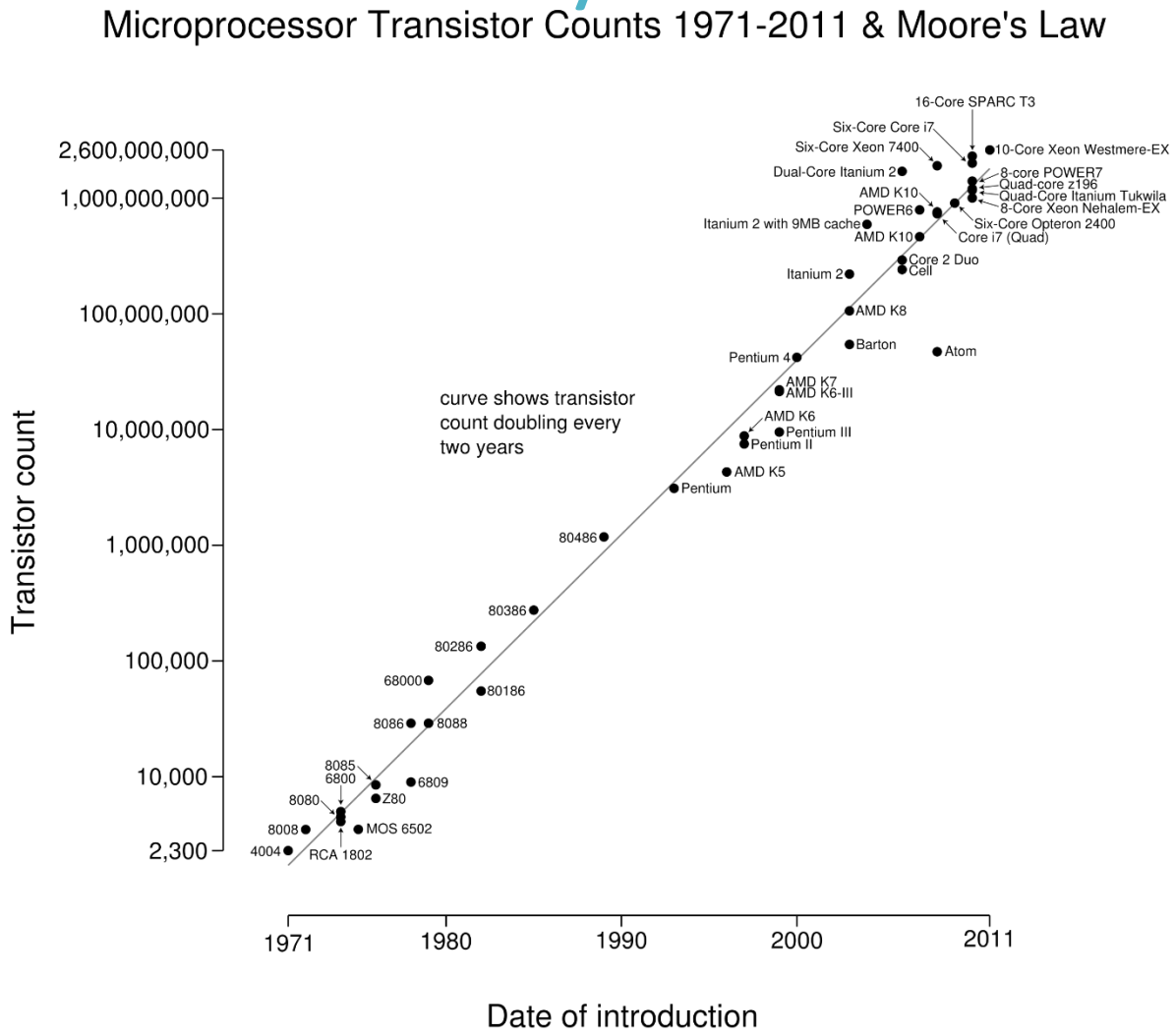
1. A párhuzamos feldolgozás elvi alapjai

kertesz.gabor@nik.uni-obuda.hu

Bevezetés

Az ingyen ebédtől a multi-core rendszerekig

Moore törvénye



Gordon Earle Moore, az Intel társalapítója 1965-ben publikált cikke híresült el később Moore törvényeként:

A mikrochipen található tranzisztorok száma 18-24 havonta megduplázódik.

Ez a megfigyelésen alapuló törvény 40 éve érvényes.

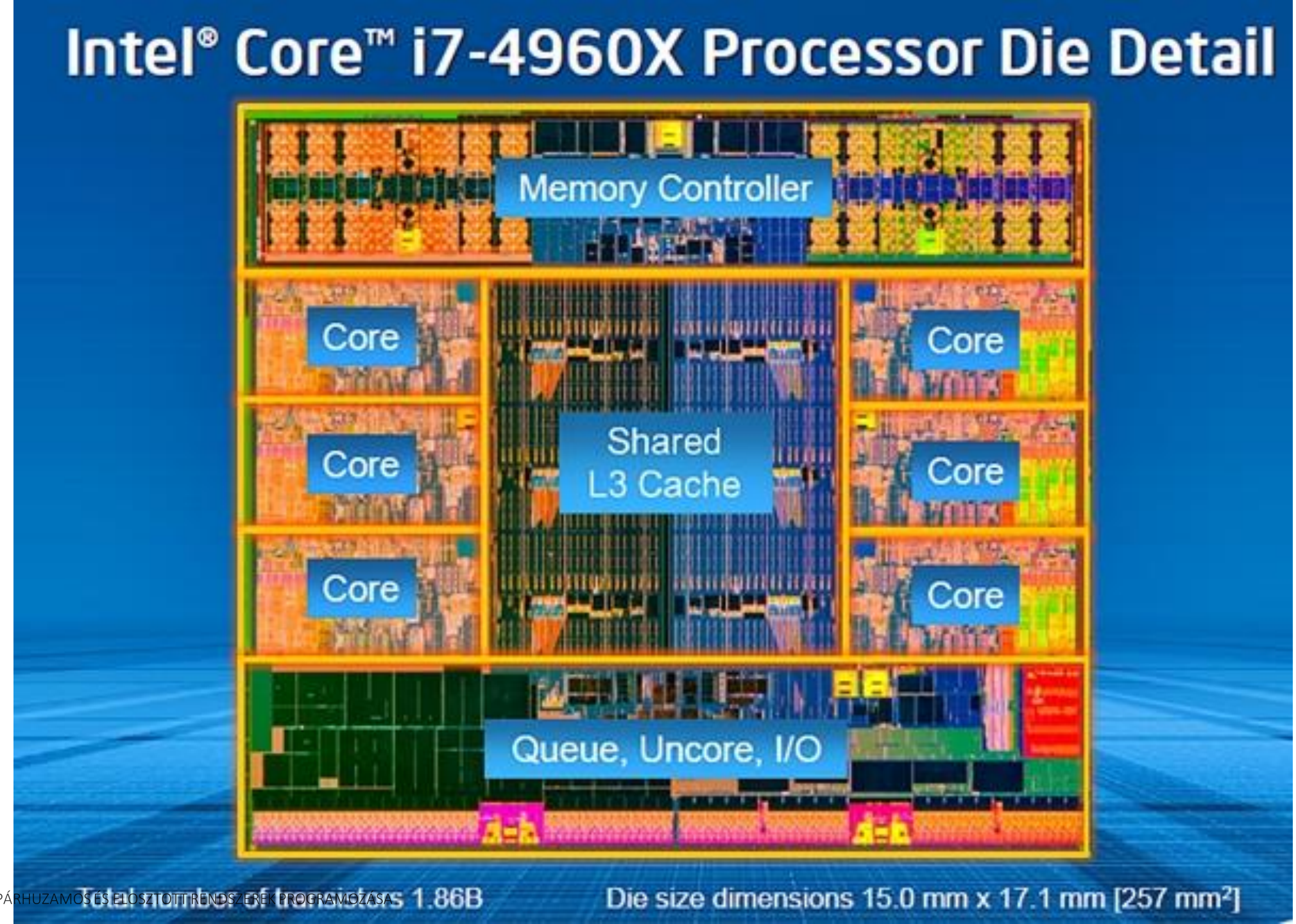
A tranzisztorok száma egyenes arányban hatott az órajel-sebességre.

