

# Párhuzamos és Grid rendszerek (2. ea)

párhuzamos algoritmusok tervezése

Szeberényi Imre  
BME IIT

<szebi@iit.bme.hu>

Az ábrák egy része Ian Foster: *Designing and Building Parallel Programs* (Addison-Wesley) c. könyvéből származik.



## Hol tartunk ?

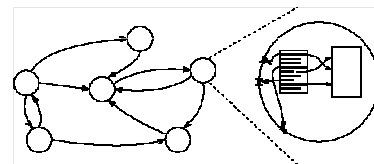
- Megismerkedtünk az alapfogalmakkal
- Megismertük a fontosabb párhuzamos architektúrákat:
- SMP (NUMA, ccNUMA)
- MPP
- CLUSTER

## Hol tartunk ? /2

- Felvázoltunk fizikai és logikai összeköttetéseket
- Egyszerű absztrakciós modellt alkottunk a párhuzamos gépek leírására
- Említett programozási modellek
  - Közös memóriás
  - Elosztott közös memóriás
  - Üzenet küldéses

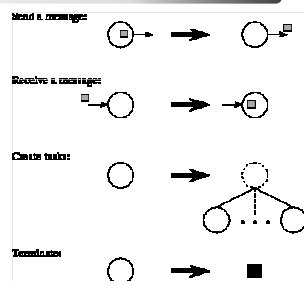
## Taszk/csatorna modell /1

- minden taszk szekvenciális programot futtat
- minden taszknak van saját memóriája
- taszkok csatornákkal kapcsolódnak
- a csatornák üzenetsorokat valósítanak meg



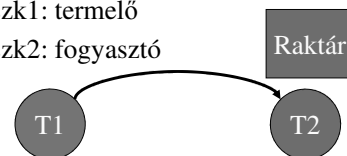
## Taszk/csatorna modell /2

- taszkok konkurenssek
- van lokális memóriájuk
- küldés aszinkron
- fogadás szinkron
- csatornához in/out portokkal csatlakoznak
- taszkok tetszőlegesen rendelhetők össze a processzorokkal



## Taszk/csatorna modell /3

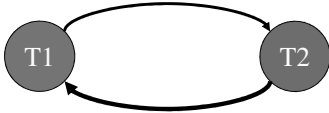
- Példa: termelő-fogyasztó probléma
  - taszk1: termelő
  - taszk2: fogyasztó



- ha a fogyasztó lassabb, akkor a felhalmozódik a termelt adat
- ha a termelő a lassabb, akkor vár a fogy.

## Taszk/csatorna modell /4

- Példa: termelő-fogyasztó probléma
  - taszk1: termelő
  - taszk2: fogyasztó



- második csatornán a fogyasztó jelzi, ha kér újabb adatot
- a termelő ennek hatására termel

## Taszk/csat. modell jellemzői

- A modell közvetlenül hozzárendelhető az idealizált számítógéphez.
- A taszk egy soros kódot reprezentál.
- A csatorna processzorok közötti kommunikációt valósít meg.
- A taszk működése független a taszk-processzor összerendeléstől, taszkok számától.
- Moduláris felépítést tesz lehetővé.

## Taszk/csatorna vs. üzenet

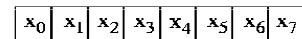
- Az üzenet egy adott taszknak szól, ezért kevésbé absztrakt, mint a csatorna.
- Az általános üzenetküldéses modell szerint nem lehet dinamikusan új taszkat létrehozni. (Több megvalósításban lehet.)
- Egy processzor csak egy taszkat futtathat. (Több megvalósításban ez sem korlát.)

