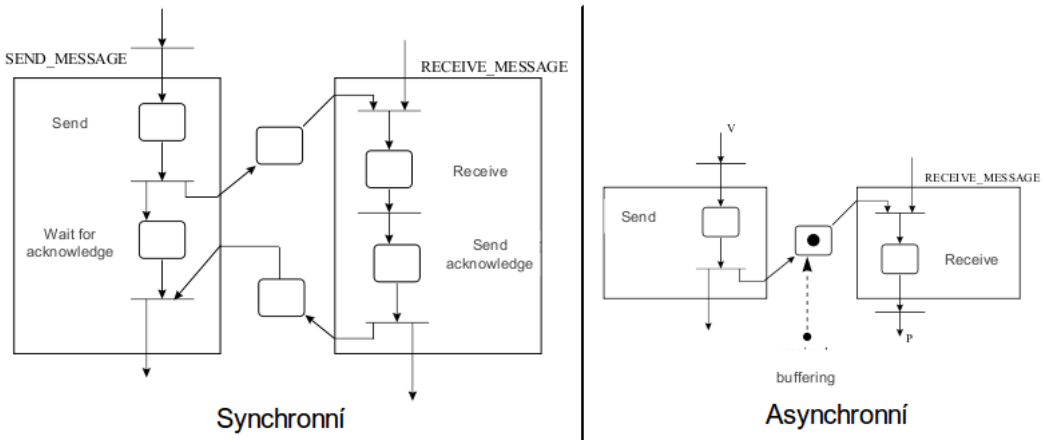
Distribuované a paralelní algoritmy – předávání zpráv a knihovny pro paralelní zpracování (MPI)

Z FITwiki

Obsah

Asynchroní/synchronní komunikace



Paralení výpočty

Situace

- obrovské sítě počítačů propojené rychlým spojením -> je možné používat distribuované výpočty
- potřeba jazyka a protokolu (nejpoužívanější jsou PVM a MPI)

PVM (Parallel Virtual Machine)

- Vytvořen jednou skupinou.
- Distribuovaný operační systém.
- Přenosný mezi HW.
- Heterogenní (různé možnosti reprezentace dat).
- Velká odolnost proti chybám (může se zotavit při výpadku některých stanic)
- Dynamický (přidat/odebrat proces, přidat odebrat stanice, vyrovnání zátěže, chyby).

Implementace PVM

- Démon, bežící na stanicích.
- Démoni spolu komunikují.
- Spojení démonů tvoří jeden výkonný virtuální počítač.
- Démon má pod sebou procesy, kterým je nadřazen.
- První démon je označen jako master.
- Master se stará o nastavení, přidávání, hlídání.
- Výpadek mastera způsobí problémy

MPI (Message Passing Interface)

- Vytvořen fórem firem.
- Knihovna poskytující funkce.
- Přenosný mezi HW a SW (je to knihovna).
- Heterogenní (zabalení různých dat do daných typů).
- Zaměřen na vysoký výkon.
- Přesně definované chování.
- Statický (pevný počet stanic, pevný počet úkolů, vyšší výkon).
  - MPIv2 zavádí možnost začínat a ukončovat skupiny úkolů
- Není odolnost proti chybám (pokud vypadne jedna stanice, je nutný výsledky zahodit).

Implementace MPI

- Na každém počítači běží procesy (jeden na CPU).
- Procesy mají identifikaci.
- Procesy znají ID ostatních procesů.
- Procesy komunikují mezi sebou přímo.
- Proces neumí fork() (MPIv1)
-

Kooperativní komunikace

- `send()` a `recv()`
- odesílatel i příjemce se přímo podílí na komunikaci

Jendostranná komunikace (One-sided)

- `Put()` a `Get()`
- pouze jeden proces se přímo podílí na komunikaci
- zápis/čtení z paměti druhého procesu bez jeho účasti
- součást MPIv2

Informace o prostředí

- kolik je dostupných procesorů?
- který procesor jsem já?

Základní koncepty MPI

- procesy mohou být seskupenu do skupin (groups)
- zprávy se vždy předávají a přijímají v daném kontextu (existuje výchozí kontext obsahující všechny procesy, MPI_COMM_WORLD)
- skupina a kontext = komunikátor
- proces je identifikován jeho rankem (jednoznačné ID) ve skupině příslušející komunikátoru
- zprávy mohou být označeny tagy, které pomáhají příjemci identifikovat zprávu (někdy nazýváno message types)
- podporuje blokující i neblokující zasílání zpráv
- podporují kolektivní operace (broadcast, reduce)

Datové typy

- data ve zprávách reprezentovány jako trojice (address, count, datatype)
- umožňují předávat data mezi platformami používajícími různé reprezentace dat
- datové typy
  - základní typy (MPI_INT, MPI_DOUBLE)
  - contiguous array (co to je??)
  - strided block (co to je??)
  - indexovaná pole
  - struktury

Synchronizace
: využívá se bariér (procesy čekají dokud všechny nedojdou k bariéře)