Feszültségkorlát

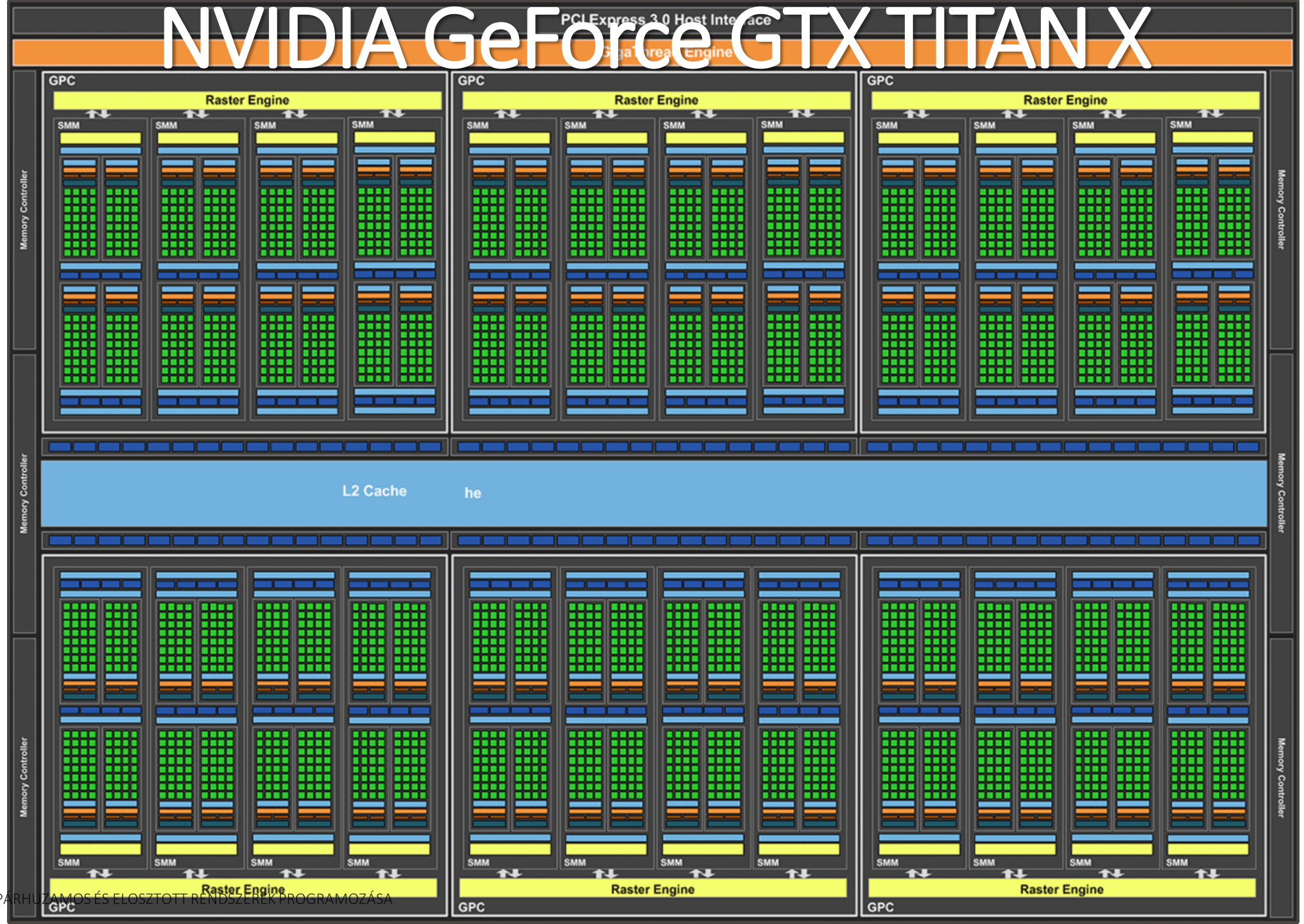
- A tranzisztorsűrűség továbbra is növekszik, de az egységek sebessége már nem
- A magasabb órajelfrekvenciához tipikusan magasabb feszültség szükséges
- Magasabb tápfeszültséghez nagyobb teljesítményű tápegység kell
- A magasabb feszültség nagyobb hőt termel

„Free lunch”

- Egy szoftverfejlesztő megkapta feladatként, hogy egy programon *gyorsítson*
- ~10 évvel ezelőtt
  - Várt egy évet, és a következő generációs, magasabb órajelű processzoron gyorsabban futott a programja: „free lunch”
- Manapság
  - A több műveletvégző alkalmazása, azaz a párhuzamos programozás az egyetlen megoldás


Intel® Core™ i7-4960X Processor Die Detail
Memory Controller
Core
Core
Core
Core
Core
Core
Shared
L3 Cache
Queue, Uncore, I/O
Total number of transistors 1.86B
Die size dimensions 15.0 mm x 17.1 mm [257 mm²]

NVIDIA GeForce GTX TITAN X



Hogyan párhuzamosítunk?

