Párhuzamos algoritmusok tervezésének alapjai

A. Grama, A. Gupta, G. Karypis és V. Kumar: „Introduction to Parallel Computing”, Addison Wesley, 2003

könyv anyaga alapján

Vázlat

- Bevezetés
  - Részfeladatok és dekompozíció
  - Processzek és leképzés
- Dekompozíciós technikák
  - Adat dekompozíció
  - Rekurzív dekompozíció
  - Felfedező dekompozíció
  - Spekulatív dekompozíció
- Leképzési technikák és terhelés kiegyensúlyozás
  - Statikus és dinamikus leképzés

Cél

- A párhuzamosítás maximalizálása
- A párhuzamosításból következő extra tevékenységek (overhead) redukálása
- A potenciális sebesség- és teljesítmény-növelés maximalizálása

Fő lépések a cél irányában

- A párhuzamosan végezhető munkarészek meghatározása
  - részfeladatok
- A részfeladatok processzorokhoz rendelése, leképezés
  - processzek *vs.* processzorok
- Az input/output & közbenső adatok különböző processzorok közötti szétosztása
- A megosztott adatok menedzselése
  - input vagy közbenső adatok
- A processzorok szinkronizálása a párhuzamos futás különböző pontjain

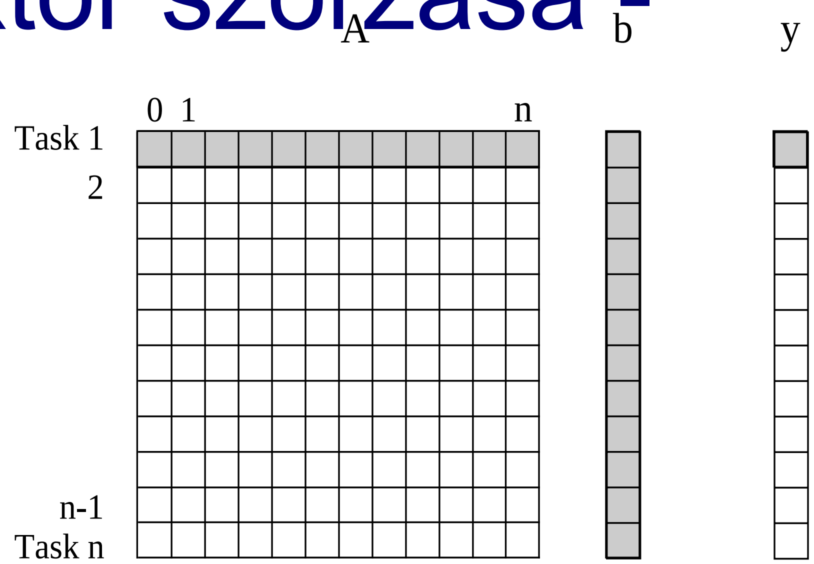
Bevezetés

Dekompozíció, részfeladatok,
függőségi gráfok, processzek

Dekompozíció, részfeladatok és függőségi gráfok

- Párhuzamos algoritmusok fejlesztésekor az első lépés a probléma olyan részfeladatokra bontása, amelyeket majd párhuzamosan végrehajthatunk
- Az adott probléma többféleképpen bontható szét
- A részfeladatok lehetnek azonos, vagy eltérő méretűek, vagy akár szakaszosak
- A dekompozíciót irányított gráffal lehet reprezentálni, ahol a csomópontok részfeladatokat jelentenek, az élek pedig azt mutatják, hogy a részfeladat eredménye szükséges a következő rész feldolgozásához: *feladat függőségi gráf*

Sűrű mátrix és vektor szorzása - példa



- Az *y* output vektor minden elemének számítása független a többi elemétől. A sűrű mátrix-vektor szorzatot *n* részfeladatra lehet bontani.

Megfigyelések:

- A részfeladatok megosztanak adatot (*b* vektor), de nincs közöttük semmilyen vezérlési kapcsolat
- A végrehajtáskor semmilyen részfeladatnak sem kell várnia másikra
- Minden részfeladat ugyanolyan méretű (műveletek száma)
- Ez a maximális számú részfeladat, amire szét tudtuk bontani a problémát?

Adatlekérdezés feldolgozása - példa

A következő lekérdezést dolgozzuk fel:

MODEL = ”CIVIC” AND YEAR = 2001 AND

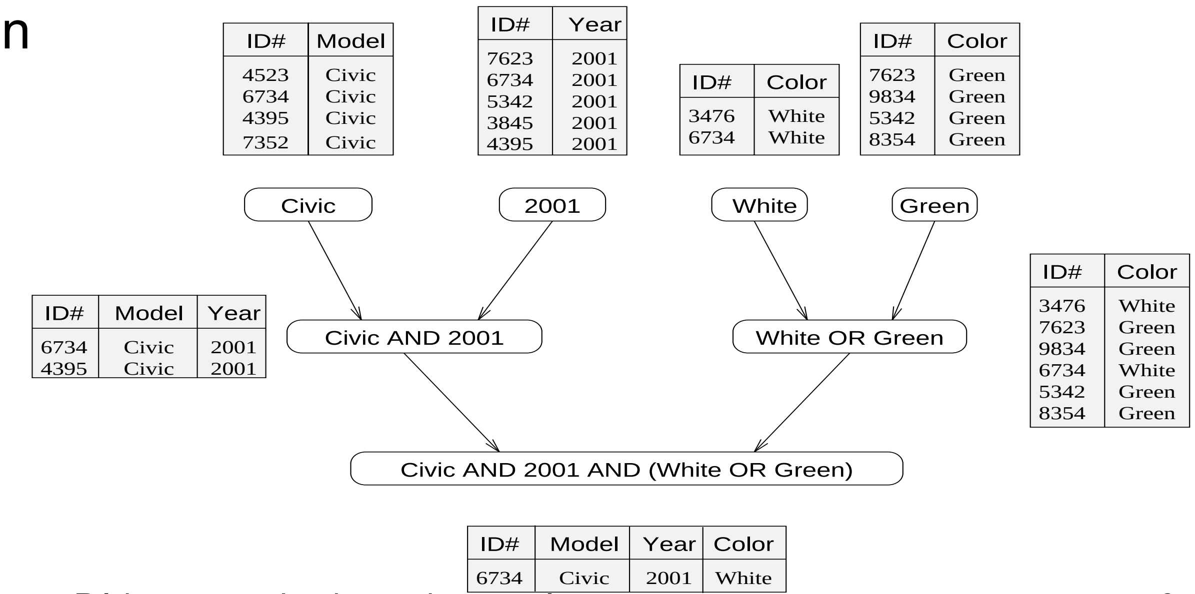
(COLOR = ”GREEN” OR COLOR = ”WHITE”)

Az adatbázis:

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

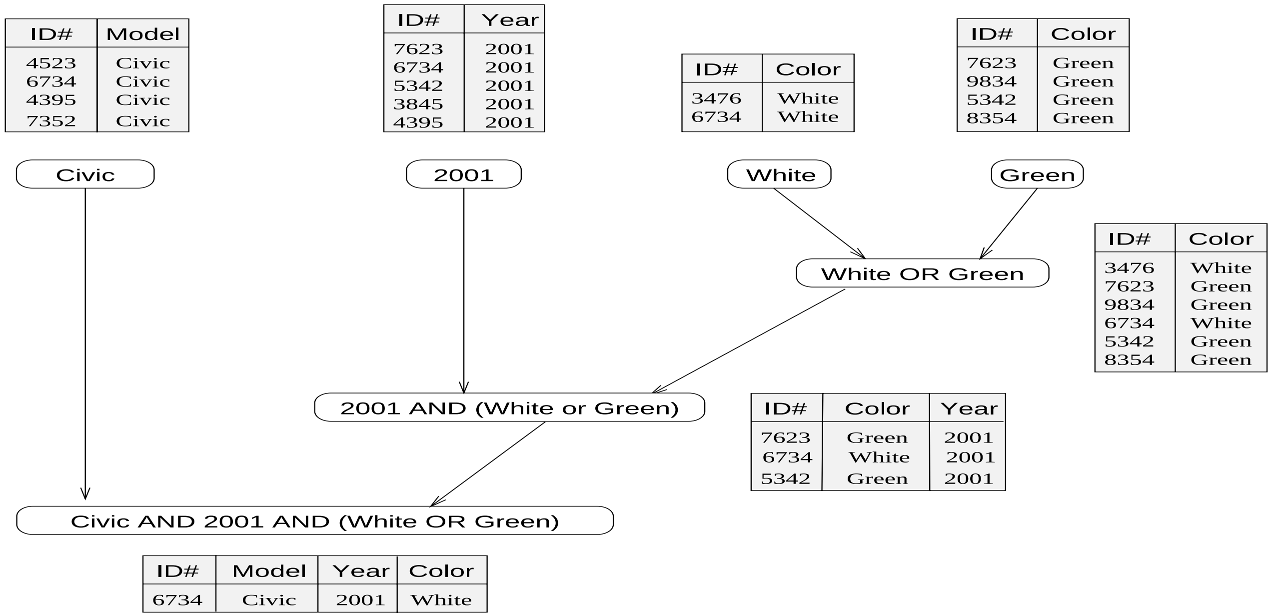
Adatlekérdezés feldolgozása (példa)

- A lekérdezés megvalósítását – különböző módon – részfeladatokra lehet bontani
- Minden részfeladat egy közbenső táblát generálhat, amely bizonyos kikötésnek tesz eleget
- A gráf élei azt jelentik, hogy a részfeladat outputja szükséges a következő lépésben



