HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.05.13.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

Párhuzamos feldolgozás

Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék

ghorvath@hit.bme.hu, belso@hit.bme.hu

HATÉKONY UTASÍTÁSFELDOLGOZÁS

- Hogyan tettük eddig hatékonnyá az utasítás-végrehajtást?
  - ***Utasítás szintű párhuzamosítással***
    - **Egyszerű pipeline:** átlapolt utasítás végrehajtás
    - **Szuperskalár:** a CPU több, általa párhuzamosíthatónak talált utasítást hajt végre egyszerre
    - **VLIW/EPIC:** a CPU több, a fordító / programozó által párhuzamosíthatónak talált utasítást hajt végre egyszerre

- Korlátok:
  - Minél több utasítás áll feldolgozás alatt
    → annál nagyobb a sansz az egymásrahatásra
  - Minél több a műveleti egység
    → sok forwarding út kell
  - Ciklusidő nem lehet tetszőlegesen rövid
  - Időnként becslés / spekuláció szükséges
    → ha rosszul sikerül, kárba ment egy csomó idő

HATÉKONY UTASÍTÁSFELDOLGOZÁS

- Miért szeretjük mégis az utasításszintű párhuzamosítást?
- Mert egyszerű programozni!
  - A programozó
    - Fejében folyamatábra
    - Szekvenciális algoritmust ír
    - Skalár változókon
  - A párhuzamosítási lehetőségeket a CPU és/vagy a fordító fedi fel
    - Módja: függőségi analízis, adatáramlásos elv
    - El van rejtve a programozó elől
- **Alternatíva:**
  - Explicite mondja meg a programozó, hogy a programjának mely részeit lehet párhuzamosan végrehajtani!
  - Vezérlőtoken többszörözés / összevárás (FORK/JOIN)

PÁRHUZAMOSSÁG FORMÁI

- Explicit párhuzamosság hardver támogatása:
  - Több végrehajtási szálat kezelő processzorok
    - Van több program counter, de ezek mégsem valódi multiprocesszoros rendszerek!
  - Valódi multiprocesszoros rendszerek

TÖBB SZÁLAT KEZELŐ PROCESSZOROK

- Mi az utasítás-pipeline rákfenéje?
  - Cache hiba
  - TLB hiba
  - Adatfüggőségek
- Ilyenkor mi történik?
  - Semmi

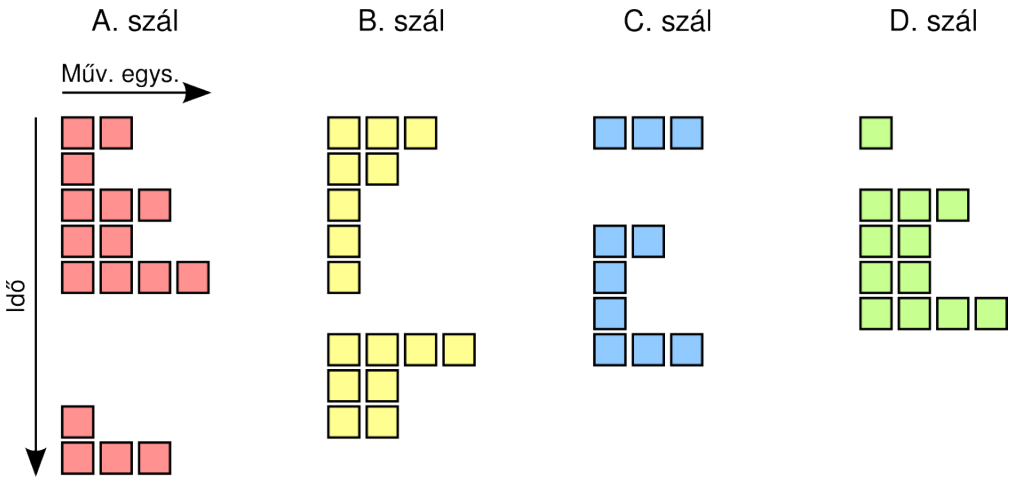
    → Szünet történik
- Megoldás:
  - Kezeljünk több utasításszámlálót!
  - Ha fennakadás van, a megoldásig tegyük félre az utasítássorozatot, és vegyük elő a másikat