- Az általában használt szekvenciális programoktól a párhuzamosak alaposan különböznek
- Leggyakrabban nem az a leghatékonyabb, ha az egyes lépéseket párhuzamosítjuk
- A teljes algoritmus újragondolása is szükséges lehet

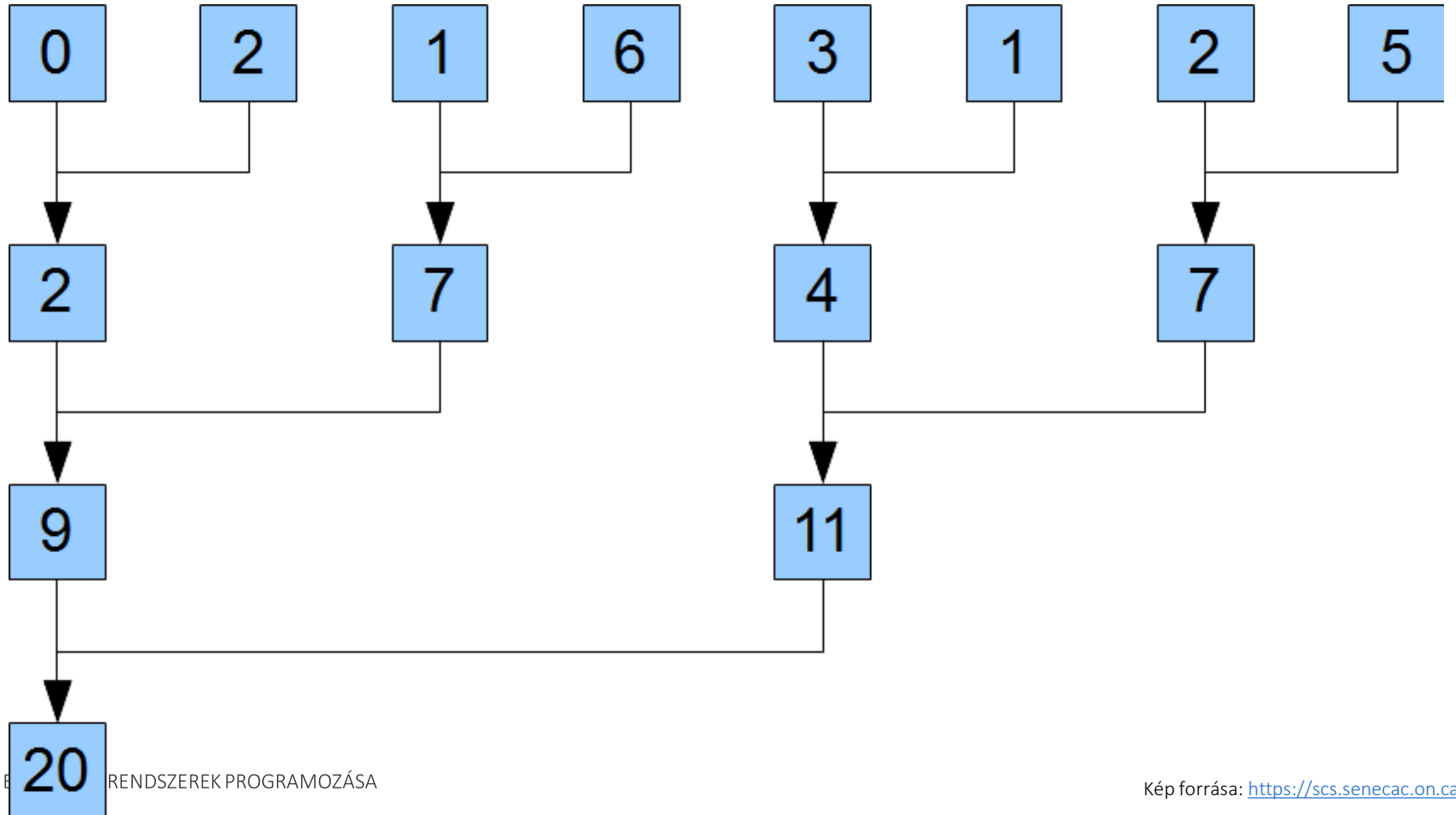
Példa - összegzés

- Adjuk meg a számok összegét!

- 1. lépés: 0+0
- 2. lépés: 0+2
- 3. lépés: 2+1
- 4. lépés: 3+6
- 5. lépés: 9+3
- 6. lépés: 12+1
- 7. lépés: 13+2
- 8. lépés: 15+5

Példa - lehetséges párhuzamos megoldás

- Parallel reduction (részletesen később)



Kép forrása: https://scs.senecac.on.ca/~gpu610/pages/content/cudam_p.html

Párhuzamos hatékonyság

Amdahl és Gustafson

Teljesítmény

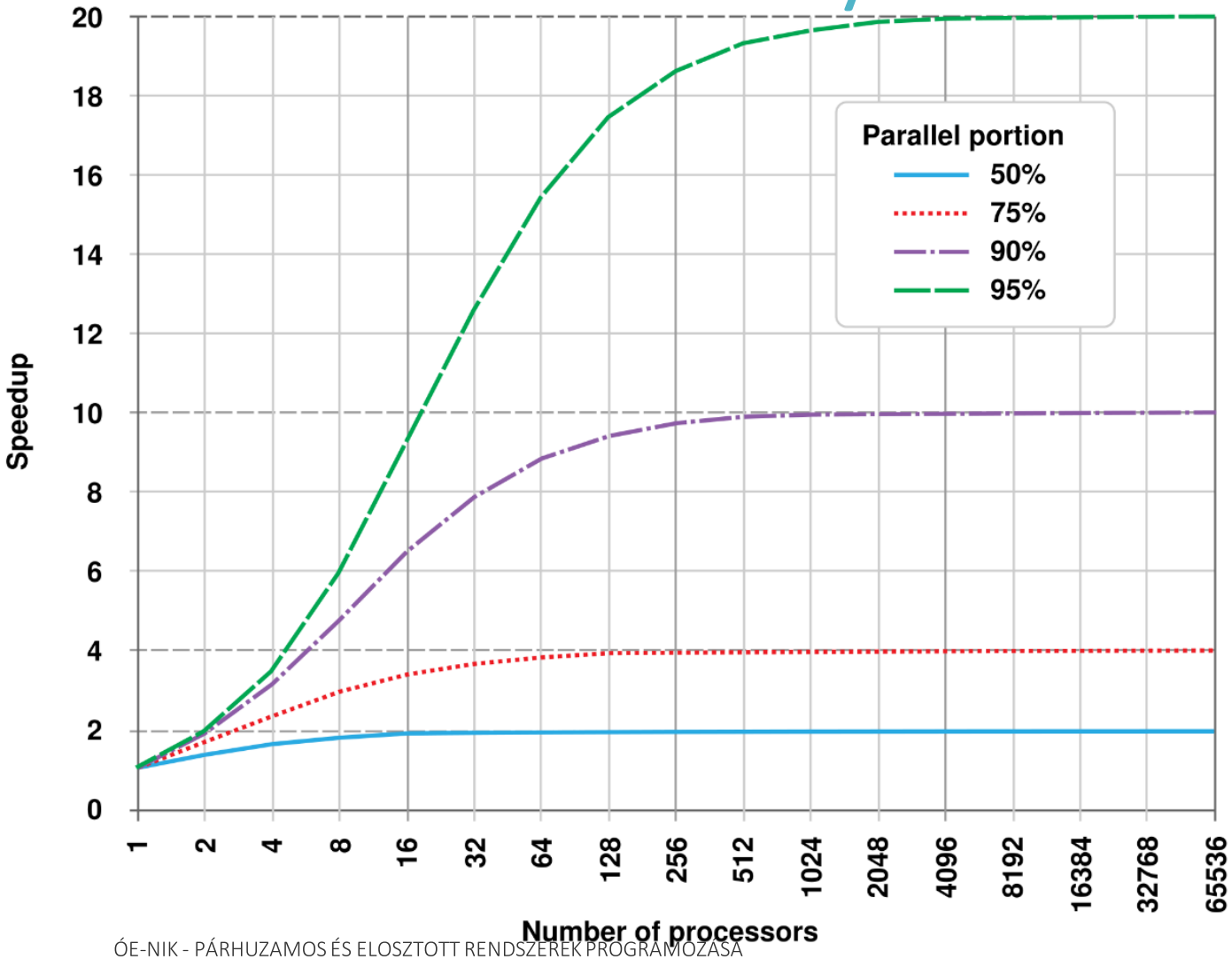
- Ahogy a magok száma növekszik, a relatív teljesítmény csökken
  - 2-4 processzor vs 8-16
  - Több ezer GPU mag
- Az igazán jó algoritmus független a műveletvégzők számától: az optimális megoldás **skálázható**

