače propojené přímo nebo nepřímo



Svazky stanic Unix/Linux COW, NOW

Výhody:

- vysoce výkonné pracovní stanice a PC jsou relativně levné
- do systému lze zabudovat nejnovější procesory jak se objeví
- dá se použít existující nebo modifikované sw
- nastartování komunikace trvá ale 10^1 až 10^2 mikrosekund

SW nástroje pro svazky:

Parallel Virtual Machine (PVM) - vyvinuto koncem 80.let
Message-Passing Interface (MPI) - standard definován v 90. letech

Programování multiprocesoru s MP

Zahrnuje rozdělení problému na části, které se mají provádět současně, aby se došlo k celkovému řešení.

Problém je rozdělen na jistý počet souběžných procesů, větší nebo roven počtu procesorů P. Opět nutná správa procesů.

Obvykle se používá knihovních programů pro zasílání zpráv, které se připojují ke konvenčnímu sekvenčnímu kódu.

Procesy budou komunikovat zasíláním zpráv; je to jediný způsob výměny dat nebo mezivýsledků mezi procesory.

Architektury MP oproti SM

- Programovací model SM je snadnější, protože přenosy dat jsou automatické.
  Důkaz: MP se dá efektivně implementovat na SM, ale opačně to neplatí.
- Na většině strojů se SM jsou tradiční operace MP (send/receive) podporovány sdílenou oblastí paměti. Send do ní zapisuje data nebo ptr, receive data čte.
- Na strojích s MP se dá zařídit sdílený virtuální adresový prostor na úrovni stránek (např. sw vrstva TreadMarks Software Virtual Memory SVM běží na svazcích).

Comparing shared memory and distributed memory programming models

Feature	*Shared Memory*	*Distributed Memory*
Ability to parallelize small parts of an application at a time	Relatively easy to do. Reward versus effort varies widely.	Relatively difficult to do. Tends to require more of an all-or-nothing effort.
Feasibility of scaling an application to a large number of processors	Currently, few vendors provide scalable shared memory systems (e.g., ccNUMA systems).	Most vendors provide the ability to cluster nonshared memory systems with moderate to high-performance interconnects.

Comparing shared memory and distributed memory programming models

Feature	*Shared Memory*	*Distributed Memory*
Additional complexity over serial code to be addressed by programmer	Simple parallel algorithms are easy and fast to implement. Implementation of highly scalable complex algorithms is supported but more involved.	Significant additional overhead and complexity even for implementing simple and localized parallel constructs.

Comparing shared memory and distributed memory programming models

Feature	*Shared Memory*	*Distributed Memory*
Impact on code quantity (e.g., amount of additional code required) and code quality (e.g., the read-ability of the parallel code)	Typically requires a small increase in code size (2-25 %) depending on extent of changes required for parallel scalability. Code readability requires some knowledge of shared memory constructs, but is otherwise maintained as directives embedded within serial code.	Tends to require extra copying of data into temporary message buffers, resulting in a significant amount of message handling code. Developer is typically faced with extra code complexity even in non-performance-critical code segments. Readability of code suffers accordingly.

Comparing shared memory and distributed memory programming models

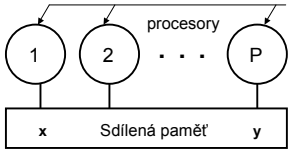
Feature	*Shared Memory*	*Distributed Memory*
Availability of application development and debugging environments	Requires a special compiler and a runtime library that supports OpenMP. Well-written code will compile and run correctly on one processor without an OpenMP compiler. Debugging tools are an extension of existing serial code debuggers. Single memory address space simplifies development and support of a rich debugger functionality.	Does not require a special compiler. Only a library for the target computer is required, and these are generally available. Debuggers are more difficult to implement because a direct, global view of all program memory is not available.

