Párhuzamos programozási modellek

- Osztályozás
  - Párhuzamos rendszerek Flynn-féle osztályozása
  - Párhuzamos rendszerek modern osztályozása
- Elméleti (idealizált) modellek áttekintése
  - A PRAM modell
  - Az adatfolyam-modell
  - A feladat/csatorna modell
- Gyakorlati (hardverben is megvalósított) modellek áttekintése
  - Végrehajtási modellek
  - Memóriamodellek
- Kommunikáció
  - Kommunikációs modellek és hálózati topológiák
  - A kommunikáció időbeli modellezése

Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

Hallgatói tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük *szükséges, de nem elégséges* feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

Párhuzamos rendszerek osztályozása

- **Az osztályozás elősegíti a rendkívül sokféle megoldás áttekintését**

  Lényegében minden osztályozás egységes rendszert kínál a különböző megoldások jellemzőinek tárgyalásához.

- **A párhuzamos rendszerek osztályozása Michael J. Flynn szerint**

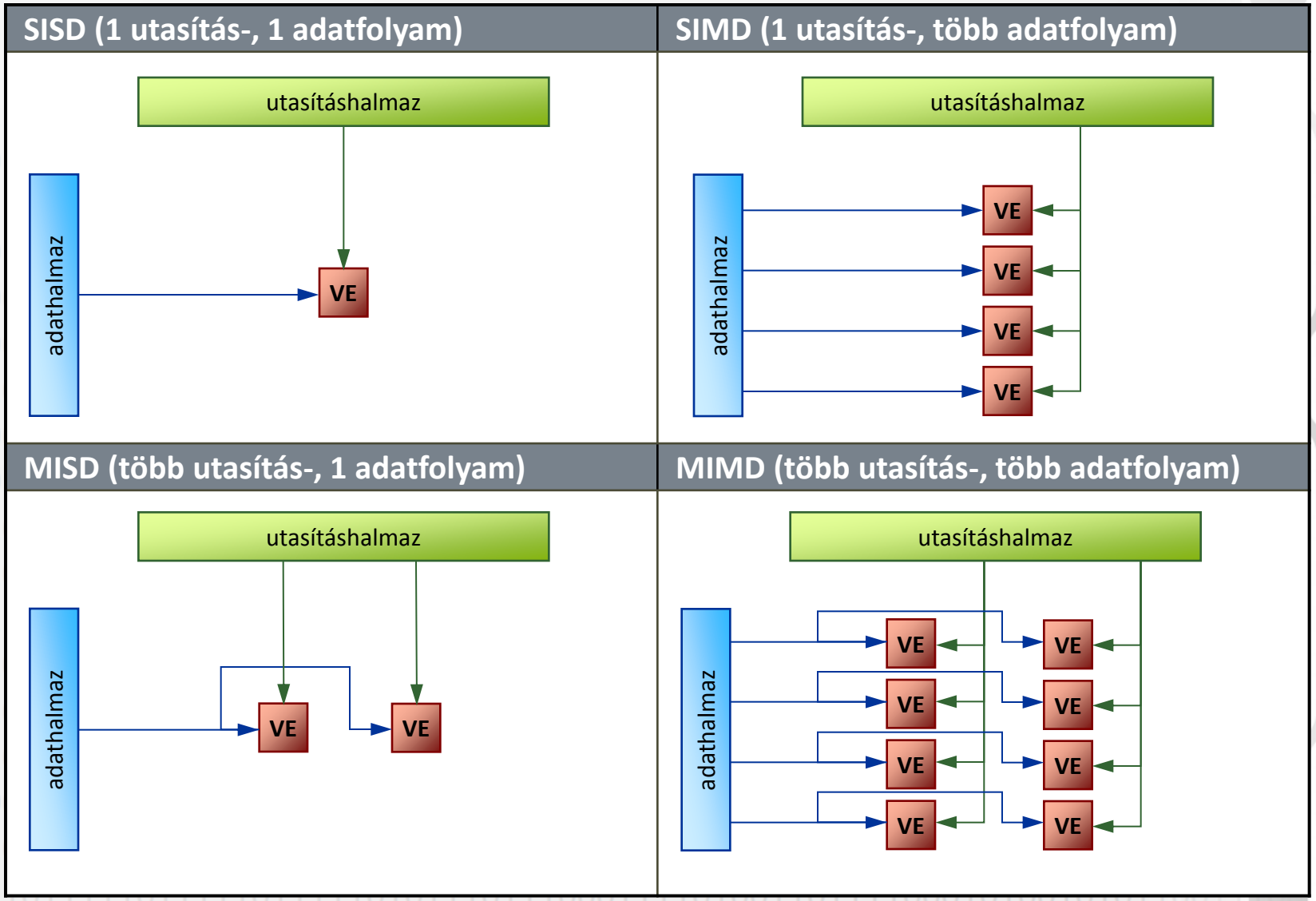
  Az 1966-ból származó Flynn-féle osztályozás a párhuzamos programozás legismertebb és legnépszerűbb tárgyalási kerete. Alapfeltevése, hogy minden („átlagos”) számítógép utasításfolyamokat hajt végre, melyek segítségével adatfolyamokon végez műveleteket. Az utasítás- és adatfolyamok száma alapján Flynn négy kategóriát állított fel:

SISD (1 utasítás-, 1 adatfolyam)	SIMD (1 utasítás-, több adatfolyam)
Soros működésű, „hagyományos” számítógépek	Vektorszámítógépek. Több formában létezett már; ma SSEx kiterjesztések, ill. DSP és GPGPU architektúrák formájában éli reneszánszát.
MISD (több utasítás-, 1 adatfolyam)	**MIMD (több utasítás-, több adatfolyam)**
Hibatűrő architektúrák (pl. űrrepülőgép), melyeknél több VE is elvégzi ugyanazt a műveletet, és az eredményeknek egyezniük kell	Teljesen párhuzamos számítógép, melynél minden végrehajtóegység külön-külön programozható fel.

Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

A Flynn-féle osztályozás

- **Illusztráció**



Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

A Flynn-féle osztályozás fogyatékosságai

- **A MISD kategória nehezen kezelhető**
- **Az utasítások és az adatok áramlása a valóságban nem folytonos**

  Általában a memóriából származnak az adatok, és bizonyos méretű „löketekben” (soronként) kerülnek a gyorsítótárakba.
- **Eltúlozza az egy-, illetve a több utasításfolyamot kezelő modellek közötti megkülönböztetés fontosságát**
- **Nehezen finomítható, nem ismeri el köztes osztályok létezését**

  Az SPMD (1 program, több adatfolyam) modell esetében például minden végrehajtóegység egyazon programot hajt végre, de a programon belül egy-egy időpillanatban más-más pozícióban is tartózkodhatnak.

  Az MPMD (több program, több adatfolyam) modellnél egy végrehajtóegység egy „mesterprogramot” futtat, amely a többi feldolgozóegység számára egy közös „alprogramot” oszt szét, és ez utóbbi valósítja meg a több adatfolyam feldolgozását.
- **Erősen hardverközpontú**

  Ez az osztályozási szempontrendszer kevéssé segíti párhuzamos algoritmusok kialakítását.

Párhuzamos rendszerek modern osztályozása

- **Ma már a Flynn-féle osztályozáshoz viszonyítva sokkal árnyaltabb szempontrendszer alakítható ki**
  - Programozási nyelv és környezet dimenziója: hagyományos, bővített hagyományos, explicit párhuzamos, agnosztikus...
  - Végrehajtási elemek együttműködésének dimenziója: osztott adatszerkezetek, üzenetváltás...
  - Szemcsézettség dimenziója: implicit párhuzamosság, fordítóprogram által vezérelt párhuzamosság, többpéldányos hardverelemek, osztott memória kontra hálózati kommunikáció...
    - Ez a szempont inkább folytonos spektrumnak tekinthető, mintsem diszkrét értékhalmaznak

Elméleti (idealizált) modellek 1

- **PRAM (Parallel Random Access Machine) modell**

  Az elméleti RAM gép (Random Access Machine; gyakorlatilag az összes Neumann-elvű számítógép) továbbgondolása párhuzamos formában.

  A PRAM lényegében egy osztott memóriájú absztrakt számítógép, amely párhuzamos algoritmusok bonyolultságának elemzését teszi lehetővé elvben tetszőleges számú feldolgozóegységgel. Mivel elhanyagolja a kommunikáció és a szinkronizáció problémáit, a rá készített algoritmusok becsült „költsége”:

  *O(időigény * processzorszám)*

  A memóriaírási és -olvasási műveletek ütközésének (egyazon memórahelyre való egyidejű írási és olvasási igények) kezelésére négy lehetőséget ad:

  - EREW: Kizárásos (exkluzív) olvasás, kizárásos (exkluzív) írás
  - CREW: Egyidejű olvasás, kizárásos írás
  - ERCW: Kizárásos olvasás, egyidejű írás
  - CRCW: Egyidejű olvasás, egyidejű írás

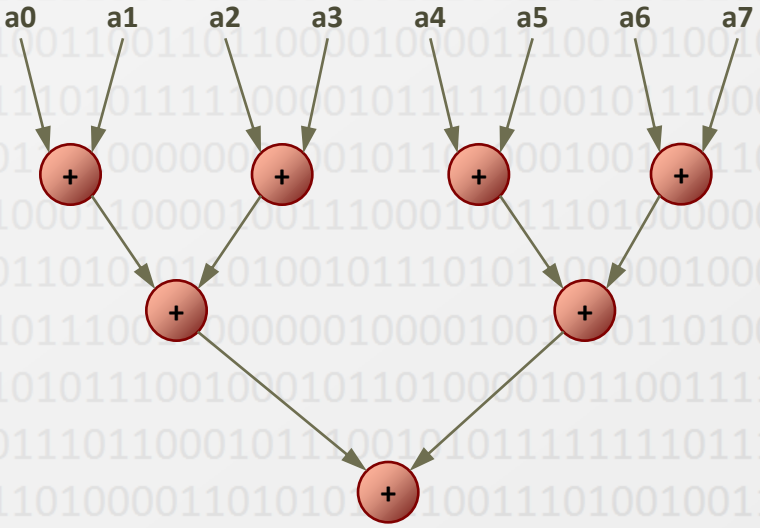
  Az egyidejű olvasás nem probléma, az egyidejű írás kezelési lehetőségei:

  - Közös: minden írási művelet ugyanazt az adatot írja, ellenkező esetben hiba történt
  - Önkényes: az egyik írás sikeres, a többi eredménytelen (nem determinisztikus)
  - Prioritásos: a feldolgozóegységek fontossági mutatója dönti el, melyik írás lesz sikeres
  - Egyéb lehetőségek (AND, OR, SUM stb. műveletekkel kombinált eredmény beírása)

Elméleti (idealizált) modellek 2

- **Adatfolyam-gráf modell**

  Ennél a modellnél a bemenő adatfolyamokból a modellstruktúrát felépítő feldolgozóegységek kimenő adatfolyamo(ka)t állítanak elő. Az adatok közötti függőségek és a párhuzamos végrehajtás módja adatfolyam-ábra segítségével illusztrálható.



Hol figyelhető meg a párhuzamosság?
Mitől függ a futási idő?
Hogyan fejezhetők ki pl. ciklusok?

Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

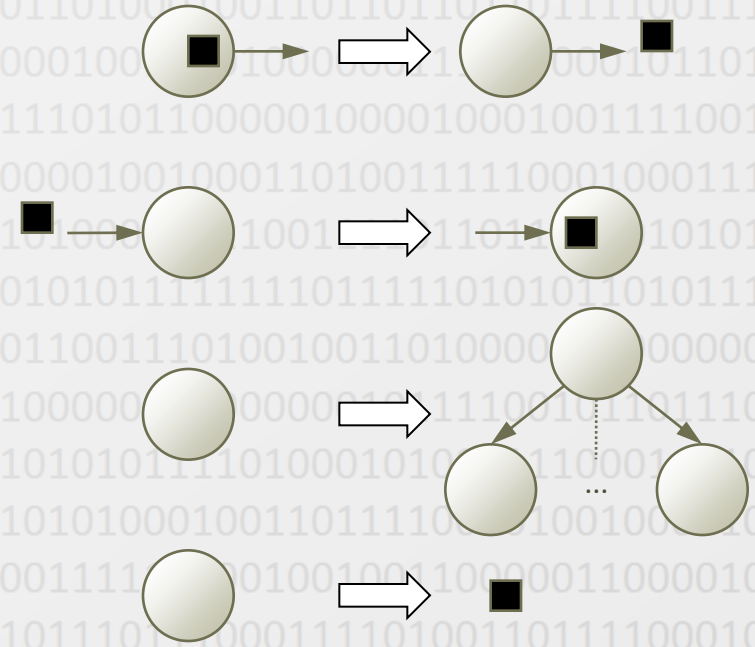
Elméleti (idealizált) modellek 3

- **Feladat/csatorna (Task/Channel) modell (Ian T. Foster, 1995)**

  Ez a modell a párhuzamosan elvégzendő számításokat tetszőleges számú, egyidejűleg végzett *feladat*ok halmazaként fogja fel, amelyeket egymással kommunikációs *csatorná*k kapcsolnak össze. Egy feladat önmagában egy hagyományos soros végrehajtású program, melynek helyi memória, valamint és be- és kimeneti portok állnak rendelkezésére.

  A feladatok végrehajtása során a helyi memória tartalmának olvasása és írása mellett négy alapműveletre kerülhet sor:

  - Üzenet küldése
  - Üzenet fogadása
  - Új feladat létrehozása
  - Feladat befejeződése

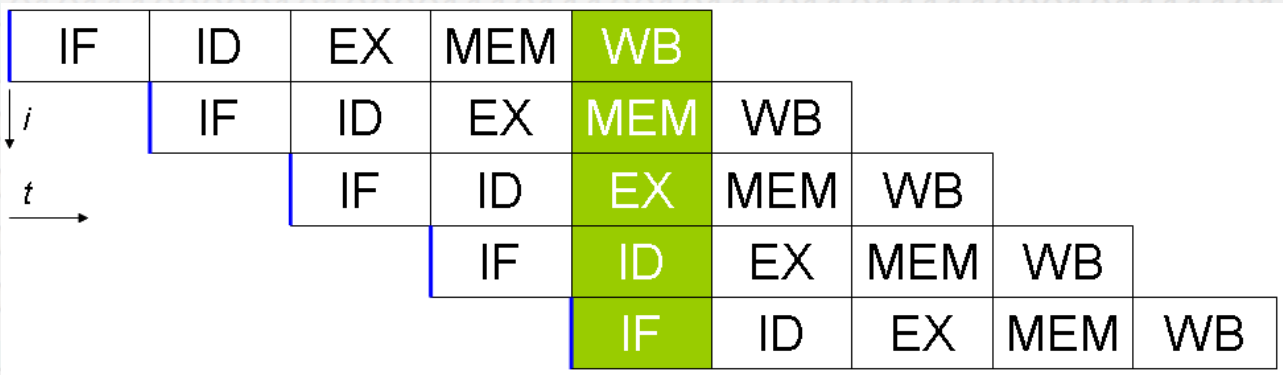


Gyakorlati modellek 1: végrehajtási modellek

- **Futószalagelvű végrehajtás**

  Az elv lényege, hogy a végrehajtást ún. lépcsőkre osztjuk, és az egyes utasításokat ezen a többlépcsős „futószalagon” folyamatosan, azaz órajelenként egyesével „léptetjük végig”.

  Az alábbi példa egy 5 lépcsős klasszikus RISC futószalagot ábrázol:



Ábramagyarázat: IF = utasításlehívás (Instruction Fetch), ID = dekódolás (Instruction Decode), EX = végrehajtás (Execute), MEM = memóriahozzáférés (Memory access), WB = visszaírás (Register write back). A vízszintes tengely (*t*) az időt, míg függőleges tengely (*i*) az utasítások sorozatát jelöli. A zöld színű oszlopban látható, hogy az első utasítás már a WB lépcsőnél tart, miközben ezzel egyidőben pl. az utolsó (azaz ötödik) utasítás lehívása van folyamatban.