Adatlekérdezés feldolgozása - példa

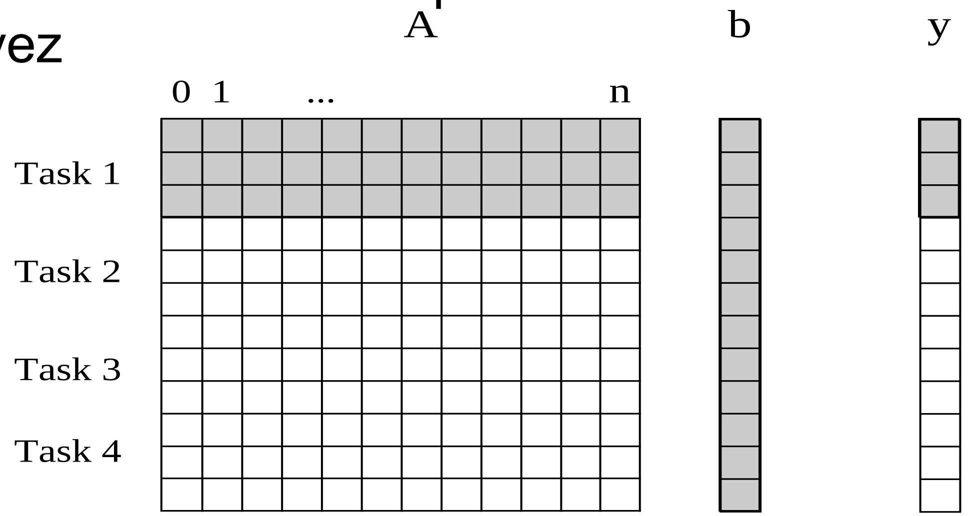
- Ugyanaz a probléma más úton is részfeladatokra bontható (jelen esetben az adatfüggőség szerint)



- Az különböző részfeladatokra bontás szignifikáns teljesítmény eltérésekhez vezethet a párhuzamos működés során

A dekompozíció szemcsézettsége

- A szemcsézettséget az határozza meg, hogy hány részfeladatra osztjuk a problémát
- Sok részfeladatra bontás *finom szemcsézettséget*, kevés számú részfeladatra történő dekompozíció *durva szemcsézettséget* eredményez



Példa: Minden részfeladat az eredményvektor három elemét határozza meg

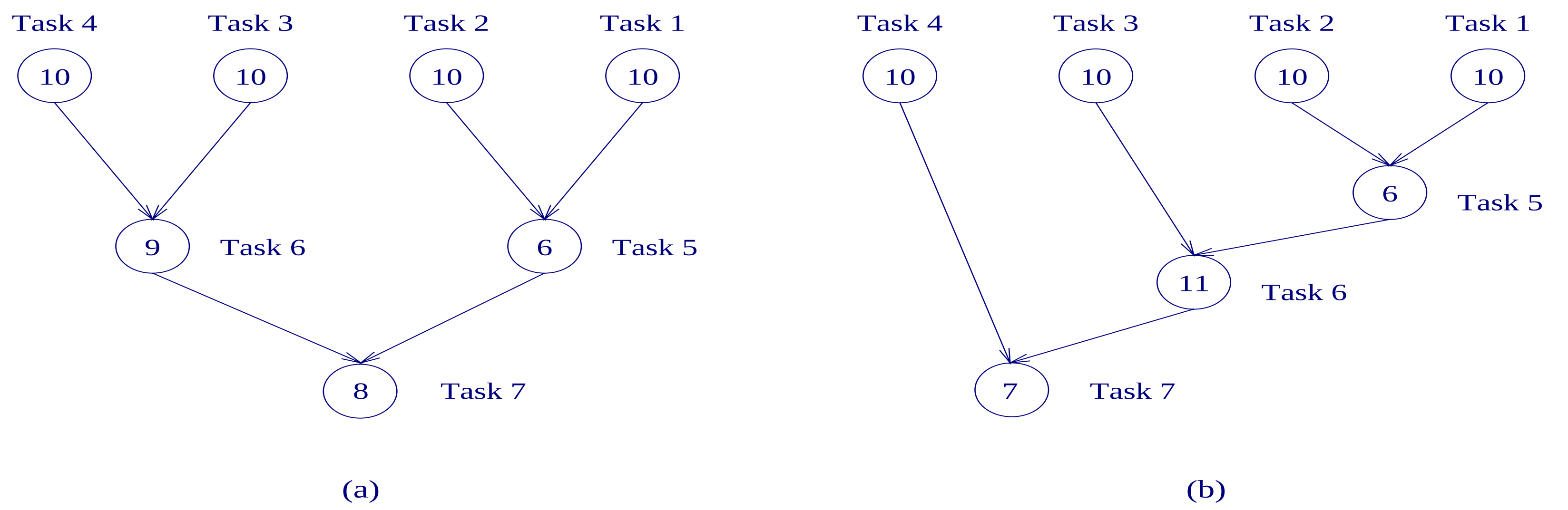
Párhuzamossági fok

- A párhuzamosan futatható részfeladatok száma határozza meg a dekompozíció *párhuzamossági fokát,* vagy *konkurencia fokát*
- Mivel a program futása során a párhuzamosan futó feladatok száma változhat, a *maximális párhuzamossági fok:* bármely pillanatot tekintve a legtöbb ilyen részfeladat
- Az *átlagos párhuzamossági fok* az átlaga azoknak a részfeladatoknak, amelyeket párhuzamosan lehet végrehajtani a program futása során
- A párhuzamossági fok növekszik, ha a a dekompozíció szemcsézettsége finomodik

Kritikus út hossza

- A feladat függőségi gráfban egy (irányított) út egymás után végrehajtandó részfeladatokat reprezentál
- A részfeladatoknak költsége (ideje) van
- A leghosszabb ilyen út határozza meg a program legrövidebb futás idejét
- A feladat függőségi gráfban a leghosszabb út hossza a kritikus út hossza

Kritikus út hossza



- Ha minden részfeladat a jelölt időegységig tart, akkor mennyi a legrövidebb futási idő a két dekompozíció esetén?
- Mennyi a maximális párhuzamossági fok?
- Mennyi az átlagos párhuzamossági fok?
- Hány processzor szükséges a két esetben a minimális idejű futáshoz?

A párhuzamos teljesítmény korlátai

- Úgy tűnhet, hogy a párhuzamosítás következtében tetszőleges kicsi lehet a futási idő a dekompozíció finomításával
- De a finomításnak természetes korlátai vannak (A sűrű mátrixos feladatban pl. (n^2) párhuzamos részfeladat lehet maximum)
- A párhuzamos részfeladatok adatot cserélhetnek egymással, amely szintén extra kommunikációs ráfordítást jelent
- A dekompozíció szemcsézettsége és a párhuzamossá alakítás extra ráfordítása közötti kompromisszum szintén meghatározhatja a teljesítmény korlátokat

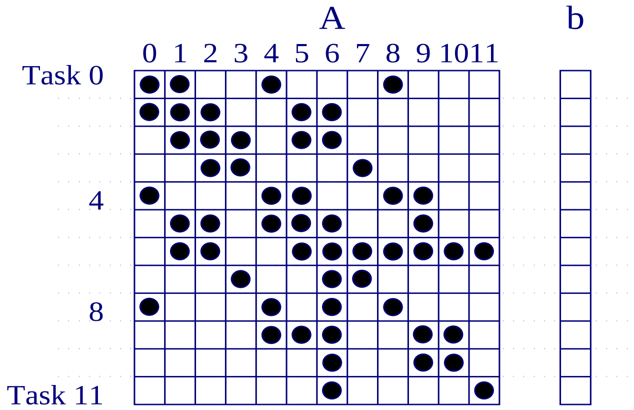
Feladat kölcsönhatási gráf

- A dekompozíció során meghatározott részfeladatok általában adatot cserélnek egymással (pl. a a sűrű mátrix-vektor szorzásnál, ha a vektort nem másoljuk le minden részfeladathoz, a feladatoknak kommunikálnia kell egymással)
- A részfeladatok mint csomópontok és a feladatok kapcsolata/az adatcsere mint élek gráfot határoznak meg: *feladat kölcsönhatási gráf*
- *A feladat kölcsönhatási gráf* adatfüggőséget reprezentál, míg a *feladat függőségi gráf* vezérlési kapcsolatot
- Feladat kölcsönhatási gráf része a feladat kölcsönhatási gráfnak

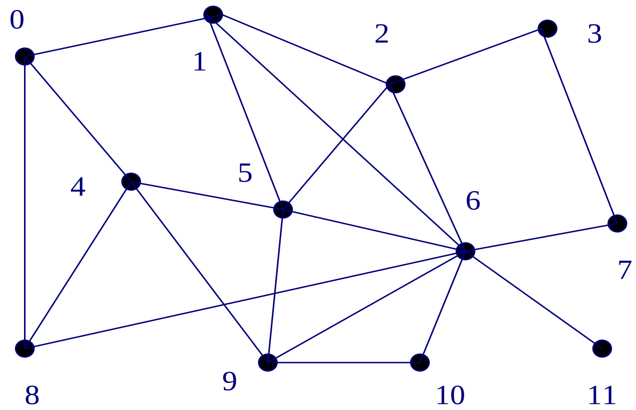
Feladat kölcsönhatási gráf - példa

Az *A* ritka mátrix és a *b* vektor szorzása a feladat

- Mint korábban, az eredményvektor minden eleme függetlenül számolható
- A korábbi sűrű mátrix-vektor szorzattal ellentétben, csak *A* nem nulla elemei vesznek részt a számításban
- Ha memória optimalizálási szempontból a *b* vektort megosztjuk a részfeladatokat között, ekkor a feladat kölcsönhatási gráf azonos *A* mátrix gráfjával (azzal a mátrixszal, amely *A* szomszédsági struktúráját reprezentálja)



(a)



(b)

Dekompozíciós módszerek

Dekompozíciós módszerek

Nincs általános recept, de gyakran a következőket használják:

- Adat dekompozíció
- Rekurzív dekompozíció
- Felderítő dekompozíció
- Spekulatív dekompozíció
- Hibrid dekompozíció

