1. fejezet

Bevezetés

Grafikus Processzorok Tudományos Célú Programozása

Bemutatkozás

Wigner Fizikai Kutatóközpont, Elméleti Osztály

GPU labor:

- Párhuzamos és elosztott számítások
- Numerikus differenciálegyenlet megoldók
- Vizualizáció
- GPU Nap évente, idén várhatóan Június vége.
- Modern Tudományos Programozás Előadássorozat

http://gpu.wigner.mta.hu
gpu (kukac) wigner.mta.hu

Az előadással kapcsolatos információk

Hét	Időpont	Óra
2.	Február 14.	1. Óra
3.	Február 21.	2. Óra
4.	Február 28.	3. Óra
5.	Március 7.	4. Óra
6.	Március 14.	5. Óra
7.	Március 21.	6. Óra
8.	Március 28.	7. Óra
9.	Április 4.	8. Óra
10.	Április 11.	9. Óra
11.	Április 18.	Tavaszi szünet Ápr. 12-18.
12.	Április 25.	10. Óra
13.	Május 2.	11. Óra
14.	Május 9.	12. Óra
15.	Május 16.	13. Óra

Az előadással kapcsolatos információk

- Időpont: Kedd 14:15-15:45
- Helyszín: ELTE É-i tömb, -1.64 Rybár István terem

- Weboldal, aktuális információk:
  http://gpu.wigner.mta.hu/index.php?id=GPU_Lecture

- Jegyszerzés:
  Programozási beadandó feladatok + szóbeli vizsga

  \+ pont a jó kérdésekért! ☺

Az előadással kapcsolatos információk

Tematika:

- Modern hardverek működése, felépítése
- Assembly létminimum
- C++ példákon keresztül az alapoktól az új standardokig
- Párhuzamosítás C++-ban
- GPU API-k és használatuk
- Párhuzamos szimulációk és esettanulmányok

Bevezetés

Bevezetés

Miről lesz szó?

- Hardverek alapvető működése
- Programozási nyelvek alapvető építőkövei
- Párhuzamosság elemei

- C++ (11, 14, 17)
- OpenCL (1.2, 2.0)
- SYCL (1.2, 2.2)

- Hogyan írjunk a fentiek segítségével hatékony szimulációkat?

Bevezetés

Wigner GPU Lab

Miért pont ezekről akarunk előadást tartani?

- Kutatók (fizikusok, vegyészek, orvosok…):
  hatalmas kísérletek, bonyolult szimulációk, rengeteg adat
- Nem programozók, nem informatikusok!
- Eredmény: a hardver kihasználatlan
- Következmény:
  egyre gyakoribb, hogy pályázatokat dobnak emiatt vissza
  versenyhátrány a jobb kihasználókkal szemben

Bevezetés

Wigner GPU Lab

Miért pont ezekről akarunk előadást tartani?

A tudományág:

- Szerteágazó
  (sok nyelv, API és platform)
- Gyorsan változik
  (új hw, új standardek, új verziók évente többször is)
- Az információ szétszórt
  (blogok, fórumbejegyzések, prezentációk, youtube előadások)

nincs idő rendszerezni, leírni könyvbe az információkat

Bevezetés: Programozás

Bevezetés: Programozás

- Program:
  utasítások sorozata a gépnek (végső soron)
- Programozás:
  utasítások kiosztása

Bevezetés: Programozás

- Program:
  utasítások sorozata a gépnek (végső soron)
- Programozás:
  utasítások kiosztása …nem egészen…

A tudományág neve:
Programming Language Theory

- Programozási nyelv:
  logikai rendszer, amiben megfogalmazunk egy problémát

- Logikai rendszer? ..ez inkább matematika…
  és ez nem véletlen!

Bevezetés: Programozás

- Típusrendszer (type system):

A programozási nyelvnek azon része, amely a nyelvi elemekhez (kifejezések, változók, függvények) tulajdonságokat (típus) rendel, és az elemek kombinációit ellenőrzi, hogy az előírt szabályoknak megfelelnek-e.

Bevezetés: Programozás

- Példa: nyelvtan

  Ez értelmes:
  „Kati elment a boltba paradicsomért.”

  Ez kevésbé:
  „Bolt paradicsomot Katiért elmenne a a.

  A nyelvtani rendszer által lesznek a mondataink „értelmesek”.

Bevezetés: Programozás

- Típusrendszer (type system):

A típusrendszer garantálja, hogy a benne megfogalmazott program eleget tesz a nyelvi rendszer szabályainak.

Következmény: kevesebb bug.