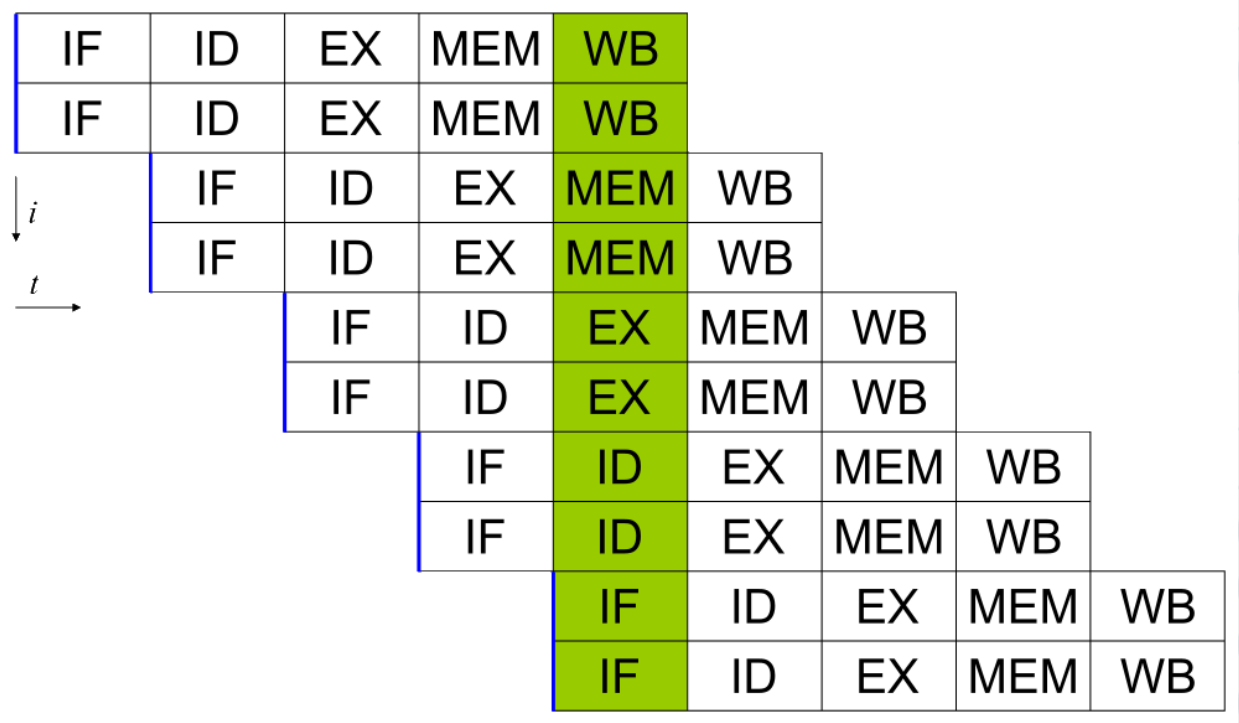
A sebességnövelés korlátját itt elsősorban a lépcsők száma, a leglassabb lépcső végrehajtási ideje, valamint a váratlan események (gyorsítótárban nem található meg az adat, elágazás következett be stb.) szabja meg.

Gyakorlati modellek 1: végrehajtási modellek

- **Szuperskalár végrehajtás**

  Ennel a megoldásnál egyszerre több futószalag működik párhuzamosan.

  Az alábbi példa egy 2 utas (két futószalagos) szuperskalár megoldást ábrázol:



A szuperskalár végrehajtás szélesítésének elvi gátat szabnak a valódi adatfüggőségek, amelyek mai programokban csak legfeljebb 4–6 utasítás egyidejű végrehajtását teszik lehetővé.

Gyakorlati modellek 1: végrehajtási modellek

- **Szuperskalár végrehajtás: az optimalizálás szerepe**

  Az alábbi három kódpélda szemantikailag (értelmét és eredményét tekintve) egyenértékű, mégis egészen eltérő teljesítményjellemzőkkel bír:

A

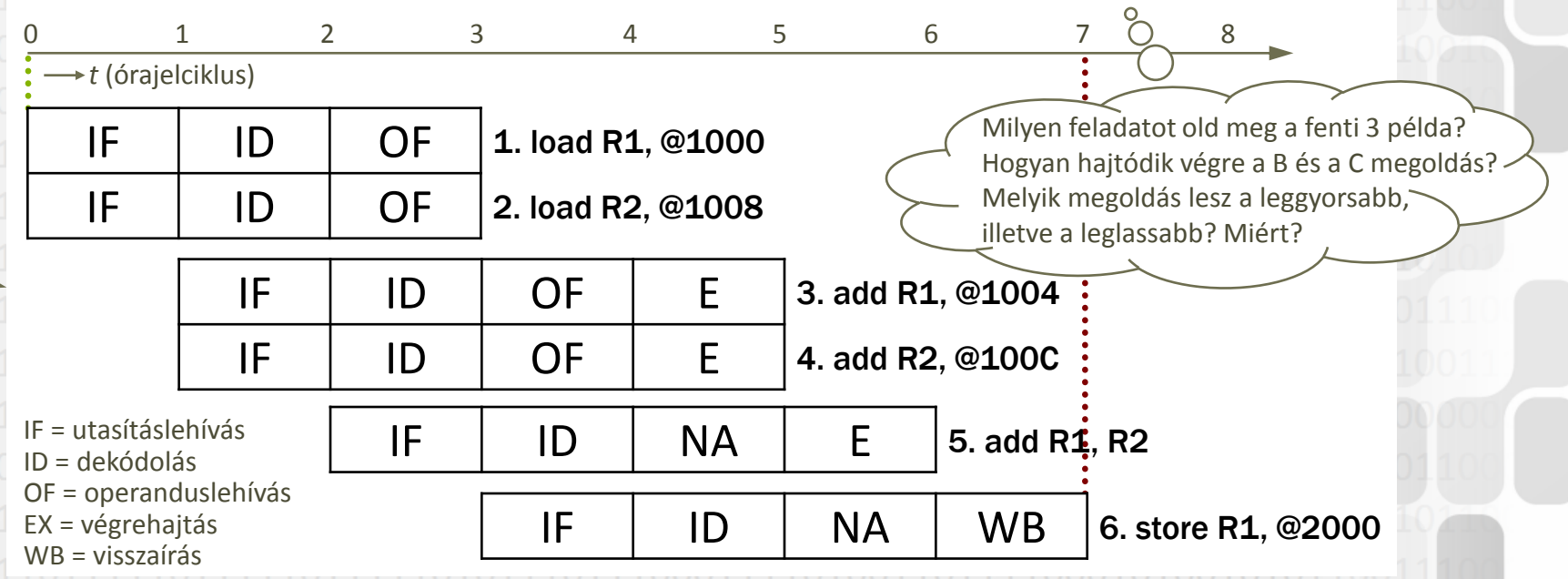
1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000