Kolektivní operace

Broadcast
: data poslána z jednoho procesu všem ostatním

Reduce
: data získána ze všech procesů, zkombinována a předána jednomu procesu

Scatter
: data z jednoho procesu rozdělena na části mezi všechny procesů

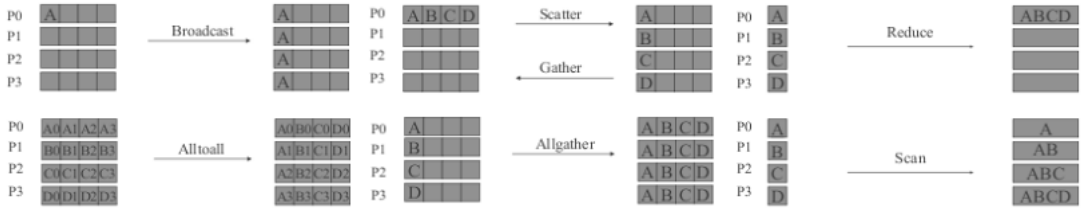
Gather
: části dat z různých procesů spojeny do jednoho procesu

Allgather
: části dat z různých procesů spojeny a předány všem procesům (gather+broadcast)

AllToAll
: data ze všech procesů rozdělena na části, příslušené řásti spojeny a předány konkrétním procesům (1 proc. první části, druhý drudé části, ...)

Scan
: každý proc obdrží zkombinovaná data předchozích procesů (a=a, b=a+b, c=a+b+c, ...)

Agregační funkce
: min, max, average, sum, and, or, xor, ...



Knihovny/jazyky

Někdo to pls zkontrolujte za správnost neručím je to vyčteno jen ze slides

OCCAM

- jazyka založený na formalismu Communicating Sequential Processes (CSP)
- imperativní procedurální jazyk
- výpočet reprezentován sítí komunikujících procesů

Základní prvky jazyka (procesy)

- přiřazení

```
x := y + 2
```

- vstup

```
keyboard ? char
```

- výstup

```
screen ! char
```

Komunikační kanály
: jeden proces do něj zapisuje ostatní čtou

Sekvence SEQ

- jednotlivé procesy jsou provedeny sekvenčně

```
SEQ
 proces
 proces
```

Paralelní spuštění PAR

- podprocesy jsou spuštěny paralelně
- proměnné zapisované v některé větvi nemohou být v ostatních větvích čteny/zapisovány

```
PAR
 proces
 proces
```

Replicated PAR/SEQ

- spuštění procesu v PAR/SEQ v několika exemplařích

```
PAR i = 0 FOR 4
  proces
```

Výběr ALT

- provede se pouze jeden z podprocesů (podle podmínky - guard)

```
ALT
 left ? packet -- guard
    proces 1
 right ? packet —- guard
    proces2
```

ADA

- používá mechanismus **randevousz**
- **Task** - oddělený sekvenční proces s lokálními daty
- tasky komunikují voláním vstupních procedur jiných tasků (volání i přijetí volání může čekat dokud není druhý připraven)

Accept

- definuje vstupní proceduru
- vnitřek procedury je spouštěn pouze při přijetí volání vstupní procedury
- parametry volání jsou předány do procedury

```
accept <entry-name> (<formal parameter list>) do
 ...
end;
```

Select

- čekání na accept více různých signálů
- podmíněná dostupnost vstupních procedur

```
select
   when <podmínka>
     accept entry1...
   or when <podmínka>
     accept entry2...
   else
```

end select;

Linda

- založeno na jazyce C
- **Tuple space** virtuální sdílená paměť, nástěnka na kterou procesy vyvěšují data a na které data hledají
- **tuple** - datová položka vkládaná na nástěnku, skládající se ze sekvence hodnot a formálních polí (slouží k vyhledávání tuple)
- asynchronní komunikace mezi procesy (proces vyvěsí data, jiný proc si data někdy později vezme)

out

- parametry jsou vyhodnoceny a výsledek je poté uložen na nástěnku
- volající ihned pokračuje, neblokující
- příklad out ('array', dim1, dim2)

eval

- pro každý parametr spustí proces, ten vyhodnotí parametr
- tyto vyhodnocené parametry jsou pak složeny do výsledné n-tice, která je vložena na nástěnku
- volající ihned pokračuje, neblokující
- ALE nechá po sobě spuštěné procesy, které vyhodnocují parametry a nevíme za jak dlouho skončí
- příklad eval ("test", F(param1), G(param2))

in

- vyhledá tuple odpovídající šabloně
- tuple je načten a odstraněn z tuple space
- blokuje dokud příslušný tuple není načten (tj. může čekat na jeho vygenerování)
- použitelné k synchronizaci mezi procesy
- existuje i v neblokující podobě inp, která se ihned vrátí

rd

- vyhledá tuple odpovídající šabloně

- tuple ja načten, ale je ponechán v tuple space
- blokuje dokud příslušný tuple není načten (tj. může čekat na jeho vygenerování)
- použitelné k synchronizaci mezi procesy
- existuje i v neblokující podobě rdp, která se ihned vrátí

- Stránka byla naposledy editována 6. 6. 2011 v 12:45.