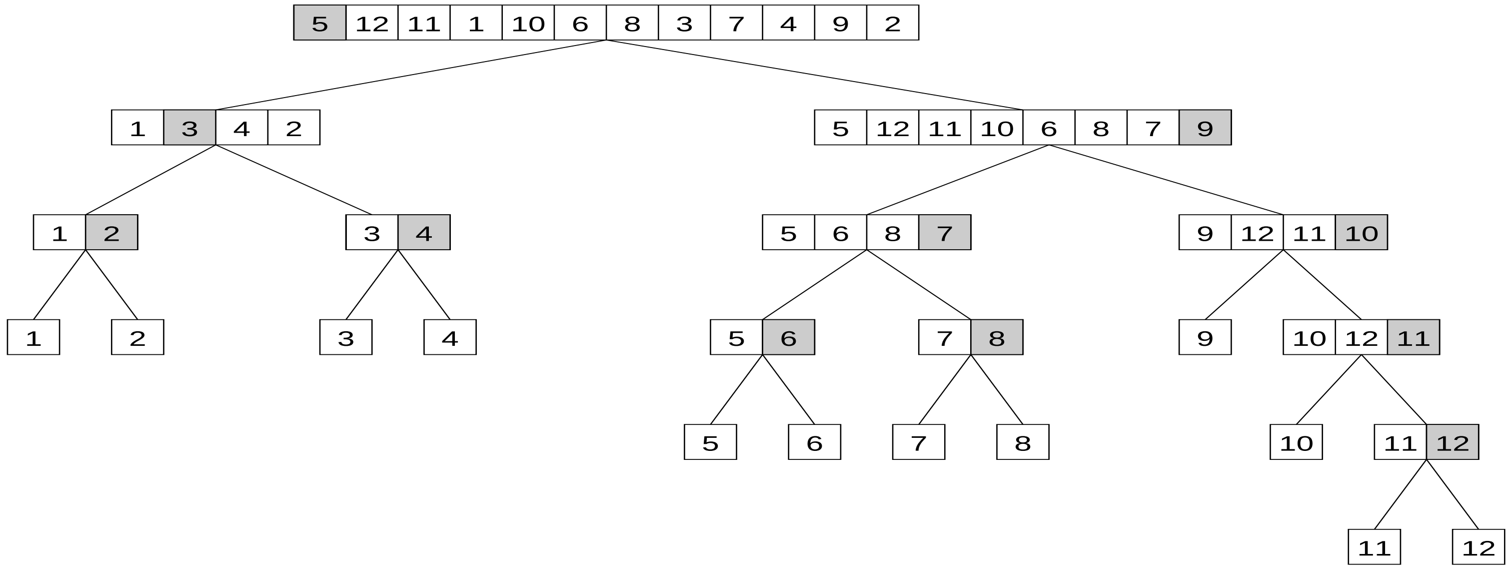
feladat dekompozíciós módszerek

Rekurzív dekompozíció

- Általában minden olyan esetben használható, amikor az „oszd meg és uralkodj” stratégiát alkalmazhatjuk
- Az adott feladatot először részekre bontjuk
- Ezek a részproblémákat rekurzívan tovább bontjuk a kívánt szemcsézettség eléréséig

Rekurzív dekompozíció - példa

- Egy klasszikus példa a Quicksort



- A vezérlőelem segítségével két részre bontjuk a listát, és a részlistákat párhuzamosan dolgozhatjuk fel (részlisták feldolgozása független részfeladat). Rekurzívan végezhető

Rekurzív dekompozíció - példa

- Lista minimális elemének keresése (vagy más asszociatív operáció) esetén is alkalmazható az oszd meg és uralkodj elv
- Kiindulás: soros megvalósítás

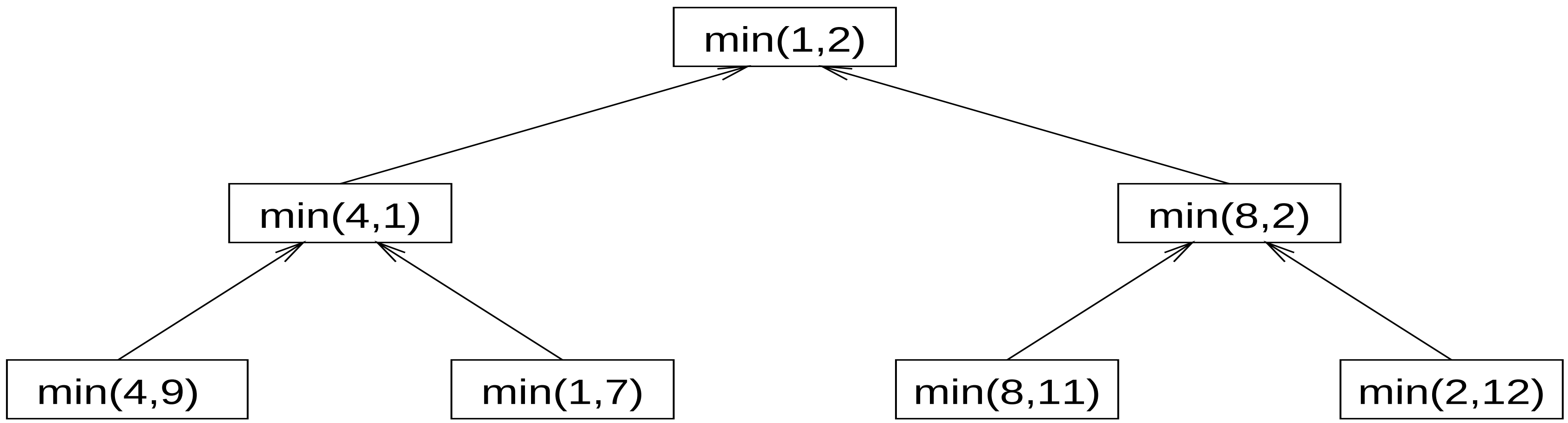
1. **function** SERIAL_MIN (*A, n*)
2. **begin**
3. *min* = A[0];
4. **for** *i* := 1 **to** *n* − 1 **do**
5. **if** (*A*[*i*] < *min*) *min* := *A*[*i*];
6. **endfor**;
7. **return** *min*;
8. **end** SERIAL_MIN

Rekurzív dekompozíció - példa

```
function RECURSIVE_MIN (A, n)
begin
if ( n = 1 ) then
    min := A [0]  ;
else
    lmin := RECURSIVE_MIN ( A, n/2 );
    rmin := RECURSIVE_MIN (  &(A[n/2]), n - n/2 );
    if (lmin  < rmin) then
            min := lmin;
    else
            min := rmin;
    endelse;
endelse;
return min;
end RECURSIVE_MIN
```

Rekurzív dekompozíció - példa

- A feladat függőségi gráfban minden csomópont két szám közül a kisebbet adja vissza. Az eredeti halmaz, amiben a minimumot keressük: {4, 9, 1, 7, 8, 11, 2, 12}



Adat dekompozíció

- Nagy mennyiségű adaton dolgozó problémák esetén használatos
- Az alapelv, a részfeladatokat úgy kapjuk meg, hogy a nagyszámú adatból indulunk ki
- Gyakran két lépésben valósítják meg az adat dekompozíciót:
  - 1: Adatok felosztása
  - 2: Az adat-particionálásból indukált számítási feladat particionálás
- Milyen adat particionálásából induljunk ki?
  - Input/Output/Közbenső
- Az indukált számításokat miként hajtsuk végre?
  - Tulajdonos- számol szabály: amihez rendeljük az adatot, az végzi a számítást

Adat dekompozíció: output adatból

- Ha az output részei egymástól függetlenek
- Az input egyszerű függvénye az output
- A particiók részfeladatokhoz rendelése a probléma természetes megközelítése

Output adat dekompozíció - példa

- Két *n* x *n* mátrix, *A* és *B* szorzatának eredménye *C*. Az eredmény mátrix négy részre (is) osztható

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

$$\begin{aligned}
\text{Task 1: } & C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
\text{Task 2: } & C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
\text{Task 3: } & C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
\text{Task 4: } & C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}$$

(b)

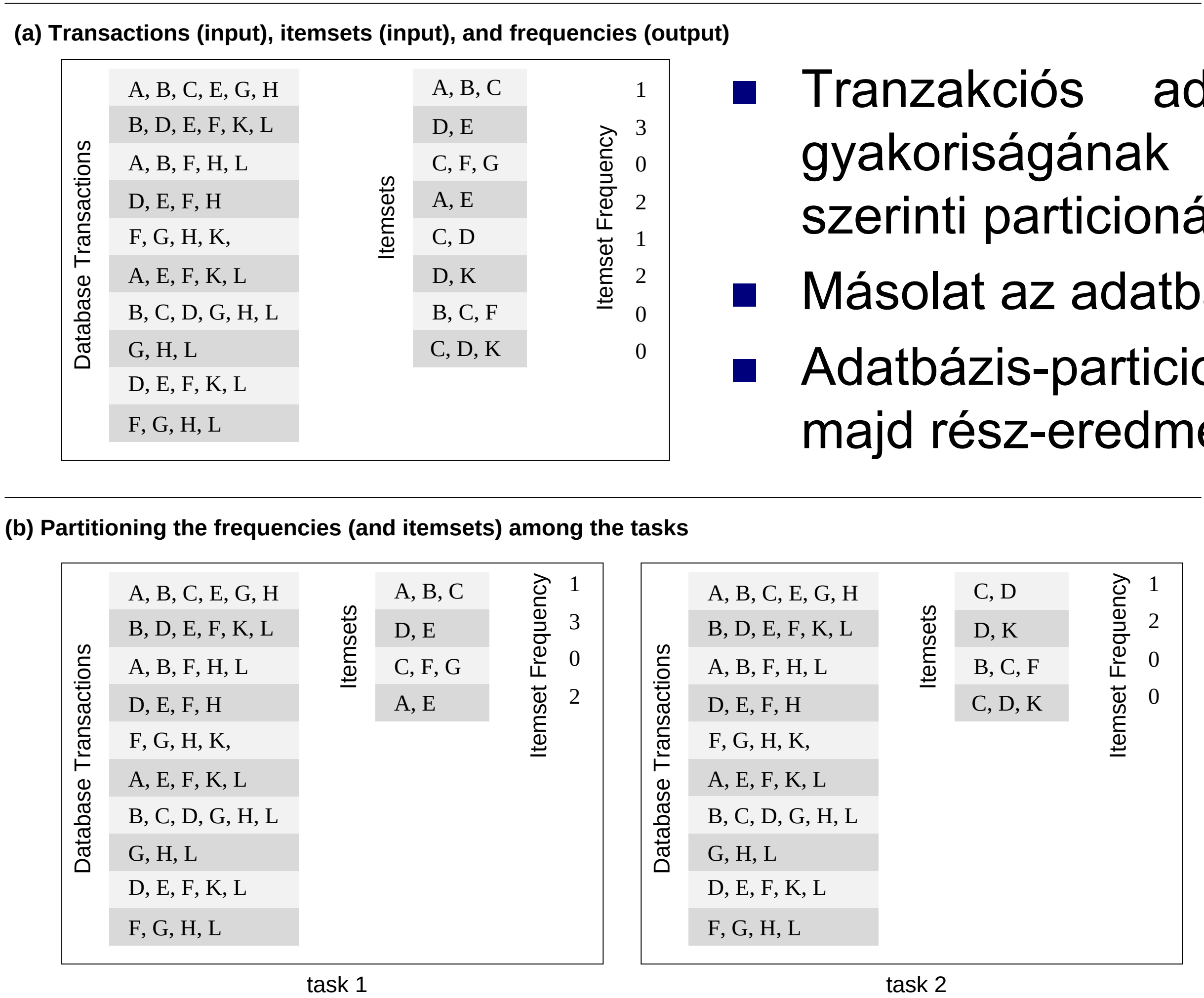
Figure 3.10 (a) Partitioning of input and output matrices into 2×2 submatrices. (b) A decomposition of matrix multiplication into four tasks based on the partitioning of the matrices in (a).