B

1. load R1, @1000
2. add R1, @1004
3. add R1, @1008
4. add R1, @100C
5. store R1, @2000

C

1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000

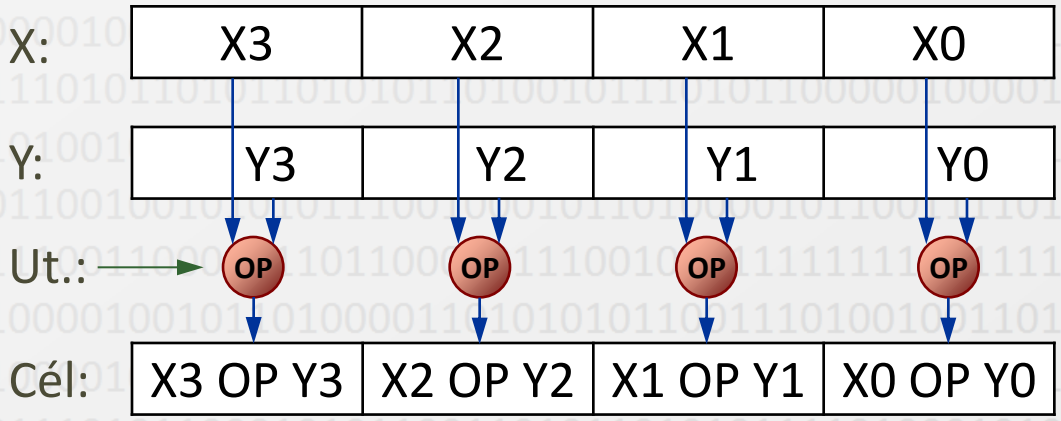


Gyakorlati modellek 1: végrehajtási modellek

- **Vektoralapú (SIMD) végrehajtás**

  A vektoralapú megoldások lényege, hogy a fordítóprogram automatikusan (vagy programozói segítséggel) észleli a függőségek nélküli azonos kódrészleteket. SIMD végrehajtás esetén a program más-más adatokon végzi el ugyanazt a műveletsort, ezt pedig párhuzamosan is végre lehet hajtani.

  A SIMD műveletek végrehajtási elvét illusztrálja az alábbi ábra és kódpélda*:




```
void Quarter(int[] array)
{
        for (int i = 0; i< array.Length; i++)
            array[i] = array[i] >> 2;
}
```

```
void QuarterSSE2(int[] array)
{
    ...
$B1$10:
    movdqa      xmm0, XMMWORD PTR [edi+edx*4]
    psrad       xmm0, 2
    movdqa      XMMWORD PTR [edi+edx*4], xmm0
    add         edx, 4
    cmp         edx, ecx
    jb          $B1$10
    ...
}
```

* Intel Corporation, „The Software Optimization Cookbook”, 2006, ISBN 0-9764832-1, 192. és 195-196. oldal

Gyakorlati modellek 1: végrehajtási modellek

- **Vektoralapú (VLIW) végrehajtás**

  A VLIW végrehajtásnál egy igen hosszú („very long”) gépi kódú utasításszóba („instruction word”) a fordítóprogram a megfelelő elemzést követően számos elemi utasítást egymás mellé „csomagol”, és az egy „csomagban” szereplő utasítások párhuzamosan hajtódnak végre.

  Sajnos a párhuzamosítási lehetőségeket előre felismerni és megfelelően kiaknázni képes fordítóprogramok készítése egyelőre a lehetetlenséggel határosan nehéz*.

  Példa VLIW műveletekre:

Utasítás				
n-1	...	...	...	...
n	BRANCH	ADD	ADD	MUL
n+1	...	...	...	...

* Donald E. Knuth, „Interview with Donald Knuth”, 2008. április 25., http://www.informit.com/articles/article.aspx?p=1193856

Gyakorlati modellek 1: végrehajtási modellek

- **Szál szinten párhuzamos végrehajtás**

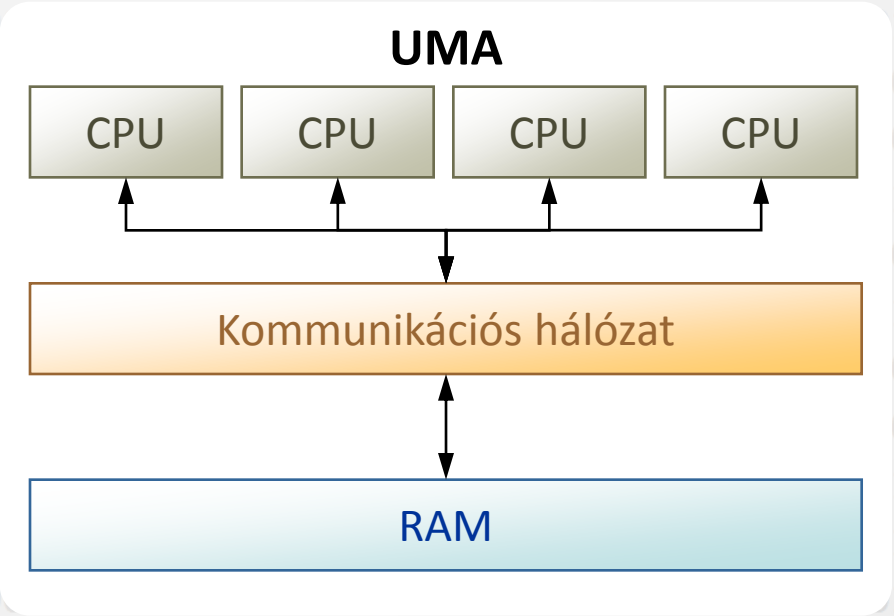
  Ennél a modellnél azt használjuk ki, hogy egy végrehajtási szálon belül biztosan lesznek feloldhatatlan adatfüggőségek, amelyek korlátozzák a párhuzamosítást. A várakozások miatt felszabaduló számítási kapacitást a modell úgy igyekszik növelni, hogy egy-egy végrehajtóegységet egynél több szálból is „ellát” feladatokkal. Általában az operációs rendszer ezt a beépített szál szintű párhuzamosságot több „logikai” feldolgozó egységként kezeli.