A szabályok ellenőrzése: *type checking*
A típus rendszerek elméletét vizsgáló tudományág: *type theory*

Bevezetés: Programozás

- Típusrendszer és Logika
  Curry-Howard korrespondencia (dióhéjban):

Logika	Program
Bizonyítás	Program
Tétel	Típus
Hipotézisek	Argumentumok

Bevezetés: Programozás

- Típusrendszer és Logika
  Curry-Howard korrespondencia (dióhéjban):

Logika	Program	C++
Konjunkció	Szorzat típus	struct
Diszjunkció	Összeg típus	union+enum[1]
Implikáció	Függvény típus	R f(A…)
Implikáció bevezetése	Függvény definíció	R f(A a, A b){ return a*b; }
Implikáció eliminálása	Függvényhívás	f(a, b);

[1] Az igazi összegtípus a „tagged union” lenne, ami nem létezik C++-ban.

Bevezetés: Programozás

- Az analógiák itt nem érnek véget:

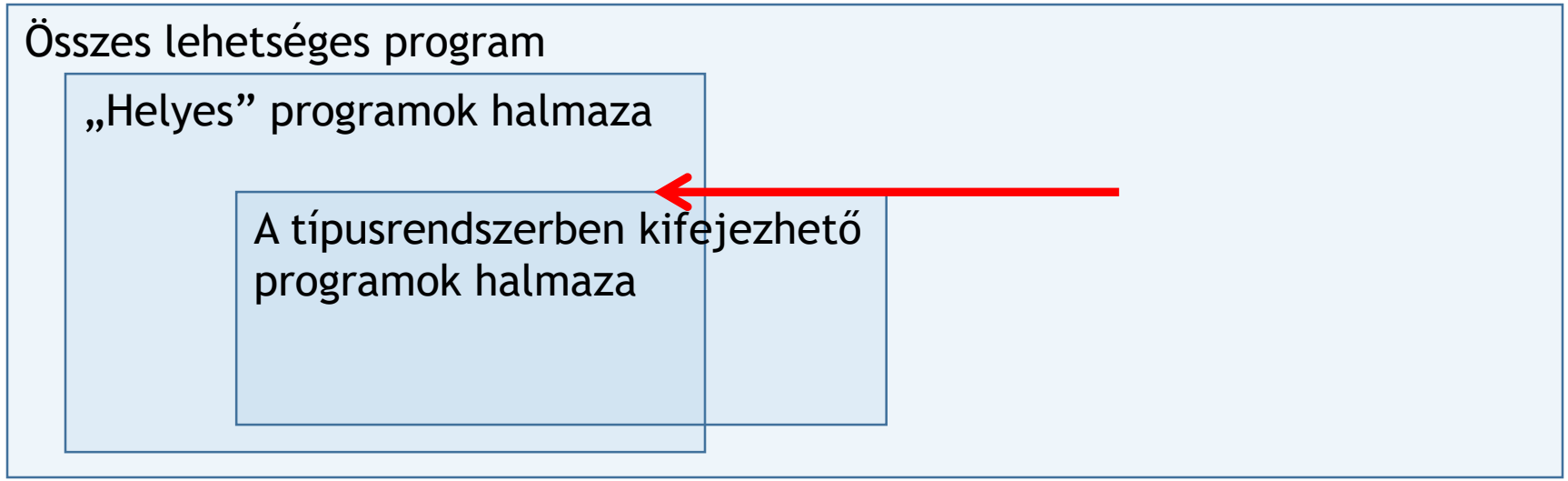
Category Theory	Physics	Topology	Logic	Computation
object X	Hilbert space X	manifold X	proposition X	data type X
morphism $f: X \to Y$	operator $f: X \to Y$	cobordism $f: X \to Y$	proof $f: X \to Y$	program $f: X \to Y$
tensor product of objects: $X \otimes Y$	Hilbert space of joint system: $X \otimes Y$	disjoint union of manifolds: $X \otimes Y$	conjunction of propositions: $X \otimes Y$	product of data types: $X \otimes Y$
tensor product of morphisms: $f \otimes g$	parallel processes: $f \otimes g$	disjoint union of cobordisms: $f \otimes g$	proofs carried out in parallel: $f \otimes g$	programs executing in parallel: $f \otimes g$
internal hom: $X \multimap Y$	Hilbert space of 'anti-X and Y': $X^* \otimes Y$	disjoint union of orientation-reversed X and Y: $X^* \otimes Y$	conditional proposition: $X \multimap Y$	function type: $X \multimap Y$

Table 4: The Rosetta Stone (larger version)

http://arxiv.org/abs/0903.0340

Bevezetés: Típusrendszerek

- A típusrendszer segít kiszűrni a bugokat.
- Jó típus rendszer:
  Minél több rossz programot képes kiszűrni,
  Minél kevesebb helyes programot tesz kifejezhetetlenné.