> **Scalability** is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. (…) A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a ***scalable system***. (Forrás: Wikipedia)

Futásidő

- $T_{total}(1) = T_{setup} + T_{compute} + T_{finalization}$
- $T_{total}(p) = T_{setup} + \frac{T_{compute}(1)}{p} + T_{finalization}$
- Nem minden rész párhuzamosítható

Amdahl törvénye

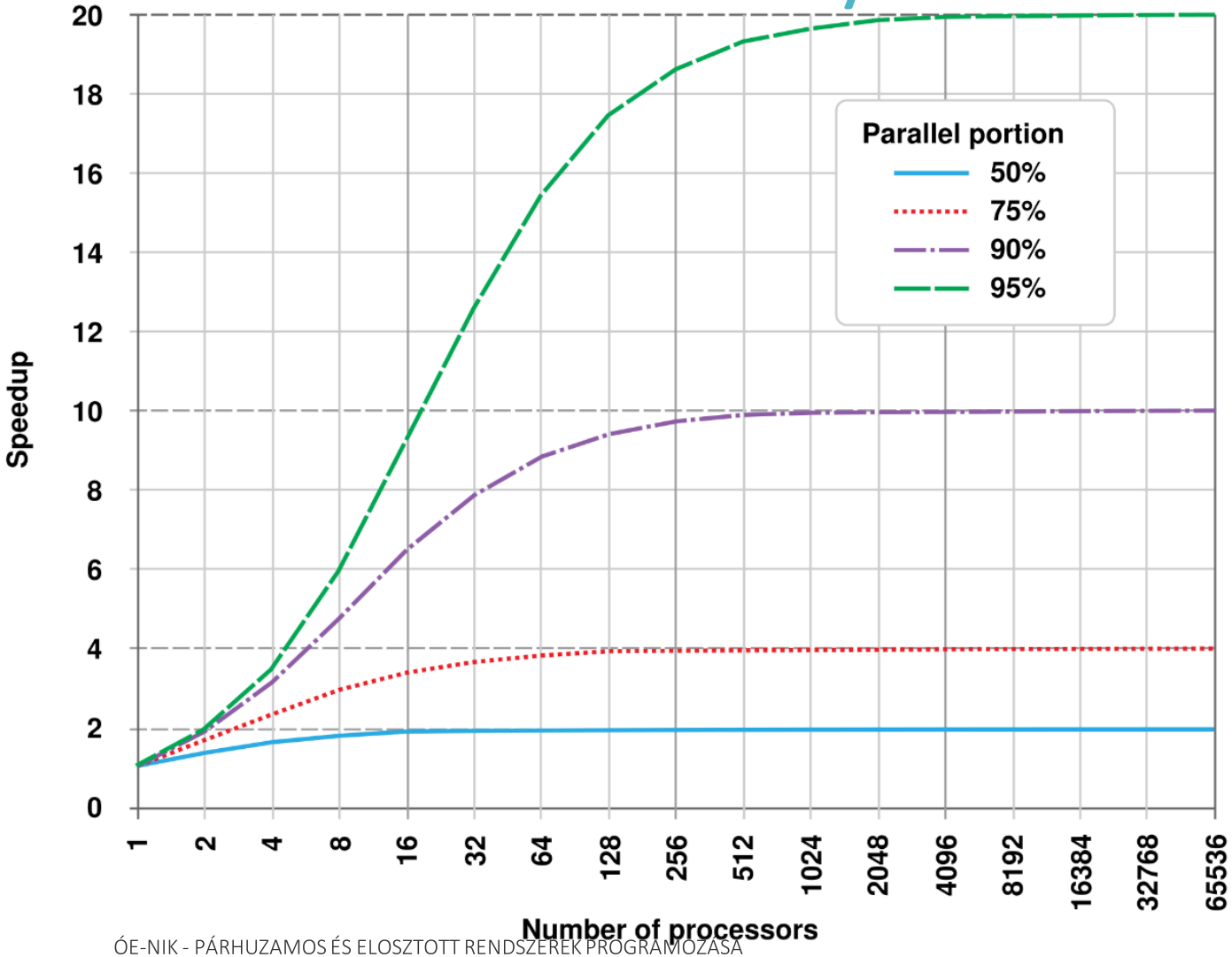


Gene Amdahl 1969-ben publikálta a róla elnevezett törvényt, amely azt modellezi, hogy egy szekvenciális program párhuzamos implementációjának futtatása során milyen kapcsolat jelenik meg a processzorok száma és a várható sebességnövekedés között.

$$S_{latency}(p) = \frac{1}{(1-f) + \frac{f}{p}}$$

$S_{latency}$: az elméleti gyorsulás mértéke
f : a párhuzamosítható szakasz (pl.: 0.5)
p : p gyorsulásának mértéke, műveletvégzők száma (pl.: 16)