Tvorba paralelních procesů

z podprogramů, funkcí nebo libovolných bloků kódu

1. Rodičovský proces pozastaven, všechny synovské procesy spuštěny současně, rodičovský proces se znovu spustí až všichni potomci skončí (syn - syn)
2. Rodičovský proces zplodí (spawns, *fork*s) jednoho nebo více potomků a všichni pracují souběžně. Volání *join* ukončí synovský proces ale nutí rodiče čekat až všichni potomci ukončí a pak pokračovat. (asyn - syn)
3. Rodičovský proces zplodí potomky a dál se s nimi nezabývá.

PRAM - abstraktní model multiprocesoru



ER - exclusive read
EW - exclusive write
CR - concurrent read
CW - concurrent write
- common value
- arbitrary value
- priority (index of P_i)
- combining

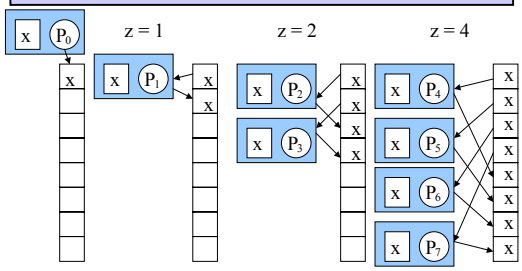
Varianty PRAM :
EREW, CREW, ERCW, **CRCW**

Použití:
teoretická analýza složitosti algoritmů; neexistuje-li efektivní paralelní algoritmus pro PRAM, nebude existovat ani pro reálné počítače.

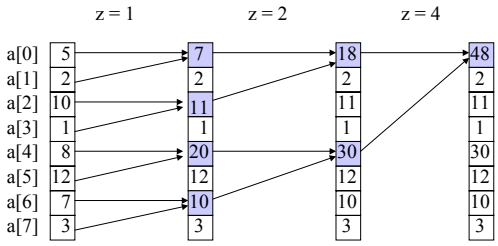
Definice modelu PRAM

- Soubor očíslovaných procesorů $P_0, P_1, P_2, \ldots$ (neomezený)
- Soubor buněk SM $M[0], M[1], M[2], \ldots$ (neomezený)
- Každý procesor P_i má svou privátní lokální paměť (registry) a zná svůj index i
- Každý procesor může atomicky přistupovat do libovolné buňky SM v konstantním (jednotkovém) čase, pokud nedojde ke konfliktu přístupů (viz dále) - vícebránová SM!
- Všechny procesory provádějí **synchronně** instrukci po instrukci nebo NOP. Řídí je centrální jednotka. Instrukce je 3-fázová "Čti-Počítej-Zapiš".
- Vstup (výstup) algoritmu PRAM je v n (n') buňkách SM.

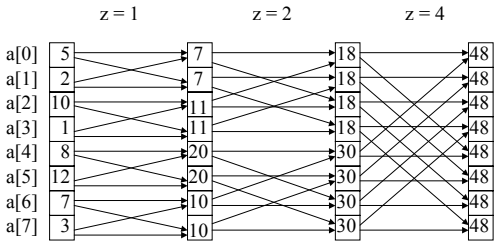
Rozhlašování (broadcast) na EREW PRAM



Součet prvků vektoru na EREW PRAM

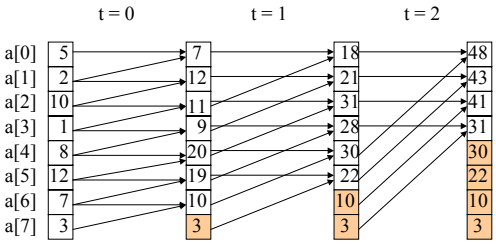


Součet prvků vektoru na EREW PRAM ve všech uzlech

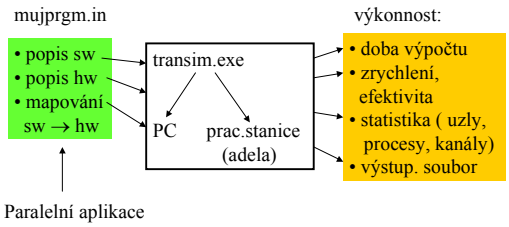


"motýlkový (butterfly) komunikační vzorek"