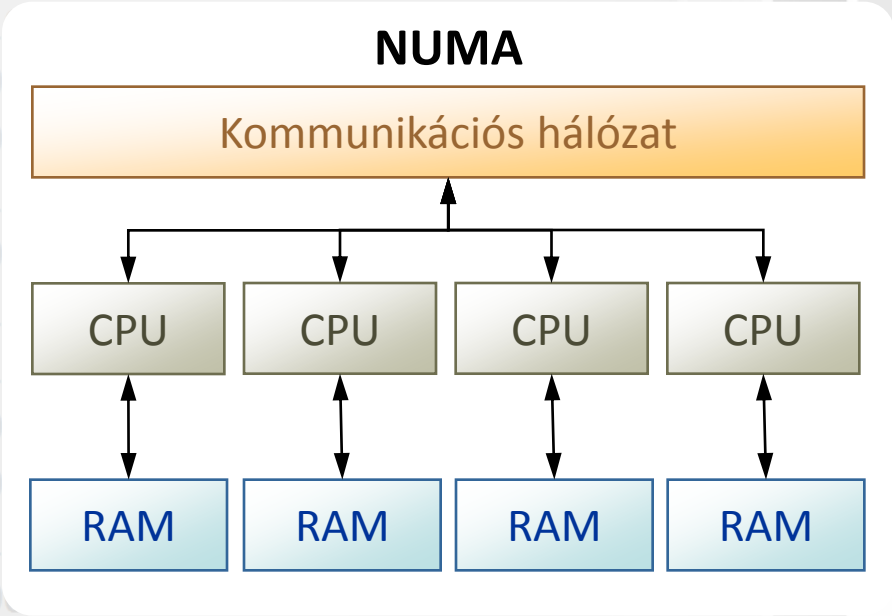
  A szál szintű párhuzamosság főbb megvalósítási lehetőségei:

  – Szimmetrikus többszálú feldolgozás (SMT)
    - Egy processzoron belül bizonyos részegységek (de nem a teljes processzor) többszörözése
    - Az Intel ezt „Hyperthreading” néven valósította meg
  – Többmagos processzor
    - Egy fizikai egységen (tokon) belül több azonos, teljes értékű processzor, melyek egymáshoz speciális adatutakkal is kapcsolódhatnak
  – Többmagos processzor SMT-vel
    - Az első két megoldás kombinációja

Gyakorlati modellek 2: memóriamodellek

- **Az UMA és a NUMA modell**
  - UMA: Uniform Memory Access (közös memória)
    - Minden processzor egyetlen közös memóriában tárolja adatokat
    - Más néven SMP (symmetric multiprocessor) architektúra
  - NUMA: Non-Uniform Memory Access (elosztott memória)
    - Minden processzor saját, független memóriával rendelkezik
    - Más néven DM (distributed memory) architektúra



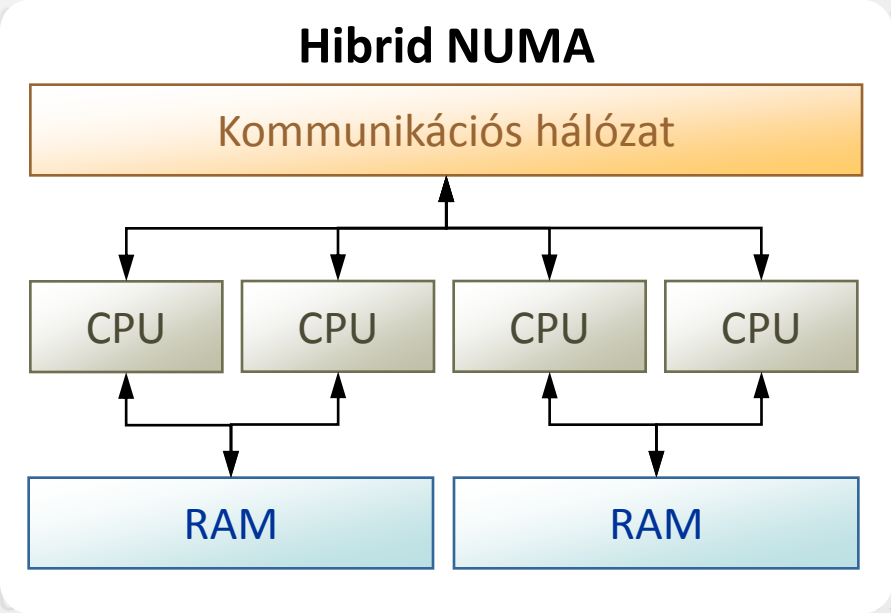


Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

Gyakorlati modellek 2: memóriamodellek

- **A Hibrid (N)UMA modell**

  Ez a rendszer egynél több memóriablokkot tartalmaz (tehát nem tisztán UMA), de ezeket a processzorok közösen is használják (tehát nem is tisztán NUMA). A nagy teljesítményű párhuzamos számítógépek (más néven „szuperszámítógépek”) körében egyértelmű trend a hibrid NUMA megoldás terjedése.