Amdahl törvénye



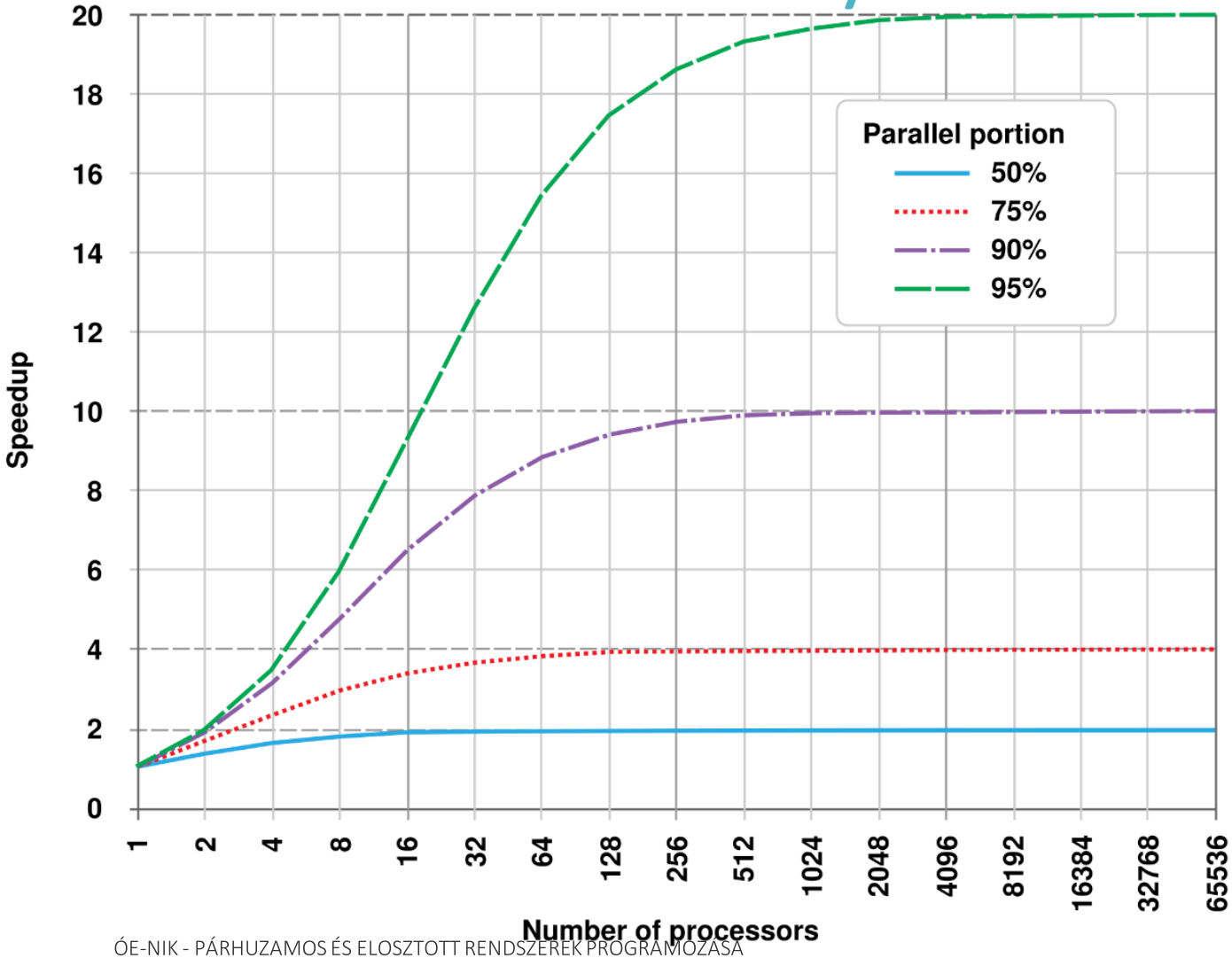
$$\lim_{s\to\infty} S_{latency}(p) = \frac{1}{1-f}$$

azaz

$$S_{latency}(p) \leq \frac{1}{1-f}$$

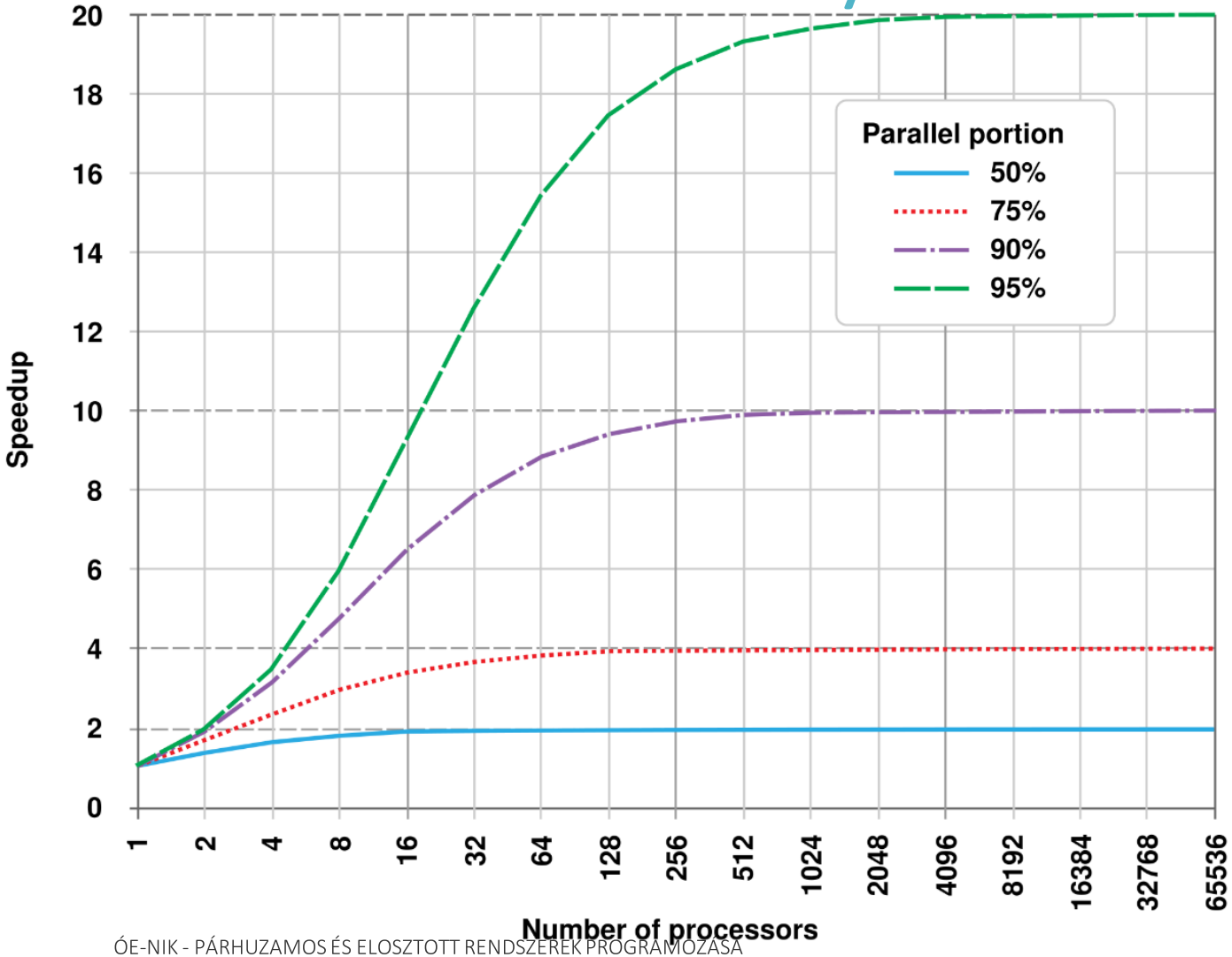
$S_{latency}$: az elméleti gyorsulás mértéke
f : a párhuzamosítható szakasz (pl.: 0.5)
p : p gyorsulásának mértéke, műveletvégzők száma (pl.: 16)

Amdahl törvénye



- Amdahl törvénye egyszerű:
  - 2-szeres teljesítménynöveléshez a program 50%-a párhuzamosítandó
  - 4-szereshez 75%
  - 10-szereshez 90%
  - 1000-szereshez 99.9%
  - 10 000-szereshez 99.99%
- És ezek nem magas célok!