Output adat dekompozíció - példa

- A particionálás nem biztos, hogy egyértelmű részfeladatra bontást eredményez. Az előző feladat két másik megoldása

Decomposition I	**Decomposition II**
Task 1: $C_{1,1} = A_{1,1}B_{1,1}$	Task 1: $C_{1,1} = A_{1,1}B_{1,1}$
Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$	Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$
Task 3: $C_{1,2} = A_{1,1}B_{1,2}$	Task 3: $C_{1,2} = A_{1,2}B_{2,2}$
Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$	Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$
Task 5: $C_{2,1} = A_{2,1}B_{1,1}$	Task 5: $C_{2,1} = A_{2,2}B_{2,1}$
Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$	Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$
Task 7: $C_{2,2} = A_{2,1}B_{1,2}$	Task 7: $C_{2,2} = A_{2,1}B_{1,2}$
Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$	Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

Output adat dekompozíció - példa



- Tranzakciós adatbázisban cikkhalmazok gyakoriságának meghatározása. Output szerinti particionálás
- Másolat az adatbázisról taszkonként?
- Adatbázis-particionálás a taszkok között, majd rész-eredmények összegzése

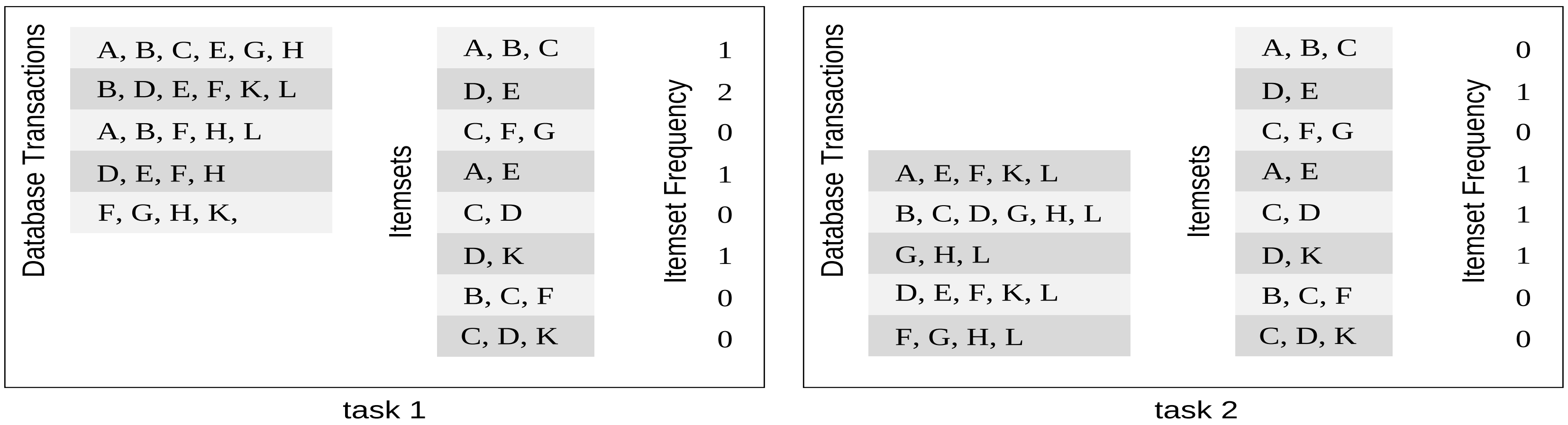
Input adat particionálás

- Alkalmazható, ha minden output az input függvényeként természetesen számítható
- Sok esetben ez a természetes út, mert az output előre nem ismert (pl. rendezés, minimum meghatározás)
- A részfeladat minden inputhoz kapcsolható. Olyan mennyiségű számítást végez el a részfeladat, amennyit csak lehet az adataiból. Rákövetkező feldolgozás kombinálja a részeredményeket

Input adat particionálás - példa

- A tranzakciós adatbázisban az inputot particionáljuk. Ez meghatározza a részfeladatokat, minden taszk részeredményeket számol minden cikkhalmazra. Ezeket összegezzük a második lépésben

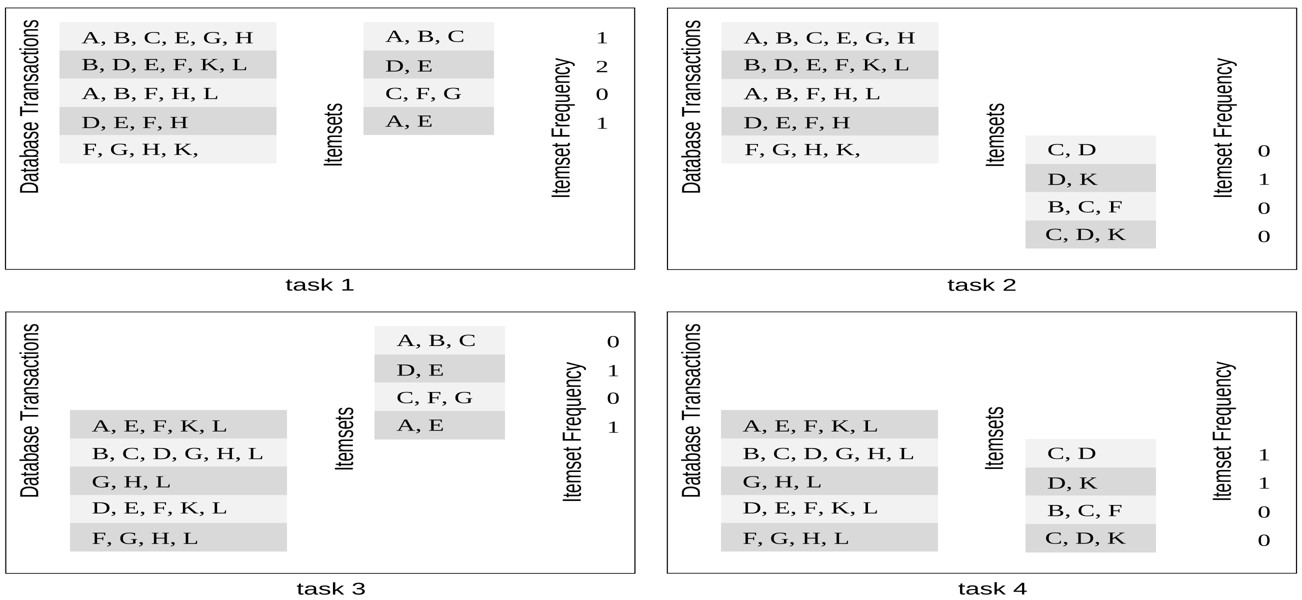
Partitioning the transactions among the tasks



Input és output particionálás - példa

- Gyakran alkalmazható magasabb fokú párhuzamosítás céljából.
- A tranzakciós adatbázist és a cikkhalmazokat is szétosztjuk

Partitioning both transactions and frequencies among the tasks



Közbenső adat particionálása

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2}\\ A_{2,1} & A_{2,2}\end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2}\\ B_{2,1} & B_{2,2}\end{pmatrix} \rightarrow \left(\begin{array}{c} \begin{pmatrix} D_{1,1,1} & D_{1,1,2}\\ D_{1,2,1} & D_{1,2,2}\end{pmatrix} \\ \begin{pmatrix} D_{2,1,1} & D_{2,1,2}\\ D_{2,2,1} & D_{2,2,2}\end{pmatrix}\end{array}\right)$$

Stage II

$$\begin{pmatrix} D_{1,1,1} & D_{1,1,2}\\ D_{1,2,1} & D_{1,2,2}\end{pmatrix} + \begin{pmatrix} D_{2,1,1} & D_{2,1,2}\\ D_{2,2,1} & D_{2,2,2}\end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2}\\ C_{2,1} & C_{2,2}\end{pmatrix}$$

A decomposition induced by a partitioning of D

Task 01: $D_{1,1,1} = A_{1,1}B_{1,1}$
Task 02: $D_{2,1,1} = A_{1,2}B_{2,1}$
Task 03: $D_{1,1,2} = A_{1,1}B_{1,2}$
Task 04: $D_{2,1,2} = A_{1,2}B_{2,2}$
Task 05: $D_{1,2,1} = A_{2,1}B_{1,1}$
Task 06: $D_{2,2,1} = A_{2,2}B_{2,1}$
Task 07: $D_{1,2,2} = A_{2,1}B_{1,2}$
Task 08: $D_{2,2,2} = A_{2,2}B_{2,2}$
Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$
Task 10: $C_{1,2} = D_{1,1,2} + D_{2,1,2}$
Task 11: $C_{2,1} = D_{1,2,1} + D_{2,2,1}$
Task 12: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$

Figure 3.15 A decomposition of matrix multiplication based on partitioning the intermediate three-dimensional matrix.

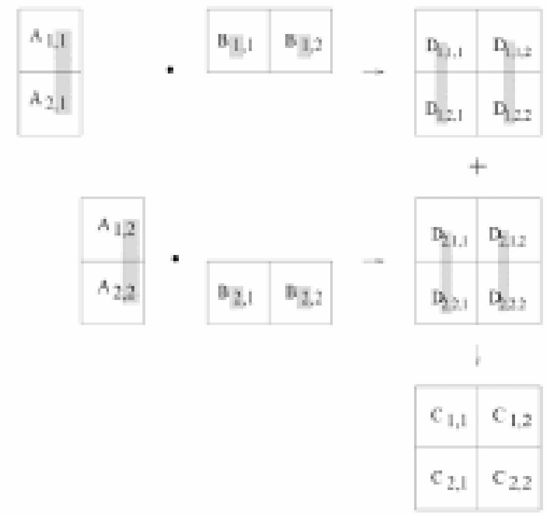


Figure 3.14 Multiplication of matrices A and B with partitioning of the three-dimensional intermediate matrix D.