## Párh. algoritmus példák /1

- Véges differenciák:
  - egy vektor minden elemére T-szer végre kell hajtani a következő műveletet:

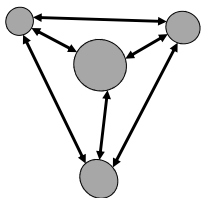
$$0 < i < N-1, 0 \leq t < T : x_i^{(t+1)} = \frac{x_{i-1}^{(t)} + 2x_i^{(t)} + x_{i+1}^{(t)}}{4}$$

- Minden elemet egy-egy taszk számol, aki kommunikál a szomszédaival:



## Párh. algoritmus példák /2

- Páronkénti iteráció (pl. atomok kölcsönös egymásra hatása)



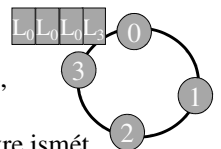
$$f_i = \sum_{j=0}^{N-1} F(\bar{X}_i, \bar{X}_j)$$

- $N*(N-1)$  üzenet kell, esetleg  $N*(N-1)/2$ , ha kihasználjuk a szimmetriát.

## Párh. algoritmus példák /3

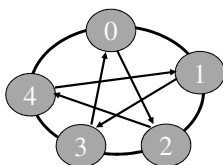
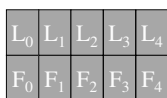
- Körkörös kapcsolat (csatorna) a fenti problémára hatékonyabb üzenetstruktúrát eredményez:

- Egy  $N$  elemű vektorba minden taszk betesz a saját adatát (koord., tömeg) és elküldi a szomszédnak.
- A bejövő üzenetbe megfelelő helyre ismét elhelyezi a saját adatát és továbbküldi azt.
- $N-1$  lépés után mindenki ismeri az a többiek koordinátáit és tömegét.
- $F$  értéke minden lépésben az új partnerek adata alapján akumulálható.



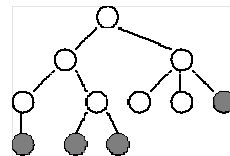
## Párh. algoritmus példák /4

- N újabb csatornával az algoritmus a szimmetria miatt tovább egyszerűsíthető:
  - hozzunk létre minden  $i$ . taszk és  $i+N/2$ -dik taszk között egy újabb csatornát.
  - az adott atomra ható erőket folyamatosan számoljuk, és küldjük is körbe.
  - $N/2$  iterációval előáll az eredmény.

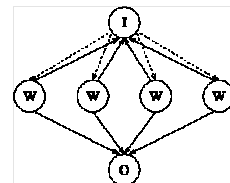


## Párh. algoritmus példák /5

- Párhuzamos keresés:
  - fában történő keresés egyszerűen párhuzamosítható



- Paraméter elemzés:
  - master-worker algoritmus

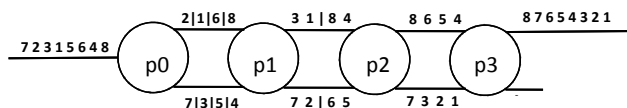


## Párh. algoritmus példák /6

- Pipeline rendezés:
  - minden elem megtartja a nagyobbát



- Pipeline merge sort

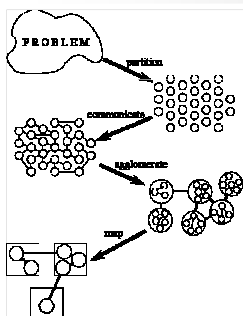


## Párh. algoritmusok tervezése

- Nem egyszerű.
- Kreativitást igényel.
- Számos iterációt tartalmaz.
- Nincs egyszerű recept.
- Vannak betartható, ajánlott lépések, módszerek.



## PCAM módszertan



1. Particionálás: Részfeladatokra osztás. NEM veszi figyelembe a fizikai gép adottságait.
2. Kommunikáció megtervezése: Részfeladatok közötti adatsere és szinkronizációs séma kialakítása.
3. Agglomeráció: Részfeladatok nagyobb egységekbe gyűjtése a hatékonyságnövelés érdekében.

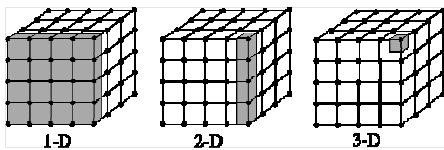
4. Leképezés: A részfeladatok processzorhoz (feldolgozó elemhez) rendelése.