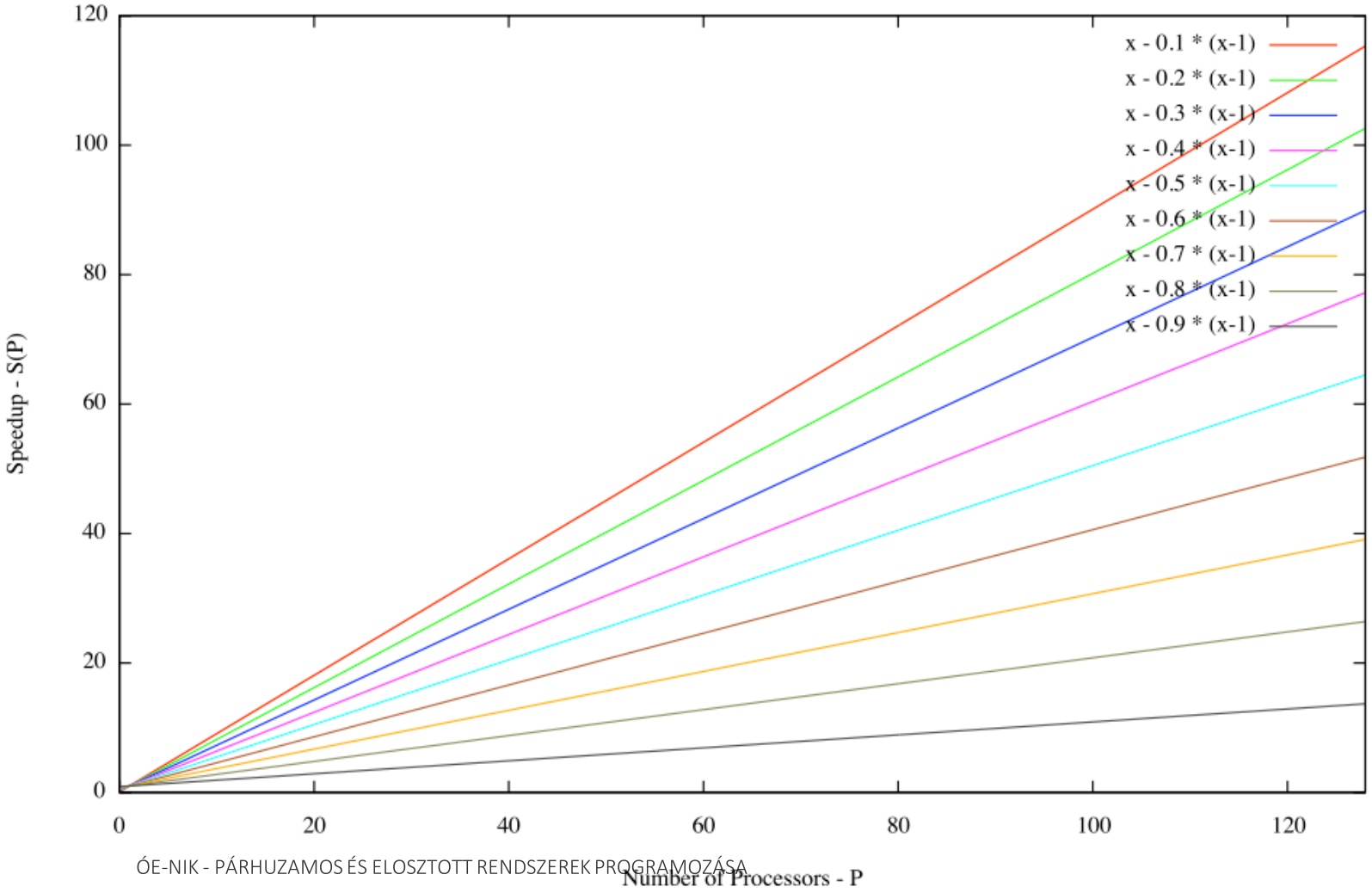
Amdahl törvénye



- Minden szekvenciálisan végrehajtandó rész **bottleneck**, amely rontja a teljesítményt

In software engineering, a bottleneck occurs when the capacity of an application or a computer system is severely limited by a single component, like the neck of a bottle slowing down the overall water flow. The bottleneck has lowest throughput of all parts of the transaction path. (Forrás: Wikipedia)