Bevezetés: Programozási nyelvek

A programozási nyelvek eszköztára és a bennük megvalósított típusrendszer szorosan összefügg.

Egy fejlettebb, magasabb szintű nyelvtől az ember több biztosítékot, garanciát vár el, mint egy alacsonyabbszintűtől.

Bevezetés: Programozási nyelvek

- Gépi kód:
  azok az utasítások, amit a hardver végre tud hajtani.

  Ha hibás utasításkódot adunk meg, a hw megáll, „halt” állapotba kerül, stb.

  Semmilyen más ellenőrzés nincs.

Bevezetés: Programozási nyelvek

- Assembly:
  A gépi kódú utasításokat pár betűs rövidítések jelölik.

  Az utasítások argumentumainak számát és méretét ellenőrzi az assembler.

  De simán összeszorozhatunk egy karaktersorozatot és egy lebegőpontos számot úgy, mint két egész számot…

Bevezetés: Programozási nyelvek

- C:
  A változók típusai ellenőrizve vannak a műveleteken keresztül, de:
  overflow-k, undefined behavior, …

  A magasabb szintű absztrakciókat csak macro-k segítségével lehet megvalósítani, amiken nincs semmilyen típusellenőrzés

Bevezetés: Programozási nyelvek

- C++:
  C-ből sok rossz örökség (de javul)

  A class rendszer számos absztrakciót összemos

  A magasabb szintű absztrakciókat a templatek segítik.
  De nincs memória biztonság, magasabb szintű kvantifikáció, stb.

Bevezetés: Programozási nyelvek

- Haskell:
  Garantálja a memóriakezelés helyességét

  A típusrendszer letisztult, a fordító több, magasabb szintű ellenőrzést tud megtenni, erősebb invariánsokat lehet bevezetni és garantálni

Bevezetés: Programozási nyelvek

- Agda, Idris, Coq, Isabelle:

  Dependens típusok, azaz a futásidejű értékek is visszahathatnak a típusokra!

  Matematikai bizonyítások

Bevezetés: elkerülhető hibák

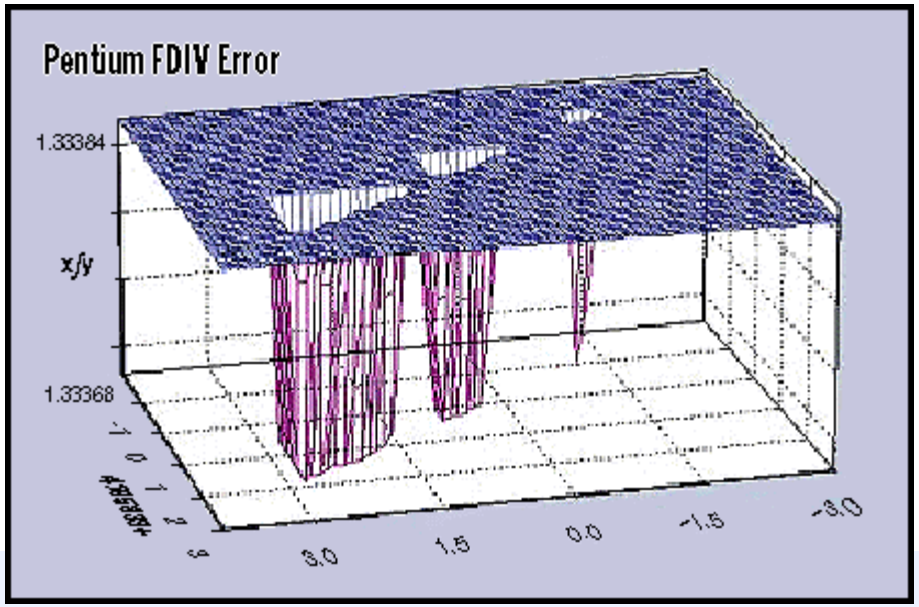
A megfelelő nyelvekkel és típusrendszerekkel számos, esetenként katasztrofális hiba is megelőzhető lett volna.

Bevezetés: elkerülhető hibák

- Intel Pentium bug (1994):

  FDIV (lebegőpontos osztás)
  Néhány speciális esetben az osztáshoz
  tartozó look-up table nem tartalmazott értékeket.