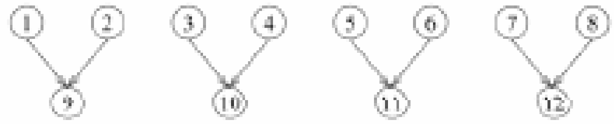


Figure 3.16 The task-dependency graph of the decomposition shown in Figure 3.15.

Felderítő dekompozíció

- Olyan számítások dekompozíciója, ahol a megoldás állapottérben történő keresésekhez kapcsolódik

Példa. Tili-toli

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

(a)

1	2	3	4
5	6	7	8
9	10		11
13	14	15	12

(b)

1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

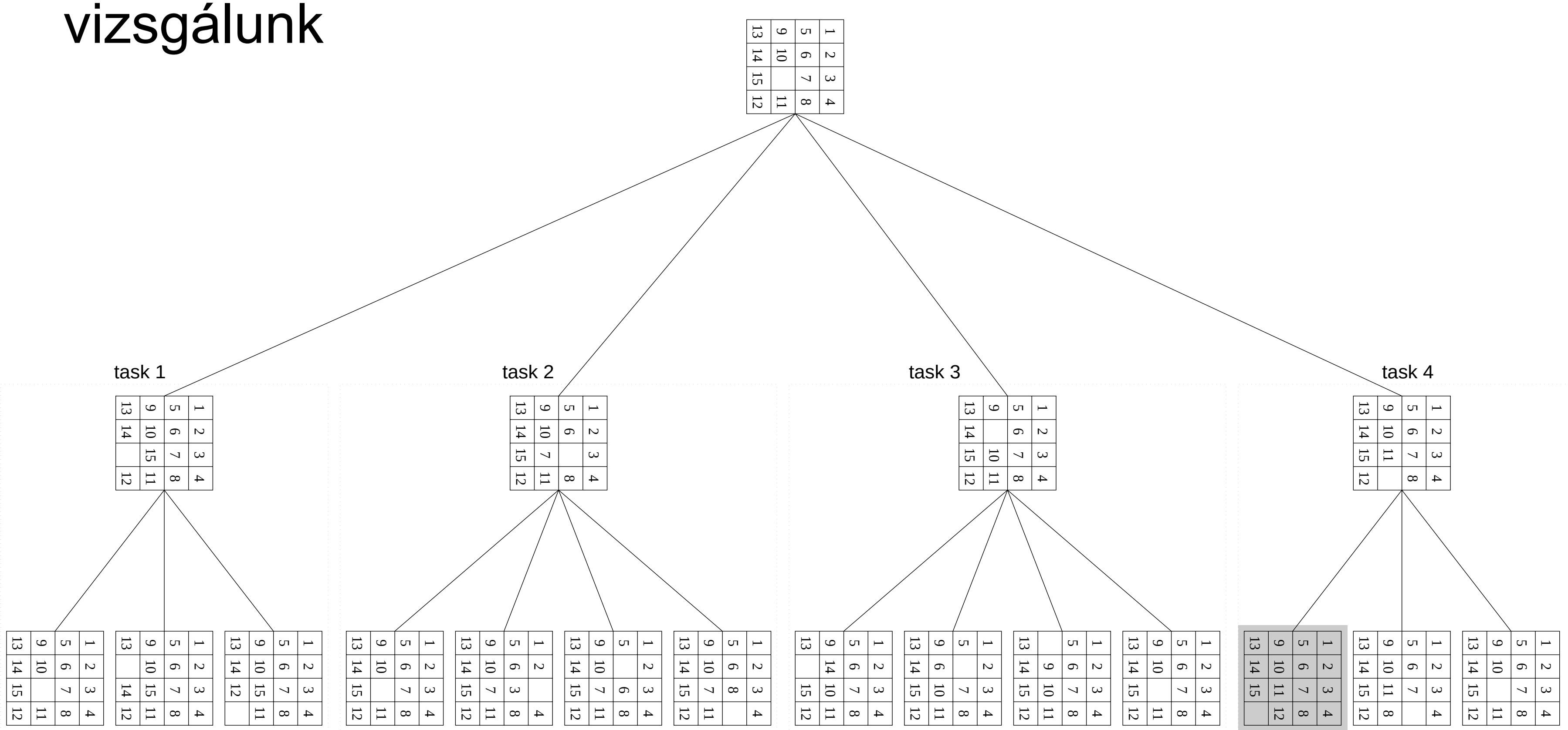
(c)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(d)

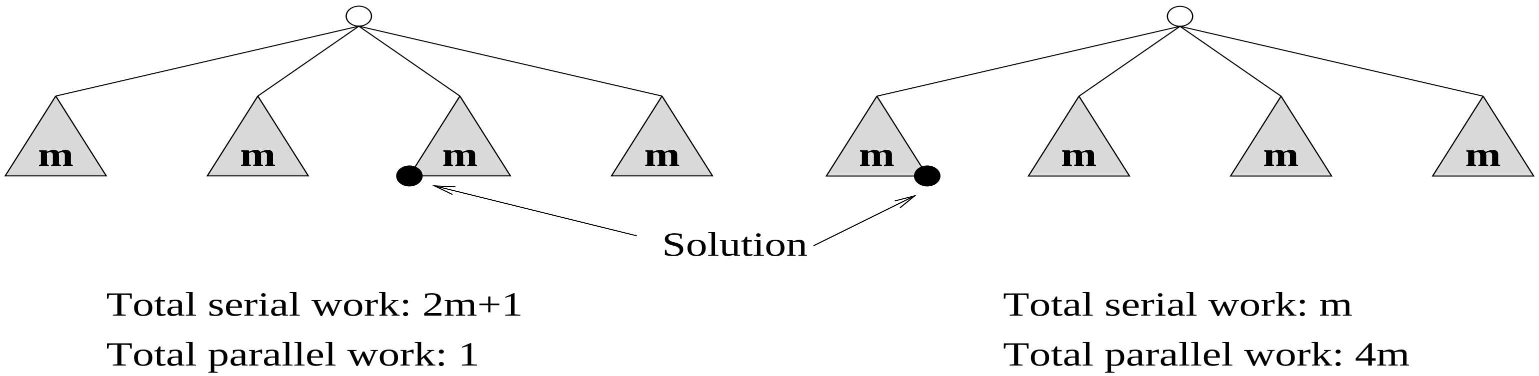
Felderítő dekompozíció - példa

- A *állapottér* felfedezése oly módon, hogy különböző, lehetséges követő lépéseket, mint önálló feladatokat vizsgálunk



Felderítő dekompozíció: Anomáliák a számításban

- A felderítő dekompozíció esetében, a dekompozíciós technika megváltoztathatja a szükséges munkamennyiséget; akár megnövelheti, akár csökkentheti azt
- Nem biztos, hogy mind hasznos munka



(a) (b)

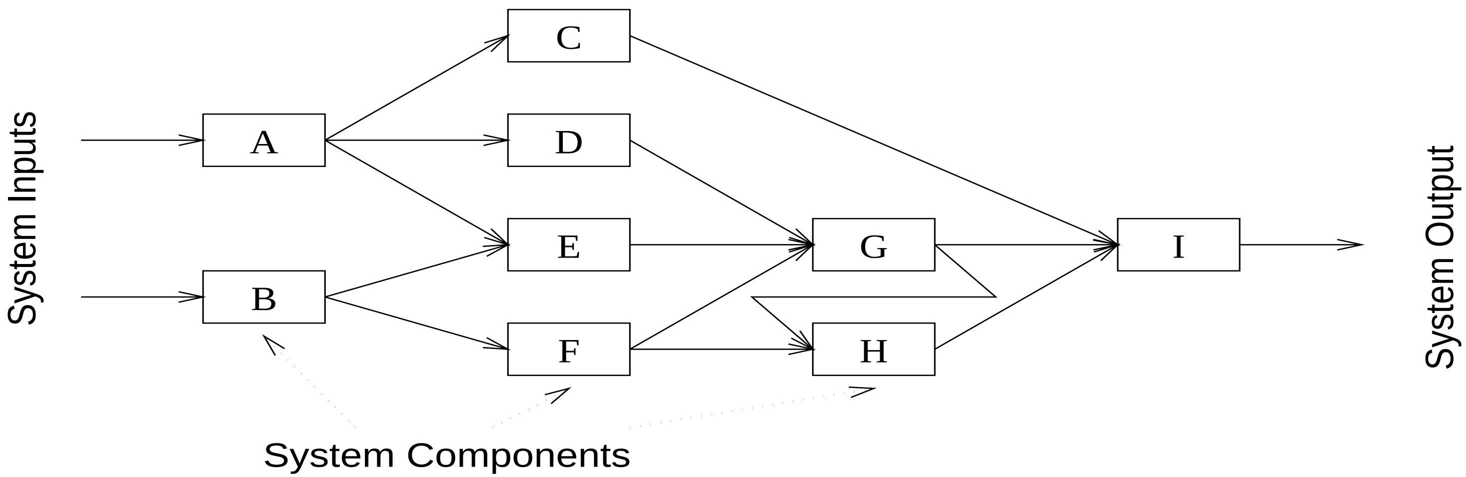
Spekulatív dekompozíció

- Akkor használandó, amikor a következő lépés sok közül az egyik lesz, hogy melyik csak akkor határozható meg, amikor az aktuális részfeladat lefutott
- Feltételezi az aktuális feladat valamilyen kimenetelét és előre futtat néhány rákövetkező lépést
  - Mint a mikroprocesszor szinten a spekulatív futtatás
- Két megközelítés
  - Konzervatív: csak akkor határoz meg feladatokat, ha azok már biztosan függetlenek
  - Optimista: akkor is ütemez feladatot, ha potenciálisan téves lehet
- A konzervatív megközelítés kisebb párhuzamosságot eredményez; az optimista megközelítés hiba esetén roll-back mechanizmust igényel