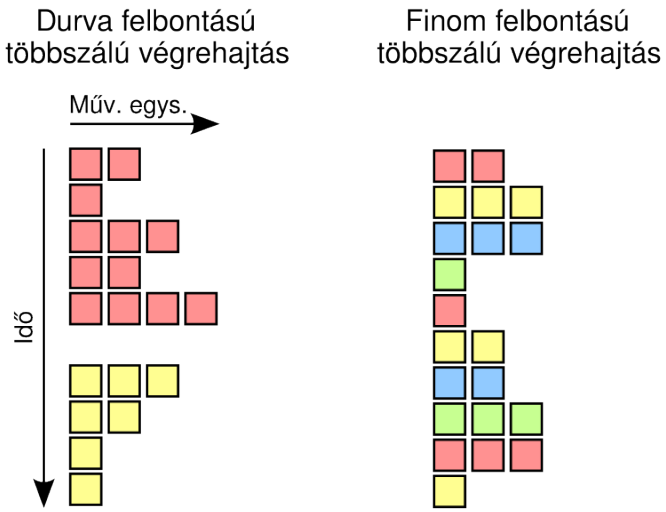
    → Sokkal kevesebb szünet
  - Ezt csinálják a **több szálat kezelő processzorok**

IDŐOSZTÁSOS TÖBBSZÁLÚ VÉGREHAJTÁS

- A többszálú végrehajtást támogató processzorok egy része
  - ***Finom felbontású multi-threading***-et támogat:
    - Minden órajelben szálat (utasításszámlálót) vált
    - Mire egy szálra újra sor kerül, jó eséllyel megoldódik a problémája (nem kell külön szünet)
  - ***Durva felbontású multi-threading***-et támogat:
    - Csak akkor vált szálat a processzor, ha az aktuális megakad
- Hardveres megvalósítás:
  - Szálváltási idő: 0 vagy 1 órajelciklus (finom, ill. durva esetben)
  - Utasítások belépéskor egy szálazonosítót kapnak
  - A processzorban több utasításszámláló és regiszter-tároló van
  - TLB, cache, elágazásbecslő adatszerkezetek, stb. osztottak
    - Bejegyzésekben új mező: melyik szálhoz tartozik
  - Minden utasítás a szálazonosítójának megfelelő regiszterkészlettel, TLB és cache bejegyzéssel, stb. dolgozik

IDŐOSZTÁSOS TÖBBSZÁLÚ VÉGREHAJTÁS

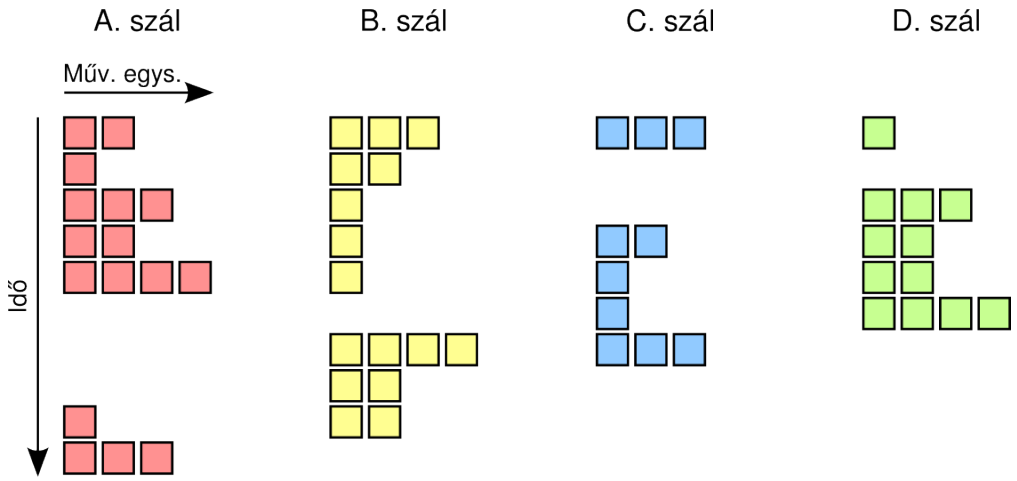




SZIMULTÁN TÖBBSZÁLÚ VÉGREHAJTÁS

- Az időosztásos alternatívája
- Csak szuperskalár architektúrával működik
- Szuperskalár esetben a sok műveleti egység gyakran kihasználatlan
  - Mert nincs elég párhuzamosítható utasítás a programban
- A nem használt műveleti egységeken hajtsuk végre egy másik szál utasításait!