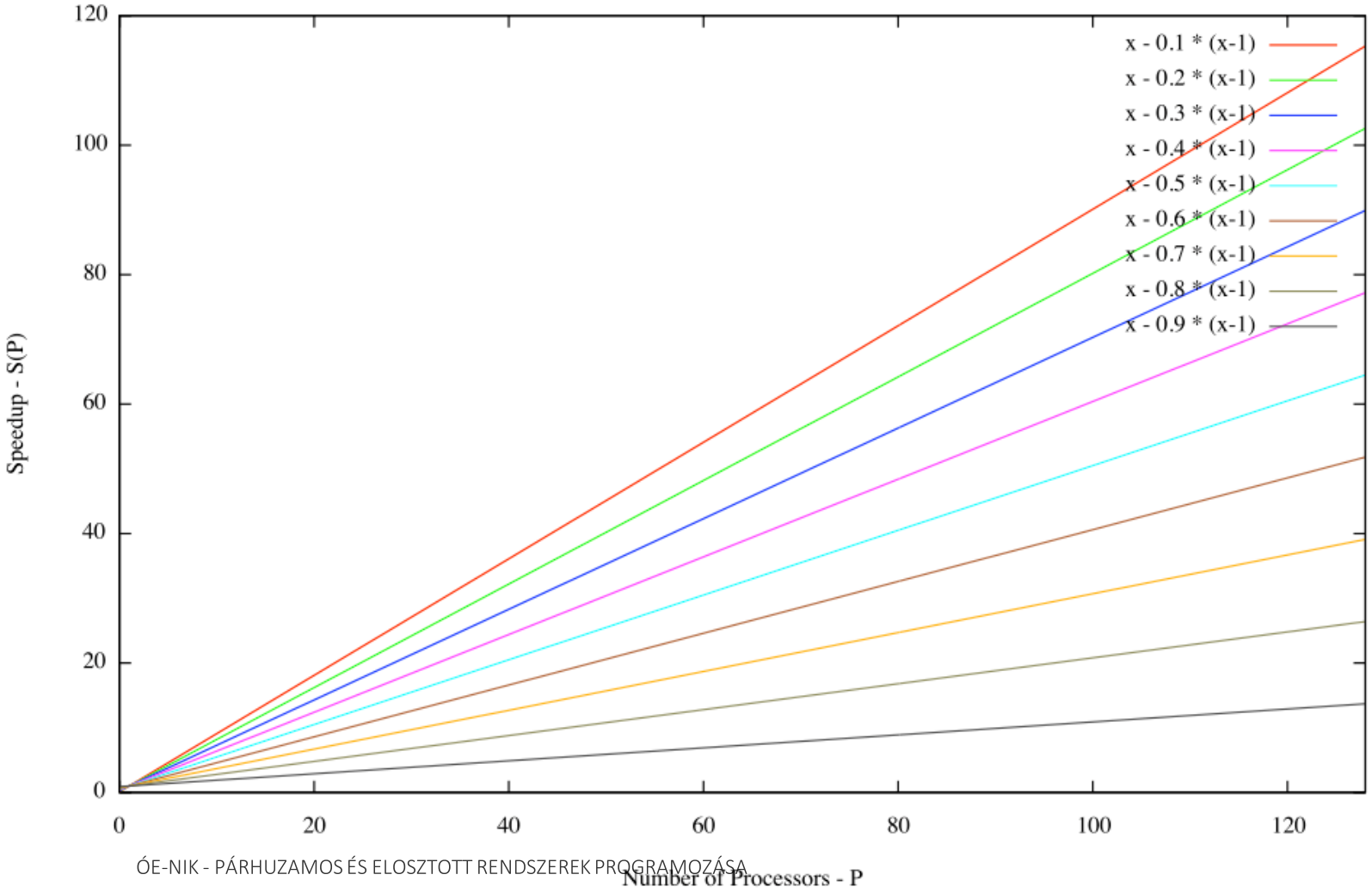
Gustafson törvénye



John L. Gustafson és Edwin H. Barsis publikálta 1988-ban a *Reevaluating Amdahl's Law* című cikküket, melyben Amdahl törvényének gyenge pontjára mutatnak rá:

Amdahl törvénye fix feladat-méretet feltételez, azaz, az erő-források növelésével nem változik a terhelés.

Gustafson törvénye



$$S_{latency}(p) = 1 - f + pf$$

$S_{latency}$: az elméleti gyorsulás mértéke

f : a párhuzamosítható szakasz (pl.: 0.5)

p : p gyorsulásának mértéke, műveletvégzők száma (pl.: 16)

Gustafson törvénye

Amdahl törvénye szerint:

Két város távolsága 60 km. Egy autó közlekedik a közöttük. Ha az autó már a távolság felét megtette 30 km/h sebességgel, akkor függetlenül attól hogy milyen gyorsan halad az út második szakaszában, *nem tud 60 km/h-nál nagyobb átlagsebességet elérni* a teljes út tekintetében (ha végtelenül gyorsan haladna, akkor 60 km/h lenne az átlagsebessége).

Gustafson törvénye szerint:

Ha elég idő és távolság van még hátra, akkor elérheti a 90 km/h átlagos sebességet, pl. ha a táv második felén 120 km/h sebességgel halad *két órán keresztül*, vagy 150 km/h-s sebességgel egy órán keresztül, stb.

Gustafson törvénye

- Míg Amdahl törvénye megadja, hogy egy fix számításigényű probléma szekvenciális megoldásának párhuzamosítása milyen hatásfokkal érhető el
  - *Ha 100-al több processzorod van, mennyivel lesz gyorsabb a megoldásod?*
- Gustafson törvénye kiemeli, hogy a párhuzamos rendszerek igazi ereje nagy méretű, korábban feldolgozhatatlannak tűnő problémák fix idejű feldolgozását ígéri
  - *Ha egy 100 processzoros párhuzamos gép 30 perc alatt oldaná meg a problémát, mennyi ideig tartana egyetlen szekvenciálisnak?*

Párhuzamos hatékonyság

- Gyorsítás (speedup)

$$S_p = \frac{T_1}{T_p}$$

- Hatékonyság (efficiency)

$$E_p = \frac{S_p}{p}$$

A párhuzamos hatékonyság annak mérőszáma, hogy milyen effektív a több műveletvégző kihasználása. A 100%-os hatékonyság azt jelentené *p* darab processzor esetén, hogy a gyorsítás *p*-szeres.

Párhuzamos hatékonyság

- A műveletvégzők számának növelésével mindig növekedni fog a hatékonyság?
  - Tegyük fel hogy a feladatot helyesen felbontottuk
  - Tegyük fel, hogy minden apró szeletén külön műveletvégző dolgozik

- Kiegyensúlyozatlanság: némely processzor munkája elfogy, míg a többi dolgozik
- A műveletvégzők felkészítése, a közöttük zajló kommunikáció és a szükséges szinkronizáció okozta *egymásra várakozás* (amely szekvenciális esetben nem a feladat része) jelentős időmennyiséget jelent a teljes feldolgozás tekintetében: **overhead**

Alapfogalmak

Konkurrencia és párhuzamosság, versenyhelyzet, holtpont

Konkurrencia

- **Konkurrencia**: két vagy több számítás történik ugyanabban az időkeretben
  - „A” feladat és „B” feladat
  - „A” végrehajtása elkezdődik majd „B” végrehajtása is elkezdődik, mielőtt „A” véget ért volna

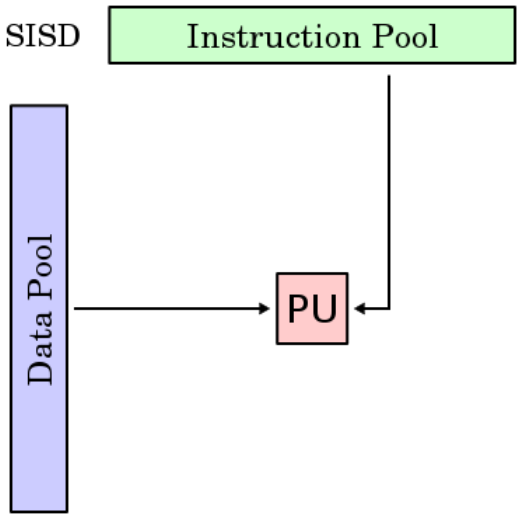
- A konkurrencia azt jelenti, hogy a két feladat megoldása egyszerre történik
- Speciális eset a **párhuzamosság**, amikor mindegyik feladat saját végrehajtó egységgel rendelkezik, és a feldolgozás szimultán (*szinkronban*) zajlik
- Másik eset az **aszinkronitás**: a végrehajtó egység feladatot válthat az aktuális feladat befejezése előtt