Pl.: 4195835/3145727



A presztízsveszteség
475M USD-be került
az Intelnek…

Másik hasonló:
Pentium F00F bug

Bevezetés: elkerülhető hibák

- ESA Ariane 5 (1996):
  64-bit float -> 16-bit int conversion overflow

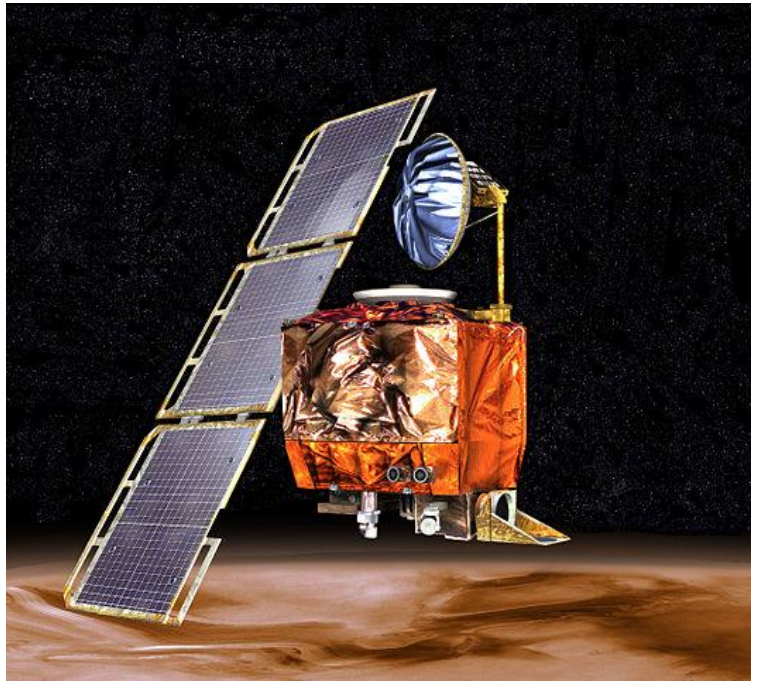
- 370M USD

Bevezetés: elkerülhető hibák

- NASA Mars Climate Orbiter (1999)

  metric-imperial eltérés, két szoftver rész között.

- 327M USD

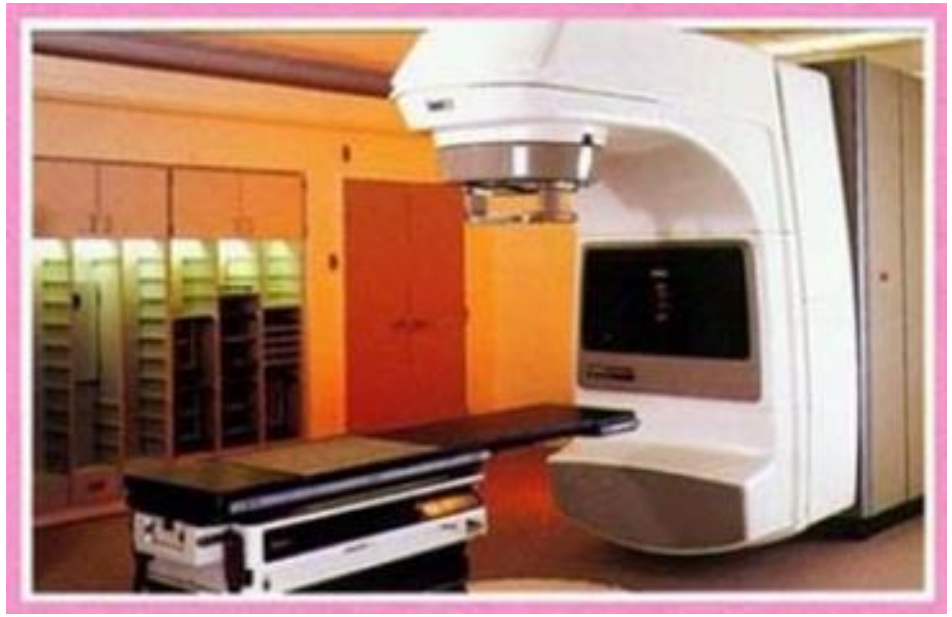


Bevezetés: elkerülhető hibák

- Therac-25 sugárterápiás készülék (1986-87)

  race condition (bővebben itt, itt)