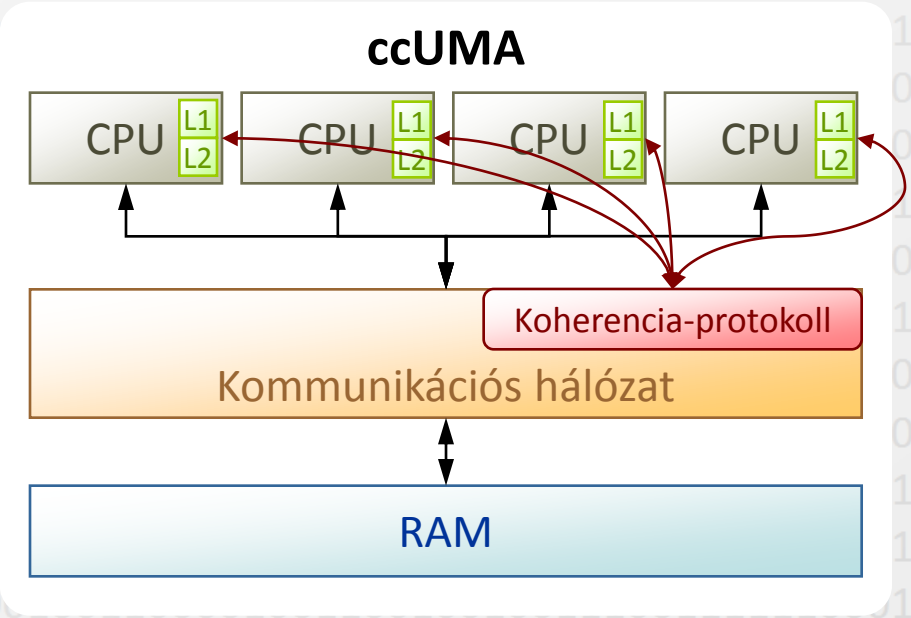
Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

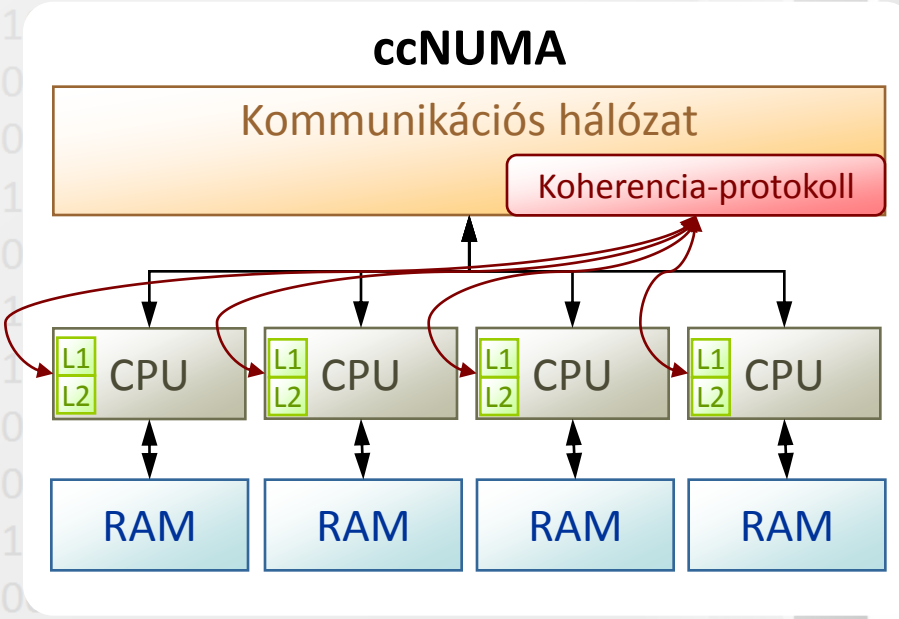
Gyakorlati modellek 2: memóriamodellek

- **A ccUMA és a ccNUMA modell**

  A processzorok mindig rendelkeznek gyorsítótárakkal, melyek a memóriából betöltött, illetve oda kiírandó adatok egy részét tárolják. Ezek tartalmának összehangolása külön feladatként merül fel, mely az UMA és NUMA modellekben igen bonyolultan oldható meg.

  A ccUMA (cache-coherent UMA) és ccNUMA modelleknél speciális, egy ún. koherencia-protokollt megvalósító egység gondoskodik az összehangolásról.





Miklós Árpád, Dr. Vámossy Zoltán, OE NIK, 2010
miklos.arpad@nik.uni-obuda.hu

Kommunikáció és kommunikációs modellek

- **A párhuzamos rendszerek elemei közötti kommunikáció tervezése a teljesítménynövelés szempontjából döntő tényező**

  A feldolgozás részeredményeinek továbbítása az egyes feldolgozóegységek között (azaz a feldolgozóegységek kommunikációja) a párhuzamos programozás „szükséges rossz” jellegű tényezője.

  Az algoritmusok kialakításánál ezért szintén lényeges feladat a kommunikációs igények minimalizálása, a megfelelő szoftveres kommunikációs struktúra megtervezése.

Kommunikációs hálózati topológiák

- **Egyszerű topológiák**
  - Teljes
  - Csillag
  - Lineáris (1D) tömb
  - 2D és 3D tömb (átfordulással vagy anélkül)

Kommunikációs hálózati topológiák

- **Hiperkocka**
  - 0D, 1D, 2D, 3D, 4D hiperkocka

Kommunikációs hálózati topológiák

- **Fák**
  - Statikus bináris fa
  - Dinamikus bináris fa

Hálózati topológiák jellemzési szempontjai

- **Átmérő**

  Bármely két csomópont között mérhető legnagyobb távolság.