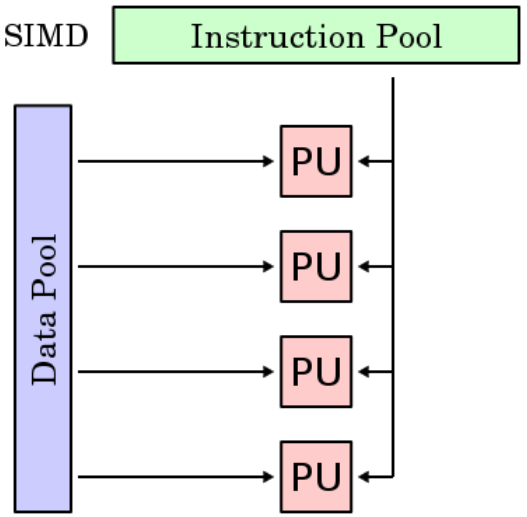
Több végrehajtó

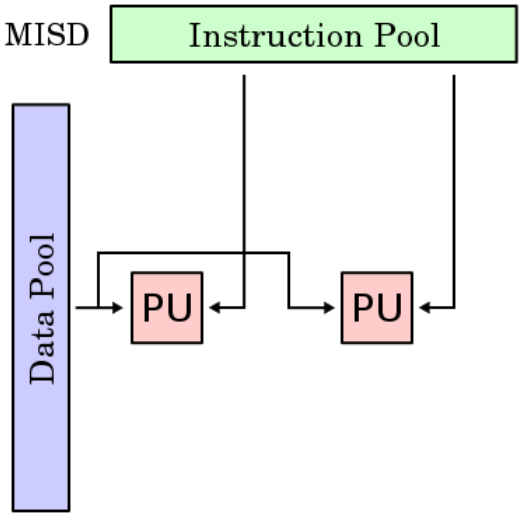
- Több processzoros rendszerek
  - Több CPU az alaplapon
  - Nehézkes kommunikáció: a RAM közös, a cache mindkét CPUn saját
- Több műveletvégző a CPUn belül
  - „Hyperthreading”: egyidejűleg két, különböző műveletvégzőt igénylő utasítás végrehajtása
  - Két logikai CPU
  - Kontextusváltás, cache-miss késleltetés: lassabb végrehajtás mint önállóan nézve
- Több mag a CPUn belül
  - „Több CPU egy CPUn”, azonban egy önálló elemként
  - Cache szinkronizáció, kommunikáció megoldott chipen belül

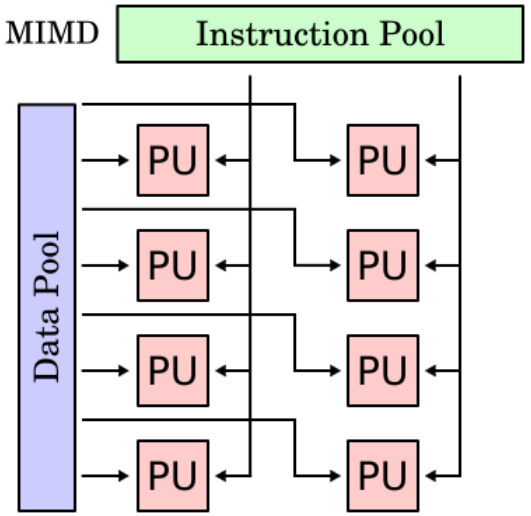
Flynn-féle osztályzás

- Michael J. Flynn, 1966
- SISD
- SIMD
- MISD
- MIMD









Terminológia

- Az operációs rendszer a programok futtatásakor azokhoz memóriaterületet és végrehajtó egységet rendel: az operációs rendszer egyszerre több **folyamat**ot (process) is futtathat, erőforrások szétosztásával, időosztással.
- A folyamaton belül egységnyi végrehajtási szekvenciát **szál**nak (thread) nevezzük
  - Egy folyamaton belül több szál is lehet
  - A szálak ugyanazt a memóriaterületet használják mint a folyamat amelynek részei
  - A szálak külön műveletvégzőt, magot használhatnak, de külön CPUt nem
  - Egyidejűleg annyi szál futtatható, ahány logikai processzor van
  - Ha több szál van, akkor ütemezéstől függően várakoznak / váltakoznak

Terminológia

- Ha a feladatok jól elválnak, és kevés kommunikáció szükséges a részek között, akkor a **multiprocessing** jó irány
  - Több elkülönülő folyamat, saját memóriaterületekkel
  - A folyamat létrehozása erőforrás- és időigényes
  - Több processzoros rendszerek esetén az egyetlen út
  - Egy CPUnál is ez történik, az operációs rendszer ütemez
- Több maggal / műveletvégzővel rendelkező, egy processzoros rendszereknél alkalmazható a **multithreading**
  - Egy folyamaton belül több szál, a szálak konkurrens végrehajtása
  - A szál létrehozása is időigényes, de gazdaságosabb
  - A szál másik magot használhat, hyperthreading is lehetséges

Utasítások végrehajtási sorrendje

- Ha a szálak nem használnak közös változókat
  - Egyszerű eset, nincs logikailag nehéz lépés
  - Ha ugyanazt csinálják a szálak, de különböző adatokon: adatpárhuzamosság
- Ha a szálak közös változó(ka)t használnak
  - Nem mindegy a végrehajtási sorrend
  - Átlapolás (interleaving) gondot okozhat

```
A = B = 0

R1 = A          R2 = B
B = 1           A = 2
```

Mi R1 és R2 értéke?

Kozsik Tamás (ELTE-IK) Java Concurrency előadása alapján

Versenyhelyzet példa

- Filmet szeretnél nézni a moziban
- A pénztárnál megkérdezed, hogy van-e szabad hely
- Azt a választ kapod, hogy igen, van
- Elmész a büfébe, néhány perc múlva visszatérsz, és jegyet szeretnél venni
- Azt a választ kapod, hogy nincs szabad hely

Versenyhelyzet példa

```
if (SzabadHely() > 0)
   JegyetVesz()
```

```
if (SzabadHely() > 0)
  JegyetVesz()
```

Az ellenőrzés és a cselekvés között eltelt időben a szabad helyek száma változhatott egy másik szál hatására

Összefoglalás

- A párhuzamos programozás az egyetlen megoldás a teljesítmény növelésére
- A párhuzamos program más felépítésű
- Nem minden rész párhuzamosítható
  - Bottleneck
  - Amdahl törvény, Gustafson-törvény
- Hatékonyság
  - Overhead
- Konkurrencia, párhuzamosság, aszinkronitás
- Versenyhelyzet

Források

- Mattson, Sanders, Massingill: Patterns for Parallel Programming, Pearson, 2005
- Clay Breshears: The Art of Concurrency, O'Reilly, 2009
- Vámossy Zoltán, Miklós Árpád, Szénási Sándor: Többszálú/többmagos processzor-architektúrák programozása, Typotex, 2016
- Peter Pacheco: An Introduction to Parallel Programming, Morgan Kaufmann, 2011
- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar: Introduction to Parallel Computing, Addison Wesley, 2003