Spekulatív dekompozíció - példa

Diszkrét események szimulációja

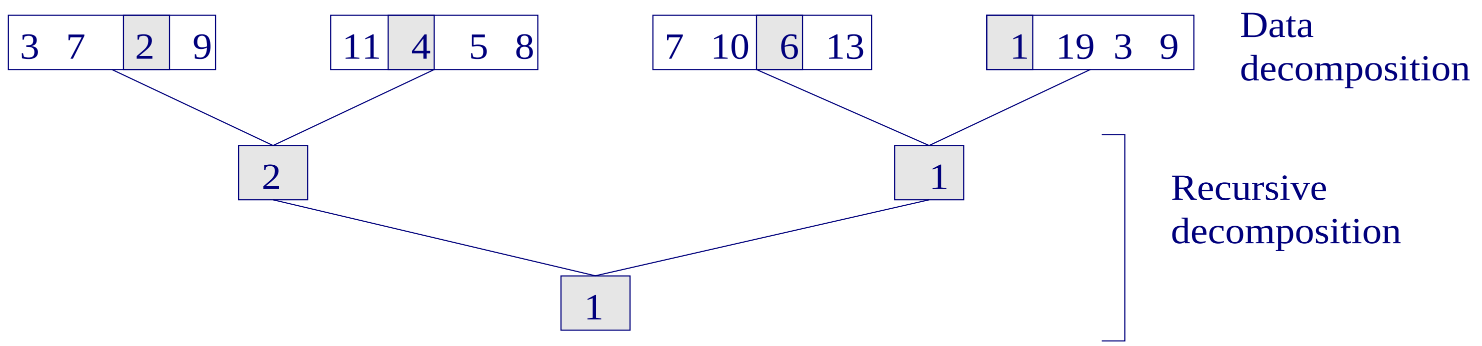
- Idő szerint rendezett eseménylista a központi adatstruktúra
- Az események idő szerinti sorrendben játszódnak, feldolgozásra kerülnek, és ha szükséges, akkor az eredmény események beillesztésre kerülnek az eseménylistába
- Csak a spekuláció révén párhuzamosítható
- Állapot visszaállítási extra feladatot követel (számítás és memória)



Hibrid dekompozíció

Gyakran előnyös a dekompozíciós technikák kombinált használata

- Példa: minimumkeresés



Leképzés

Leképzési technikák

Processzek és leképezés

- Általában a dekompozíció során meghatározott részfeladatok száma meghaladja az elérhető számítási egységek számát

- A részfeladatokat processzekre képezzük le

Megjegyzés: Nem processzorokra történő leképzésről beszélünk, hanem processzekre, mert az API-k tipikusan ezt biztosítják. A részfeladatokat összegyűjtjük (aggregáció) processzekbe és a rendszer fogja a fizikai processzorokra terhelni. A processzt feldolgozó entitásként (összegyűjtött részfeladatok és adatok összessége) használjuk és nem folyamatként.

Processzek és leképezés

- A párhuzamos teljesítmény szempontjából kritikus a részfeladatok megfelelő leképzése processzekre
- A leképzést a feladat függőségi gráf és a feladat kapcsolati gráf határozza meg
- A függőségi gráf használható a processzek közötti munka – bármely időpontban – egyenletes terítésére (minimális várakozás és optimális terhelés kiegyenlítés)
- A kölcsönhatási gráf biztosíthatja, hogy a processzek minimális kapcsolatban legyenek egymással (kommunikáció minimalizálás)

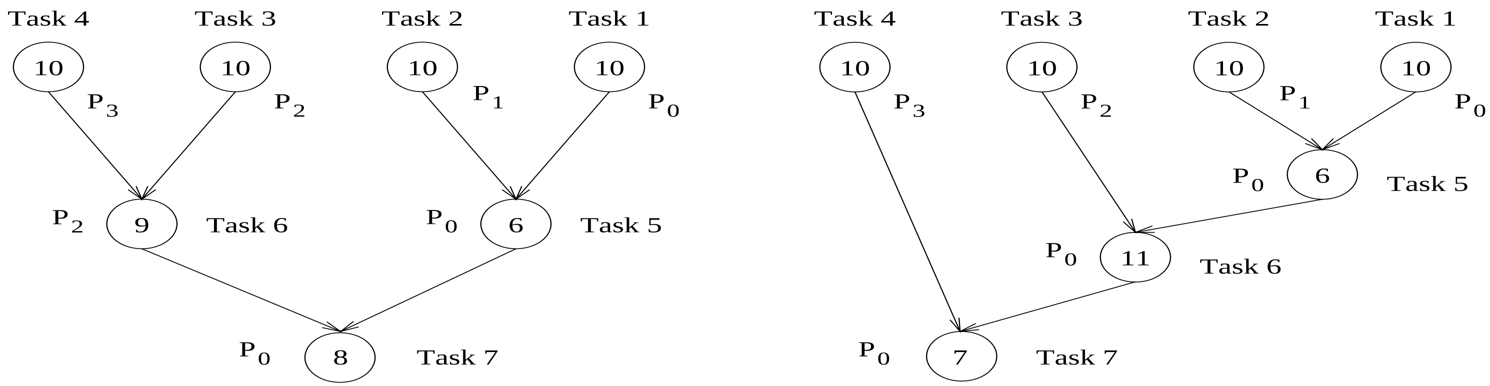
Processzek és leképezés

Cél: minimális párhuzamos futási idő megfelelő leképzéssel:

- Független részfeladatok különböző processzekre leképzése
- A kritikus úton lévő részfeladatok processzekhez rendelése ASAP
- A processzek közötti interaktivitás minimalizálása úgy, hogy a sokat kommunikáló részfeladatokat ugyanahhoz a processzhez rendeljük

Megjegyzés: Egymásnak ellenmondó kritériumok

Processzek és leképezés - példa



(a) (b)

- A függőségi gráf „szintjei” alapján lehet meghatározni a leképzést: egy szinten lévő részfeladatokat különböző processzekhez kell rendelni

A részfeladatok leképzése

- Miért kel körültekintően leképezni a részfeladatokat?
  - Véletlenszerűen hozzárendelhetjük a processzorokhoz?
- Helyes leképzés kritikus lehet, mivel minimalizálni kell a párhuzamos feldolgozás miatti extra ráfordítást
  - Ha T_p a párhuzamos futás ideje *p* processzoron és T_s a sorosé, akkor a teljes extra ráfordítási idő (*total overhead)* $T_o = p*T_p - T_s$
    - A párhuzamos rendszer munkája több, mint a sorosé
  - Az extra ráfordítás forrásai:
    - Terhelés egyenetlenség
    - Processzek közötti kommunikáció (Inter-process communication: IPC)
      - Koordinációi/szinkronizáció/adat megosztás

Miért lehet összetett a leképzés?

A feladat függőségi és a feladat kölcsönhatási gráfot figyelembe kell venni a leképzésnél

- Ismertek a részfeladatok előre?
  - Statikus vs. Dinamikus részfeladat generálás
- Milyenek a számítási követelmények?
  - Egyformák vagy nem egyformák?
  - Ismerjük előre?
- Mennyi adat kapcsolódik az egyes részfeladatokhoz?

Feladat függőségi gráf

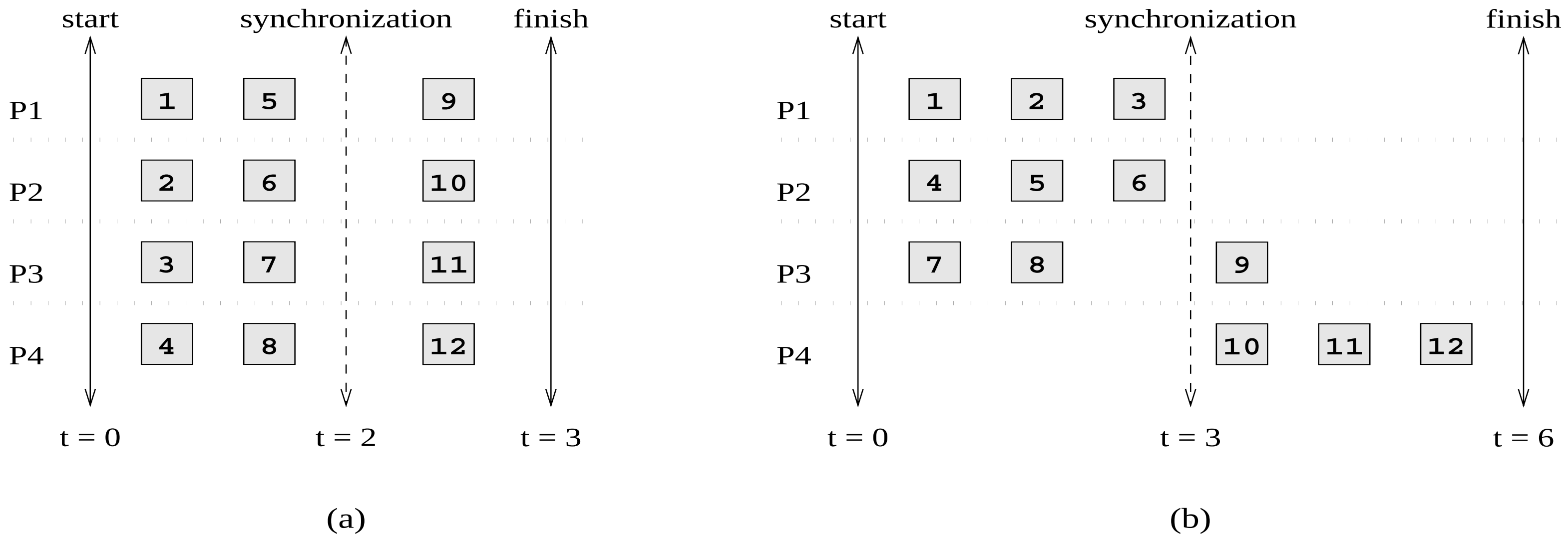
- Milyen a kapcsolat a részfeladatok között?
  - Statikus, vagy dinamikus?
  - Ismerjük előre?
  - Adatfüggőek?
  - Szabályosak, vagy szabálytalanok?
  - Csak írnak, vagy írnak és olvasnak?