SZIMULTÁN TÖBBSZÁLÚ VÉGREHAJTÁS



Szimultán
többszálú végrehajtás

TÖBB SZÁLAT KEZELŐ PROCESSZOROK

- Minden formáját könnyű hardveresen megvalósítani

Processzor	Megjelenés éve	Többszálúság formája	Támogatott szálak száma
Intel Pentium 4	2002	Szimultán	2
Intel Itanium 2	2006	Durva felbontású	2
IBM POWER5	2004	Szimultán	2
IBM POWER6	2010	Szimultán	4
IBM POWER8	2013	Szimultán	8
UltraSPARC T1	2005	Finom felbontású	4
UltraSPARC T2	2007	Finom felbontású	8
AMD Ryzen	2017	Szimultán	2

© Hálózati Rendszerek és Szolgáltatások Tanszék

VALÓDI MULTIPROCESSZOROS RENDSZEREK

- Teljes értékű processzorokból álló rendszerek
- Miért is jó ez nekünk?
  - Az utasításszintű párhuzamosság lehetőségei korlátozottak
    Explicit megközelítéssel nagyobb léptékű párhuzamosítás érhető el.
  - Költséghatékonyság: a processzorok ára exponenciálisan nő a számítási teljesítménnyel
    → Olcsóbb lassúból többet venni, mint gyorsból egyet
  - Egyszerű bővíthetőség:
    → Nagyobb számítási teljesítményhez csak hozzá kell adni még pár processzort
  - Hibatűrés:
    → Ha egy elromlik, a többi átveheti a feladatait

KORLÁTOK

- A multiprocesszoros rendszerek világában minden szép és jó?
- Nem. A bajok forrása: ***a szoftver***
- Ki párhuzamosítja a programokat?
  - ***A fordító***:
    - A programozó szekvenciális programot ír
    - A fordító felfedezi benne a párhuzamosítható részeket
    - Az elterjedt programozási nyelvekhez ilyen intelligens fordító nincs (egyelőre)
  - ***A programozó***:
    - Nehéz
    - Rengeteg a potenciális hibalehetőség
    - Ez a ma uralkodó szemlélet

A TELJESÍTMÉNYNÖVEKMÉNY KORLÁTAI

- Avagy: Ha N processzort veszek a gépembe N-szer gyorsabban fut a programom?

- Nem. Csak ha a probléma teljes mértékben párhuzamosítható.
- Általános esetben az Amdahl törvény adja meg a teljesítménynövekményt

AMDAHL TÖRVÉNYE

- Legyen a programunk
  - P része tetszőlegesen párhuzamosítható
  - 1-P része szekvenciálisan végrehajtandó
- Legyen a futási idő 1 processzoros rendszerben: *1*
- Kérdés: mennyi a futási idő *N* processzor esetén?
  - Ha az egész szekvenciális lenne: 1
  - Ha az egész párhuzamosítható lenne: 1/N
  - Ha P része párhuzamosítható: (1-P)*1 + P/N
- ***Amdahl törvénye: teljesítménynövekmény az 1 processzoros rendszerhez képest:***

$$S_P(N) = \frac{1}{(1-P) + P/N}$$

AMDAHL TÖRVÉNYE

- Példa: van 100 processzorunk. 80-szoros gyorsulást szeretnénk. A program mekkora része lehet nem párhuzamosítható?

$$P=\frac{S_P(N)-1}{S_P(N)}\frac{N}{N-1}\simeq 0.9975$$

- Mindössze 0.25%-a lehet szekvenciális!
- Ha a kód 5%-a szekvenciális (*P=0.95*):

$$S_P(100)=\frac{1}{0.05+0.0095}=16.8$$

- Mindössze 16.8-szoros a sebesség! 100 processzorral!