## Particionálás

- Cél: Párhuzamosítható részek felderítése. A művelet absztrakt, nem veszi figyelembe párhuzamos környezet HW/SW adottságait.
- Finom felbontás (sok kis részfeladat) előállítása hatékonyabb és egyszerűbb.
- A feladatot és az adatokat is kis részekre szedjük.
  - *domén dekompozíció*
  - *funkcionális dekompozíció*

## Domén dekompozíció

- Adat vagy paramétertér felosztása. Az adat lehet input, output, vagy közbülső adat.
  - Példa: Egy 3D rácson minden rácspontban ki kell számolni egy értéket. 1, 2, vagy 3 dimenziós partíció:



## Funkcionális dekompozíció

- Az algoritmus felosztása olyan részekre, melyek párhuzamosíthatók.
- Alapvetően a feladat funkcióiból adódik.
- Az adatokra is figyelni kell.
- Tipikus példa, amikor az adatok partícionálása nem járható: keresés fában.
  - funkcionálisan viszont bontható

## Hogy sikerült a partícionálás?

- Jól, ha partícionálással kapott taszkok száma nagyságrendileg több mint a proc. száma.
- Jól, ha redundancia mentes.
- Jól ha a taszkok mérete hasonló.
- Jól, ha a probléma méretével a taszkok száma is nő.

## Kommunikáció

- Kis környezetű (local) és globális
  - a taszkok csak kis környezetükben (szomszéd), vagy sok másik taszkkal is kommunikálnak.
- Strukturált és nem strukturált
  - rács, gyűrű, ... vagy más
- Statikus és dinamikus
  - végrehajtás közben változik
- Szinkron vagy aszinkron
  - koordináció hiánya

## Kommunikációs példák /1

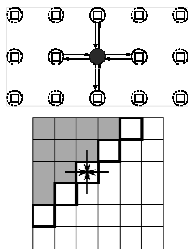
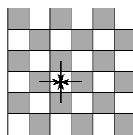
Lokális kommunikáció (Jakobi):

$$\bar{x}_{i,j}^{(t+1)} = \frac{4\bar{x}_{i,j}^{(t)} + \bar{x}_{i-1,j}^{(t)} + \bar{x}_{i+1,j}^{(t)} + \bar{x}_{i,j-1}^{(t)} + \bar{x}_{i,j+1}^{(t)}}{8}$$

(Gauss-Seidel):

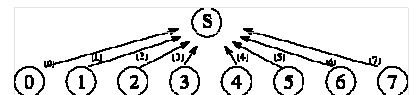
$$\bar{x}_{i,j}^{(t+1)} = \frac{4\bar{x}_{i,j}^{(t)} + \bar{x}_{i-1,j}^{(t+1)} + \bar{x}_{i+1,j}^{(t)} + \bar{x}_{i,j-1}^{(t+1)} + \bar{x}_{i,j+1}^{(t)}}{8}$$

Red-Black ordering:

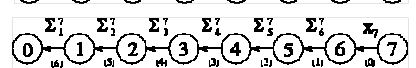


## Kommunikációs példák /2

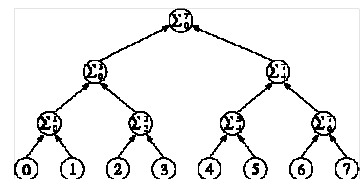
Globális kommunikáció (szumma):



Csővezeték:



Oszd meg és uralkodj:

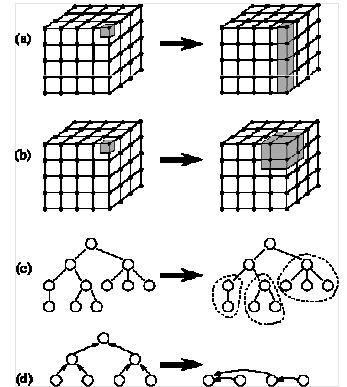


## Hogy sikerült a kommunikáció?

- Jól, ha közel azonos számú kommunikációt végez minden taszk.
- Jól, ha a taszkok csak lokális környezetükkel kommunikálnak.
- Jól, ha kommunikáció konkurensen párhuzamosan zajlik.
- Különböző taszkok konkurensen kommunikálnak.