EREW Paralelní prefixový součet (Parallel Scan, PS)



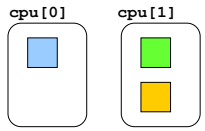
TRANSIM - jazyk pro prototypování a simulaci




```
transim -o report -z seed  mujprgm.in | more
```

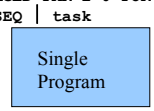
TRANSIM - paralelní procesy, MPMD

```
PLACED PAR
  SEQ | task1
  SEQ | task2
  SEQ | task3
MAP
  MAP cpu[0]: task1
  MAP cpu[1]: task2,task3
```



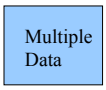
TRANSIM - paralelní procesy, SPMD

```
PLACED PAR i=0 FOR P -- replikovaný příkaz
  SEQ | task
MAP i=0 FOR P
  MAP cpu[i]:task[i]
```



vytvoří vektor P paralelních procesů task

```
task[0]
task[1]
 ...
task[P-1]
```



Paralelní procesy na stejném procesoru:

`PAR` (nelze použít s `i=0 FOR n`)

Transim: sdílené a privátní proměnné

```
INTC P: -- proměnná sdílená v popisu sw i hw
INT a:       -- nad PLACED PAR sdílené proměnné
[20] INT b:
[3] CHAN OF ANY ch: -- deklarace externích
    -- kanálů, sdílených dvojicemi procesů

PLACED PAR i=0 FOR P -- replikovaný příkaz
  INT x:  -- privátní, lokální proměnná v "p"
  CHAN OF ANY p2q,q2p: -- interní kanály
  SEQ | p              -- pojmenování procesu
    ...
  INT y:             -- lokální proměnná v "q"
  SEQ | q
    ...
```

Simulace v Transimu

Výpočty v Transimu (pouze INT) nemají vliv na simulovaný čas. Spotřebu času je třeba explicitně doplnit příkazy
- SERV(taktů) ; reprezentuje užitečnou práci
- WAIT(taktů) ; čekání
- komunikace ; bude podrobně později.

OUTSTRM ! x, y -- každý procesor může zapisovat data do souboru o simulaci
NOTE(jsem.zde) - výpis zprávy s udáním simulovaného času a procesu
Výpisy nemění simulovaný čas.

Paralelní programování

= tvorba paralelního programu k danému algoritmu

- složitější než sekvenční, další záležitosti navíc
- několik programovacích modelů oproti jednomu sekvenčnímu
- softwarové nástroje nejsou tak vyspělé jako sekvenční
- sekvenční programování je ve stavu dospělosti.

další modely	MP, HPF - zasílání zpráv, datově paralelní
rodný model	Power C - sdílené proměnné
HW + OS	SMP - SGI Power Challenge

Přístupy k paralelnímu programování

Čtyři modely paralelního programování, implicitní, zasílání zpráv, sdílené proměnné, datově paralelní se realizují doplněním Fortranu nebo C. Možná rozšíření :

- knihovní podprogramy pro podporu paralelismu a interakcí (MPI, POSIX Pthreads)
- nové konstrukty v jazyce pro podporu paralelismu a interakcí
- direktivy (příkazy) pro kompilátor (pragma)

Rozdíl mezi systémovým a paralelním programováním

Interakce procesů:

soutěžení	kooperace
Procesy soutěží o sdílené prostředky	Procesy komunikují, vzájemně se synchronizují, kombinují částečné výsledky
Systémové programy (OS, síťové nástroje) neukončenost, nedeterminovanost	Paralelní aplikační programy PDE, PDES, B&C dokončenost a determinovanost
Prostředky pro interakci:	
semafor, monitor, producent/ konzument	zámek, bariéra, atomické instrukce

Pokračování příště:
Programování zasíláním zpráv