AMDAHL TÖRVÉNYE

- Nézzük meg a formulát még egyszer:

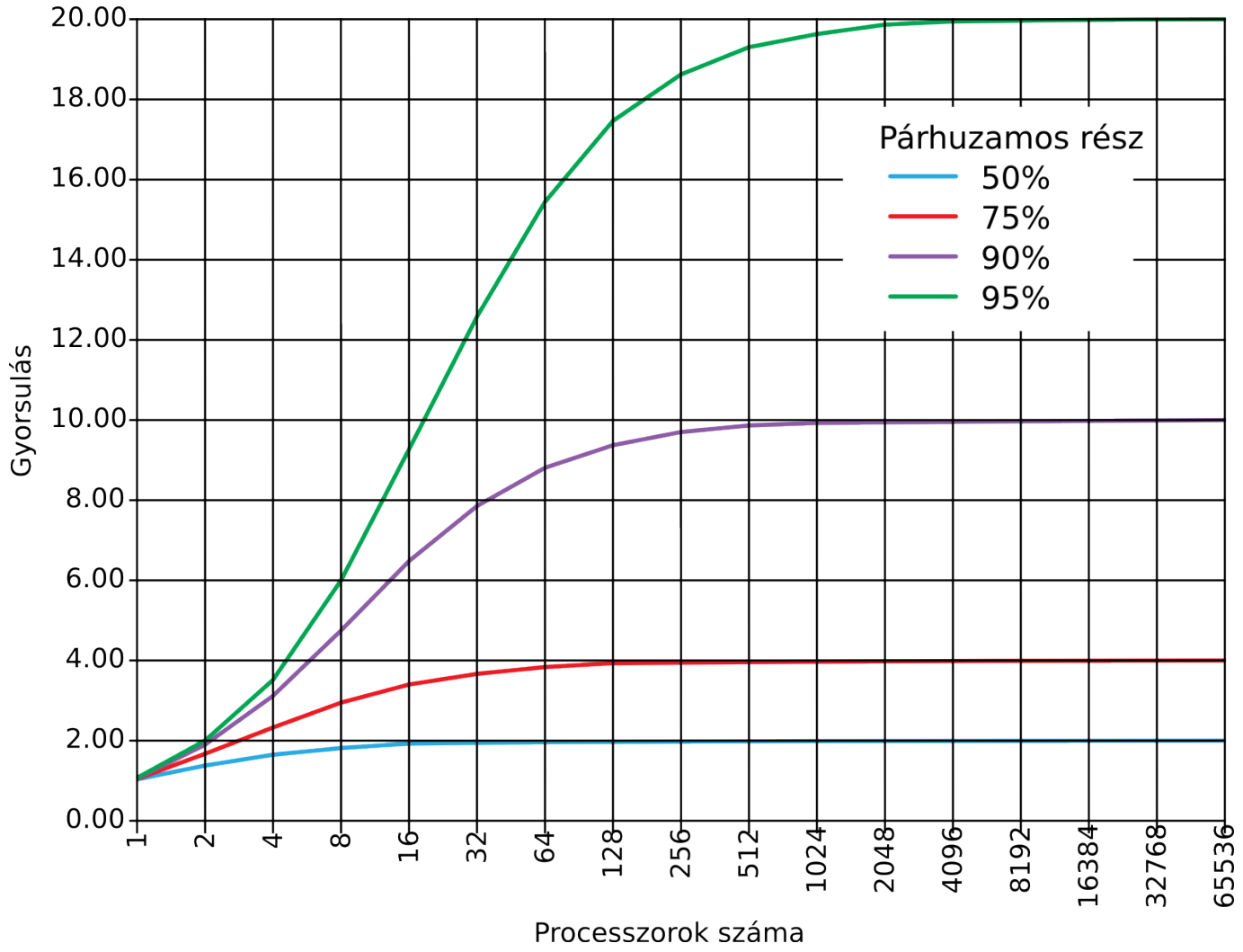
$$S_P(N) = \frac{1}{(1-P) + P/N}$$

- Végtelen sok processzorral:

$$S_P(\infty) = \lim_{N \to \infty} S_P(N) = \frac{1}{1-P}$$

- Akárhány processzorom is van, ennél nagyobb gyorsulás nem érhető el
  - A szekvenciális rész futási ideje korlátozza