- **Szomszédos élek száma**

  Bármely két csomópont közötti összeköttetések száma.

- **Kapcsolati számosság**

  Az a minimális élszám, melynek eltávolítása esetén a hálózat két, egymástól független hálózatra válik szét.

- **Felezési szélesség**

  Az a minimális élszám, melynek eltávolítása esetén a hálózat két, egymástól független, **egyforma** hálózatra válik szét.

- **Felezési sávszélesség**

  A hálózat bármely két része között minimálisan meglévő átviteli sávszélesség.

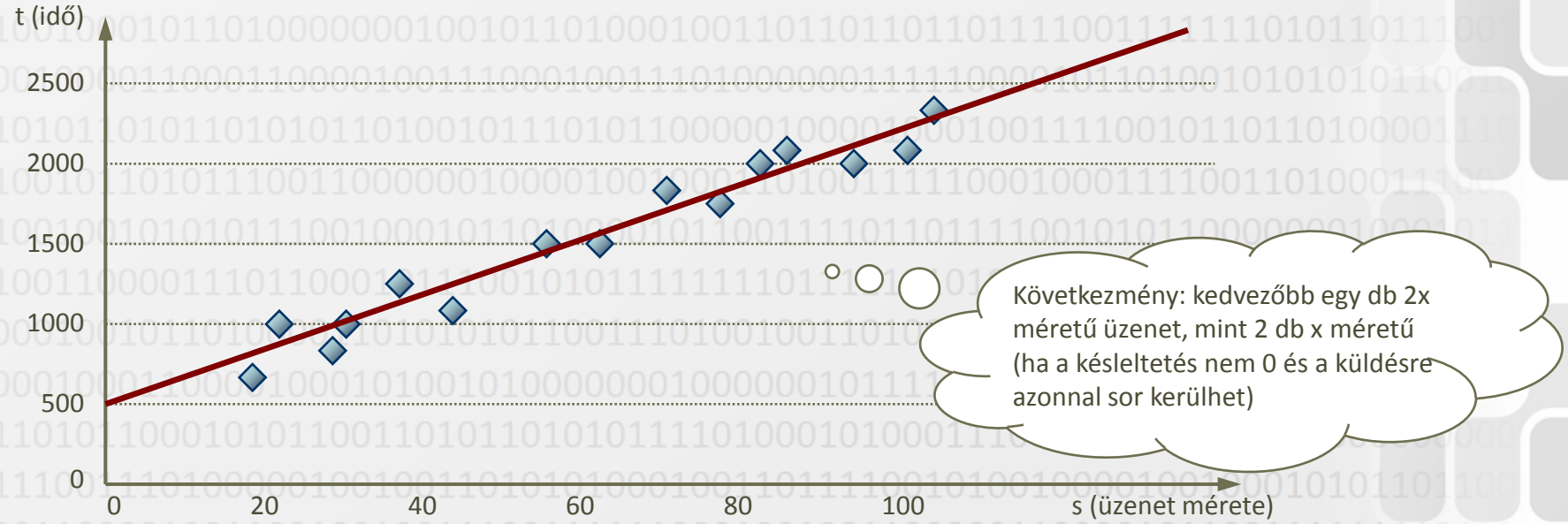
- **Költség**

  A kapcsolatok számával, átviteli paraméterekkel stb. arányos jellemző.

A kommunikáció időbeli modellezése

- **A kommunikáció időbeli lefolyásának modelljét két fő tényező jellemzi:**
  - *Késleltetés* (latency): egy-egy üzenet indítása, illetve megérkezése között eltelt idő
  - *Sávszélesség* (bandwidth): időegységenként átvihető maximális adatmennyiség

  Általánosságban a kommunikáció időbeli diagramja az alábbihoz hasonló:



A kommunikáció időbeli modellezése

- **A kommunikáció hatékonyságának növeléséhez elengedhetetlen a fenti két fő tényező kedvezőbbé tétele**

  A sávszélesség növelését általában a kommunikációs hálózat közegének fizikai adottságai korlátozzák, a késleltetés azonban a struktúra kialakításának változtatásával, illetve jobb algoritmusokkal csökkenthető.

  A késleltetés csökkentésének lehetőségei (példák):

  – Nem blokkoló küldési és fogadási műveletek

    A küldő és a fogadó oldalon pufferek segítségével biztosítjuk, hogy pusztán a kommunikáció ténye miatt ne kelljen várakoznia a feldolgozóegységeknek.

  – Többszálú programozás alkalmazása

    Amikor egy-egy feldolgozási szál várakozásra kényszerül (például egy olvasási vagy írási művelet befejezése érdekében), más szálak ezzel egyidőben más feladatokat végezhetnek, így rendszer átbocsátó képessége javul.

Felhasznált és javasolt irodalom

- [1] A. Grama, A. Gupta, G. Karypis, V. Kumar
  Introduction to Parallel Computing
  Addison-Wesley, ISBN 0-201-64865-2, 2003, 2nd ed., angol, 636 o.
- [2] B. Wilkinson, M. Allen
  Parallel Programming
  Prentice Hall, ISBN 978-0131405639, 2004, 2nd ed., angol, 496 o.
- [3] S. Akhter, J. Roberts
  Multi-Core Programming (Increasing Performance through Software Multi-threading)
  Intel Press, ISBN 0-9764832-4-6, 2006, angol, 340 o.
- [4] Iványi A.
  Párhuzamos algoritmusok
  ELTE Eötvös Kiadó, ISBN 963-463-590-3, Budapest, 2003, magyar, 334 o.
- [5] J. Albahari
  Threading in C#
  [online: 2011. 04. 30.] http://www.albahari.com/threading/