Feladat kölcsönhatási gráf

- A fenti jellemzőktől függően különböző leképzési technikák szükségesek és ezek eltérő komplexitásúak és költségűek

Leképzési esetek terhelés kiegyenlítés céllal - példa

- 1-8 után szinkronizáció szükséges
- Azonos terhelés, de nem azonos várakozás



Terhelés kiegyenlítéses technikák

Leképzési technikák lehetnek statikusak, vagy dinamikusak

- Statikus leképzés: a részfeladatok processzekre történő leképzésre előre meghatározott.
  - Ehhez pontos ismeret szükséges minden részfeladat méretéről. De ekkor is NP teljes feladat a leképzés
- Dinamikus leképzés: Futási időben történik a részfeladatok processzekhez rendelése.
  - Futási időben generálódó részfeladatok esetén
  - Előre nem ismert számítási igény

Statikus leképzési technikák

- Adat particionáláson alapuló módszerek
- Gráf particionáláson alapuló leképzések
- Hibrid módszerek

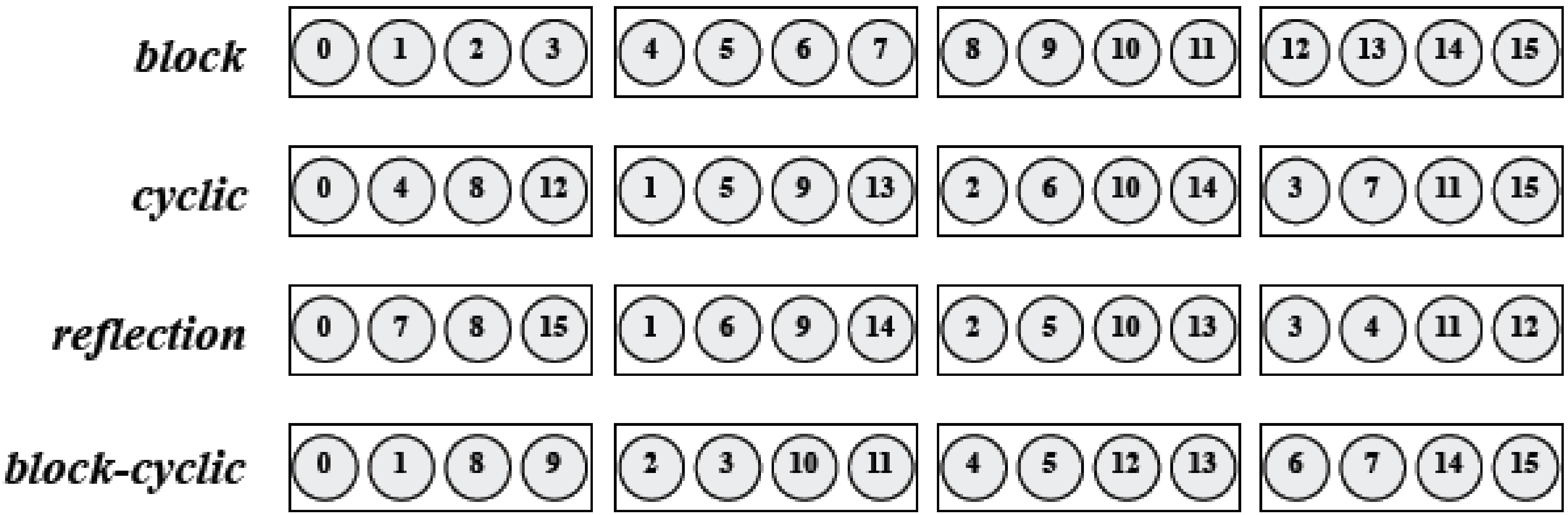
Statikus leképzés – tömb szétosztás

A feladatok és processzek növekvően címkézettek

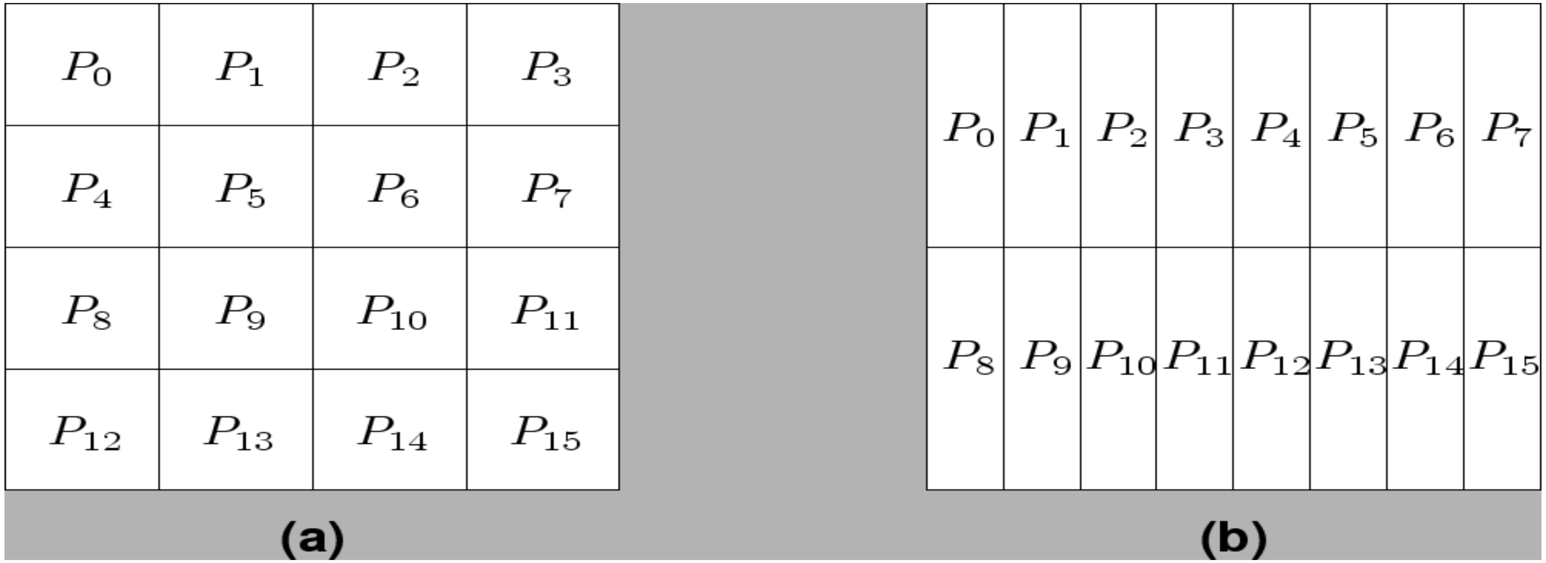
- *Blokk leképzés*: n/p darab egymás utáni részfeladat képződik le az egymás utáni processzekre
- *Ciklikus leképezés*: az *i feladat* az (*i* mod *p)* processzre kerül
- *Tükrözött leképzés*: mint a ciklikus, de a részfeladatok fordított sorrendben kerülnek ki
- *Blokk-ciklikus és blokk-tükrözött leképzés*: a részfeladatok blokkjai kerülnek hozzárendelésre