AMDAHL TÖRVÉNYE



Multiprocesszoros rendszerek osztályozása

OSZTÁLYOZÁS

- A különböző processzorokon futó taszkok közötti ***kommunikáció szerint***:
  - Osztott memórián alapuló
    - Kommunikáció: egyik taszk beírja, másik kiolvassa
  - Üzenetküldésen alapuló
    - Kommunikáció: a taszkok üzenetet írnak egymásnak, és azt küldözgetik
- A ***memóriaműveletek költsége*** (futási ideje) ***szerint***:
  - UMA (Uniform Memory Access)
    - Minden memóriaművelet azonos ideig tart
    - Mindegy, hogy egy adat hol van: minden processzornak ugyanannyi ideig tart hozzányúlni
  - NUMA (Non-Uniform Memory Access)
    - Fontos, hogy egy adat hol van
    - Célszerű egy processzor által gyakran használt dolgokat hozzá közel elhelyezni

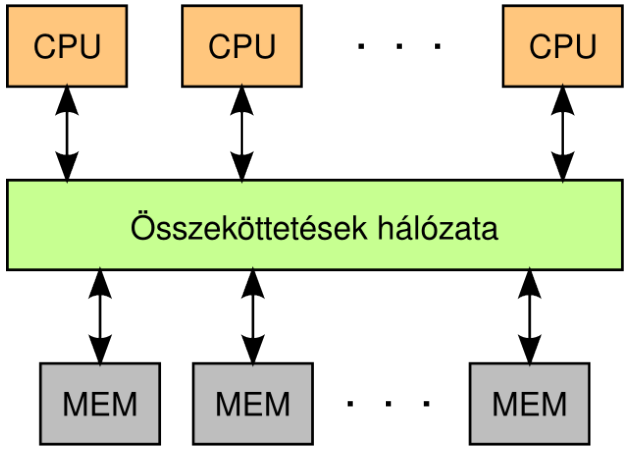
MULTIPROC. ÜZENETKÜLDÉSSEL

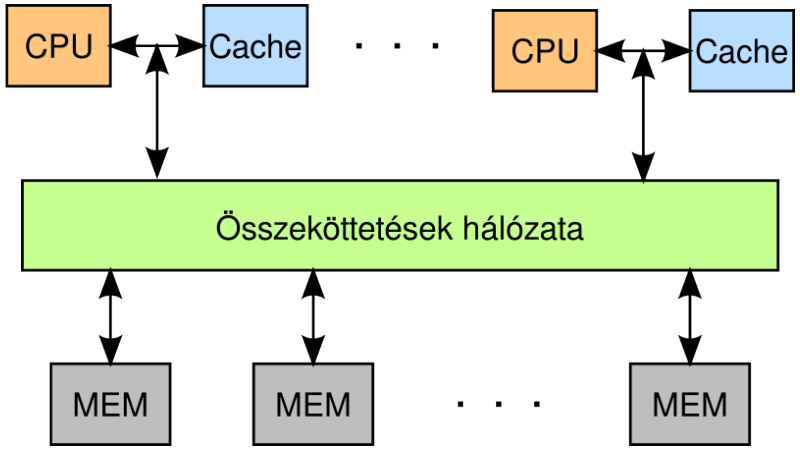
- Csomópontok: teljes értékű számítógépek, külön címtartománnyal
- Kommunikáció: üzenetekkel
  - Explicit hívások:
    - Üzenet küldése (send)
    - Üzenet fogadása (bevárása, receive)
  - Minden taszknak saját egyedi azonosítója van
  - Üzenet tartalma:
    - Rakomány (payload)
    - Küldő azonosítója
    - Címzett azonosítója
  - Összeköttetés hálózat kézbesíti
- Üzenetküldés módjai:
  - Szinkron: A felek bevárják egymást
  - Aszinkron: A küldő nem várja meg a csomag kézbesítését, fut tovább
- Open MPI, PVM