## Agglomeráció

- A tényleges párhuzamos gép kommunikációs adottságait is figyelembe véve a részfeladatokat nagyobb egységekbe gyűjtjük.



## Agglomeráció szükségessége

- A kommunikáció "költséges"
- A kommunikáció szükségtelen szinkronizációt okoz
- Térfogat-felület effektus (számítás/kommunikáció arány)
- Flexibilitás megtartása

## Hogy sikerült az agglomeráció?

- Jól, ha jelentősen növekedett a lokális kommunikáció
- Jól, ha a skálázhatóság nem romlott.
- Jól, ha az összevont taszkok mérete közel azonos.
- Jól, ha a probléma méretével növekszik a taszkok száma.
- Jól, ha a már nem vonhatók össze feladatok anélkül, hogy a skálázhatóság vagy a terhelés kiegyenlíthetőség ne romlana.

## Leképezés (mapping)

- Tényleges HW/SW környezet figyelembe vétele, leképezés a fizikai gépre.
- Jelentősen befolyásolhatja a terhelés kiegyenlítést, ütemezési algoritmust.

## Hogy sikerült a leképezés?

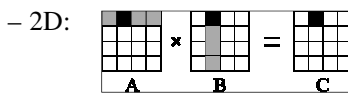
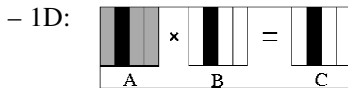
- Jól, ha nem keletkezett szűk keresztmetszet a programban.
- Jól, ha több lehetséges leképezést is megvizsgáltunk.
- Ha figyelemmel voltunk a terhelés kiegyenlítésre.

## Komplex példa: mátrix szorzás /1

Mátrix-mátrix szorzás:  $O(N^3)$   $C_{ij} = \sum_{k=0}^{N-1} A_{ik} \cdot B_{kj}$ .

- Partícionálás:

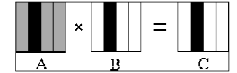
- egy  $C_{ij}$ -t csak egy processzor számoljon
- domén dekompozíció



## Komplex példa: mátrix szorzás /2

- Kommunikáció

- 1D: Minden proc.-nak szüksége van a teljes A-ra.
- tfh. egy processzor felelős az adatok szétküldéséért és begyűjtéséért (pl. SPMD)
- $T_{1d} = (P-1) \cdot (N^2 + 2 \cdot N^2/P) \approx P \cdot N^2$
- A kommunikációs igény processzor számával lineárisan nő!



## Komplex példa: mátrix szorzás /2

- Kommunikáció

- 2D: Azonos sor ill. azonos oszlop kell A-ból és B-ből.

- Algoritmus (Fox's):

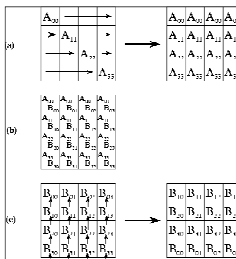
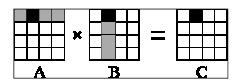
1.  $C = 0$

2. Ismétlés  $N-1$ -szer:

3. A átlót soronként ismételve tegyük egy  $A'$ -be

4.  $C_{ij} = C_{ij} + B_{ij} * A'_{ij}$

5.  $B = B$  ciklikus feltolása



- $T_{2d} = 2 \cdot (\sqrt{P}-1) \cdot N^2 + (P-1) \cdot N^2/P \approx N^2/\sqrt{P}$

## Komplex példa: mátrix szorzás /4

- Agglomeráció:

- A kommunikációs igény felmérésénél láttuk, hogy érdemes lehet nagyobb csoportokat alkotni.
- Esetleg más algoritmusok (pl. Cannon) más agglomerációt igényelhetnek.

- Leképezés (mapping):

- Rács
- Fa