Magasabb dimenzióban is alkalmazható módszerek: külön minden dimenzióban

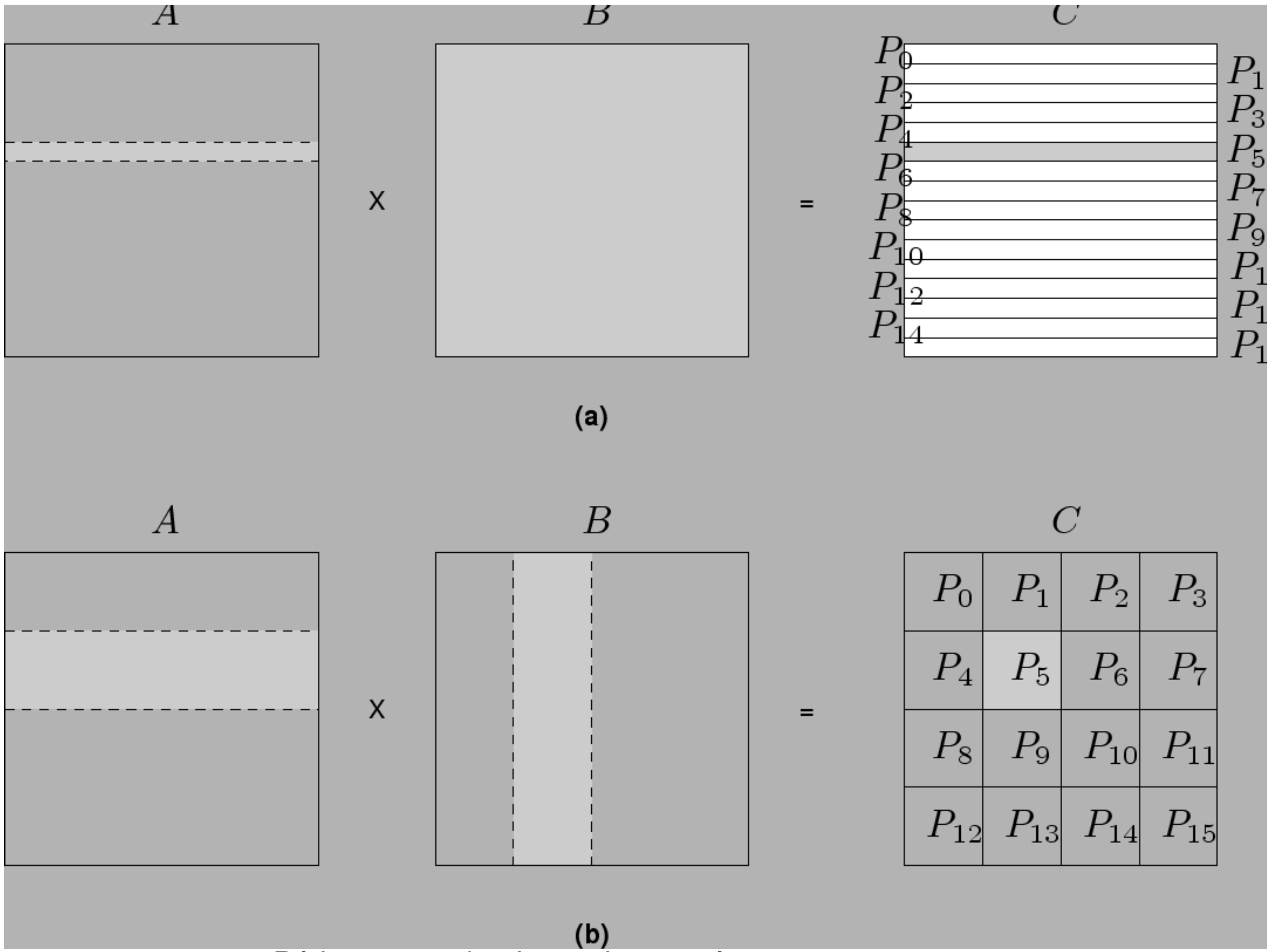
Statikus leképzés – tömb szétosztás



Tömbszétosztás - példa



Sűrű mátrix - példa



Ciklikus és blokk-ciklikus szétosztás

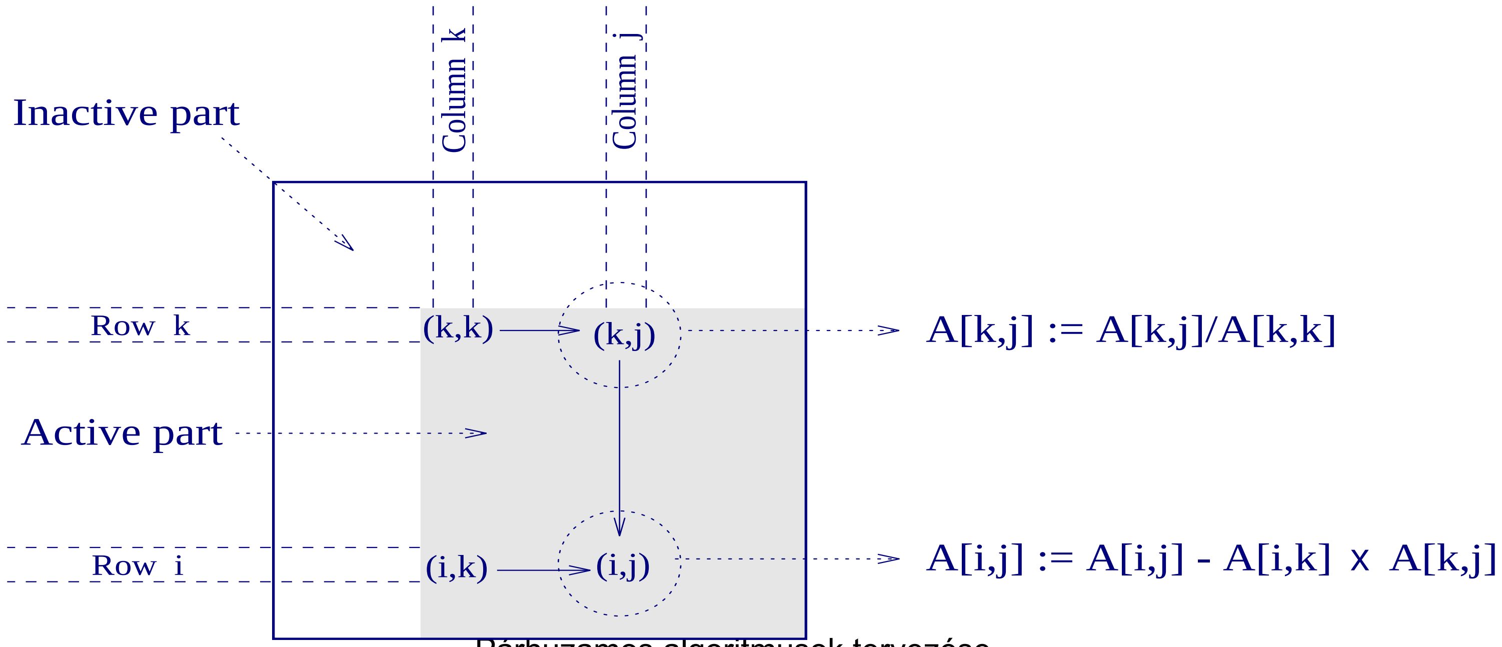
- Ha az adatelemekhez kapcsolódó számítási igény változik, blokk leképzéses módszer terhelés egyenetlenséghez vezet
- Példa sűrű mátrix Gauss elimináció

Blokk-ciklikus szétosztás

- Variation of the block distribution scheme that can be used to alleviate the load-imbalance and idling problems.
- Partition an array into many more blocks than the number of available processes.
- Blocks are assigned to processes in a round-robin manner so that each process gets several non-adjacent blocks.

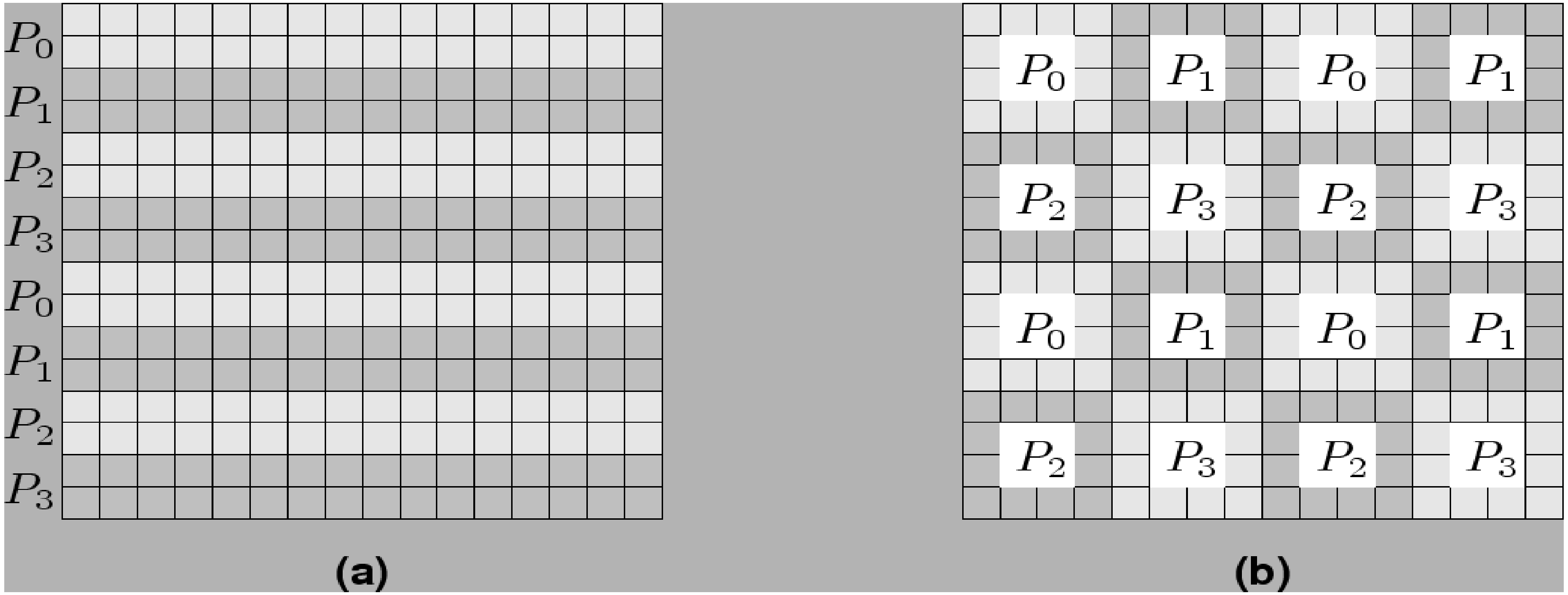
Blokk-ciklikus leképzés – Gauss elimináció - példa

Minden processz a mátrix különböző részeiből is kap részfeladatokat



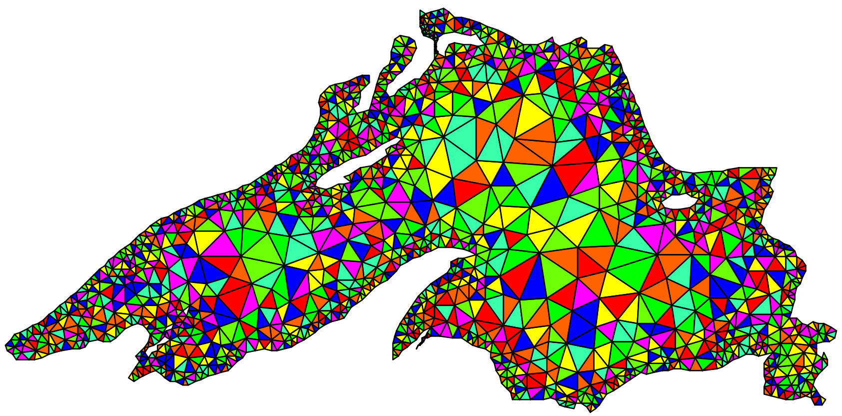
Blokk-ciklikus leképzés – Gauss elimináció - példda

- A blokk méret *n/p*, ahol *n* a mátrix mérete és *p* a processzek száma

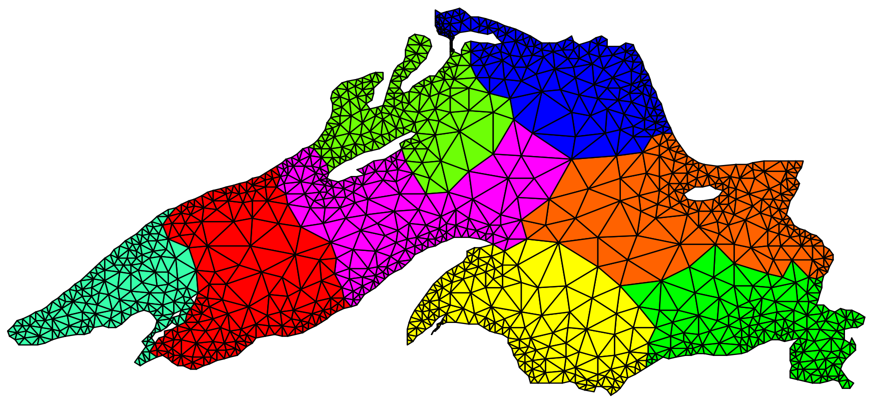


Gráf particionáláson alapuló leképzés

- A feladat kapcsolati gráf particionálásával
  - Azonos mennyiségű feladat
  - Minimális számú élvágás
  - NP-teljes probléma, heurisztika



Random Partitioning



Partitioning for minimum edge-cut.

Dinamikus terhelés kiegyenlítés

- Dinamikus leképzés esetén az elsődleges cél a terhelés kiegyenlítés
- Centralizált sémák
  - Egy processz felelős a feladatok kiadásáért
    - mester-szolga paradigma
  - Kérdés
    - Részfeladatok szemcsézettsége
- Szétosztott sémák
  - A munka bármely processz-pár között szétosztható (küldő-fogadó szerep)
  - Kérdés
    - Hogyan párosíthatók a processzek?
    - Ki inicializálja a munka kiosztást?
    - Mennyi munka kerül kiosztásra?