MULTIPROC. OSZTOTT MEMÓRIÁVAL

- A taszkok kommunikációja nagyon egyszerű

UMA eset:



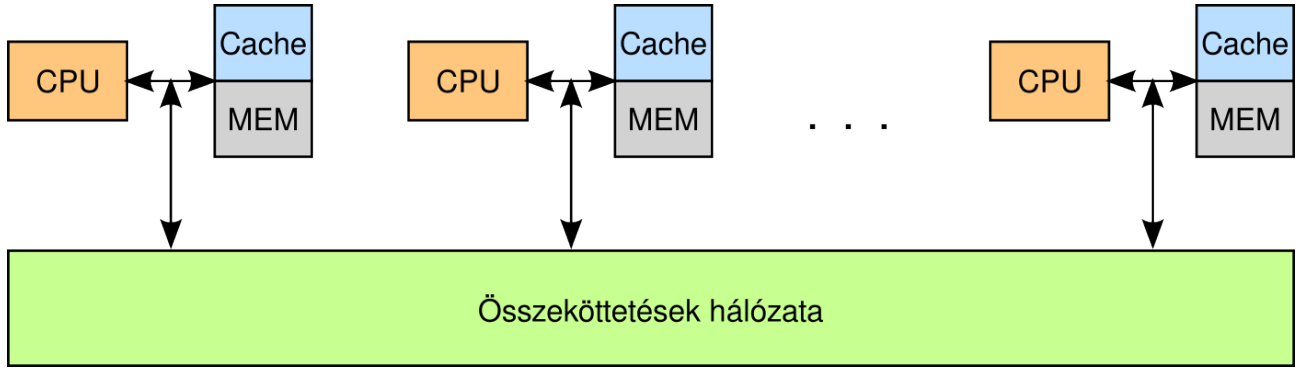


- Egyszerű megvalósítás
- Rossz skálázhatóság:
  - +1 új CPU berakásakor annak teljes memóriaforgalma terheli
- Max. 100 processzorig
- OpenMP

MULTIPROC. OSZTOTT MEMÓRIÁVAL

NUMA eset:

- A rendszer csomópontjai: CPU, Cache, Memória egyben



- Címtér közös
  - Mindenki gyorsan hozzáfér a saját címtér darabkájához
  - Ha máséhoz akar hozzáférni, az a hálózaton valósul meg
  - Hálózati terhelés: csak a taszkok közötti kommunikáció esetén → skálázható!

OSZTOTT MEMÓRIAKEZELÉS

- Koherencia probléma:

Lépés	Esemény	P1 cache	P2 cache	Memória
1				x
2	P1 olvassa	x		x
3	P2 olvassa	x	x	x
4	P1 módosítja	**x'**	**x**	**x'**

- **A memória koherens, ha a változásokról előbb-utóbb mindenki értesül**
- Speciális cache kezelés kell → **cache koherencia protokoll**
- Legegyszerűbb megoldás:
  - Write-through + write-no-allocate
    + változás esetén „invalidate” üzenet a többieknek!

OSZTOTT MEMÓRIAKEZELÉS

- Memória konzisztencia probléma

P1	P2	P3
(1) X1=1;	(3) X2=1;	(5) X3=1;
(2) Print X2,X3;	(4) Print X1,X3;	(6) Print X1,X2;

- Várt kimenet:
  - **001011** ((1),(2),(3),(4),(5),(6))
  - **101111** ((1),(3),(4),(5),(2),(6))
  - **111111** ((1),(3),(5),(6),(4),(2))
  - ...stb. → **szekvenciális konzisztencia**
- Előforduló kimenet:
  - 000000 ((2),(4),(6),(1),(3),(5))
  - Ha minden proc. soron kívüli végrehajtást csinál
  - Minden processzor önmagában szemantikailag korrekt (precedenciagráf tiszteletben tartva)

    → De az egész rendszer összességében nem!!!
  - Szoftverből kell megoldani

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

MŰEGYETEM 1782