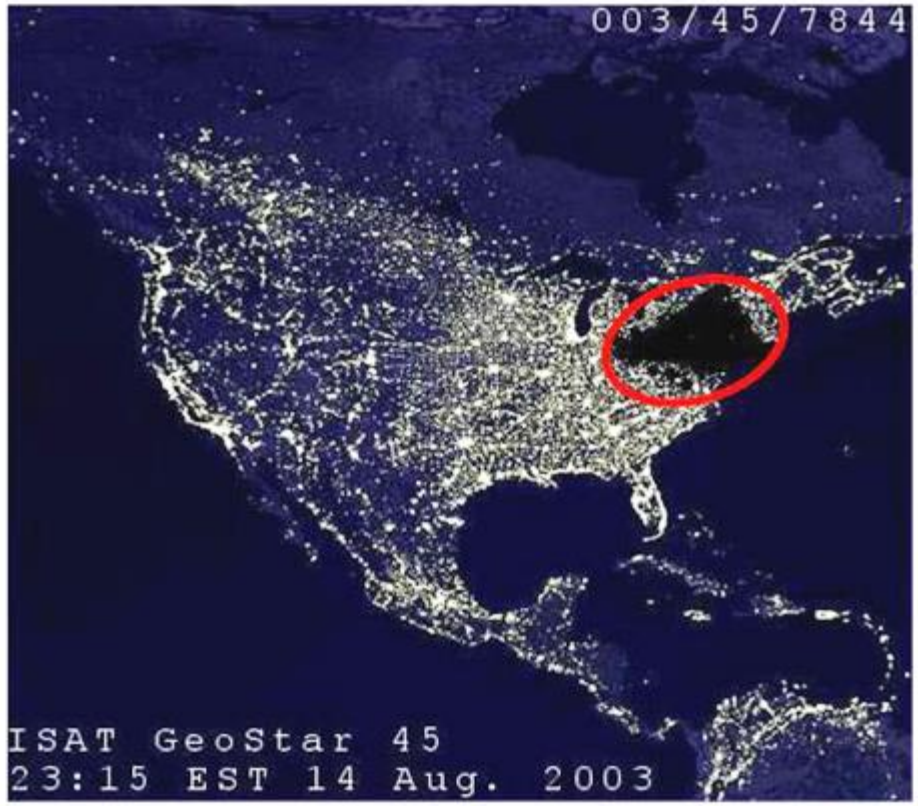
  legalább 6 esetben súlyos sugárterhelés a betegekre…



Bevezetés: elkerülhető hibák

- 2003-as áramszünet az Észak USA-ban és Kanadában, 55 millió embert érintett

- A vezérlőközpont nem értesült időben a probléma mértékéről, mert egy race-condition miatt nem működött a riasztó rendszer



Bevezetés: elkerülhető hibák

- A fentiek elkerülhetőek ma már megfelelő nyelvek és típusrendszerek használatával.

  Pl. a hardvereket már automatikus proof-assistantokkal ellenőrzik, hogy bizonyítottan helyes és teljes legyen a működésük

- A mindennapi szimulációknak látszólag nincs szükségük ezekre.

Bevezetés: szimulációk

Szimulációk:

Kompromisszumok:

- fejlesztési idő vs.
- hibák rizikója vs.
- Teljesítmény (hardver kihasználtság) vs.
- karbantarthatóság (maintainability)

Nem mindegy, hogy mennyit vesztünk a kompromisszumokkal.
Sok múlik a nyelven / típusrendszeren.

Bevezetés: szimulációk

Szimulációk:

Absztrakció: ne írjuk meg ugyanazt újra, paraméterezzük a megoldásokat.
(De ≠ deduplikáció, macro-k)

Magasabb szint: több információ a problémáról, könnyebb optimalizálni.

Bevezető példa itt.

Bevezetés: szimulációk

Ne írjuk meg kétszer a hasonló dolgokat…

Lineáris algebra

Numerikus Módszerek

Különböző hardver (GPU/CPU)

Különböző párhuzamomsítás

Különböző API-k

Bevezetés: szimulációk

Párhuzamosítás

A megfelelő absztrakciókban gondolkozva egyszerűbb érvelni a párhuzamos feldolgozás helyessége mellett.

Pl. rust concurrency 1. 2.

Bevezetés: Sapir–Whorf hipotézis

- Másnéven: nyelvi relativitás

  A (beszélt/írott) nyelvi struktúrák befolyással vannak a világról alkotott képünkre.

  A nyelvi kategóriák limitálják, meghatározzák a kognitív (gondolati) kategóriákat.

  A beszámolók alapján ez a programozási nyelvekre is igaz…

Bevezetés: blub paradoxon

- Paul Graham: the blub paradox

  Tegyük fel, hogy van egy „számegyenese” a programozási nyelveknek.

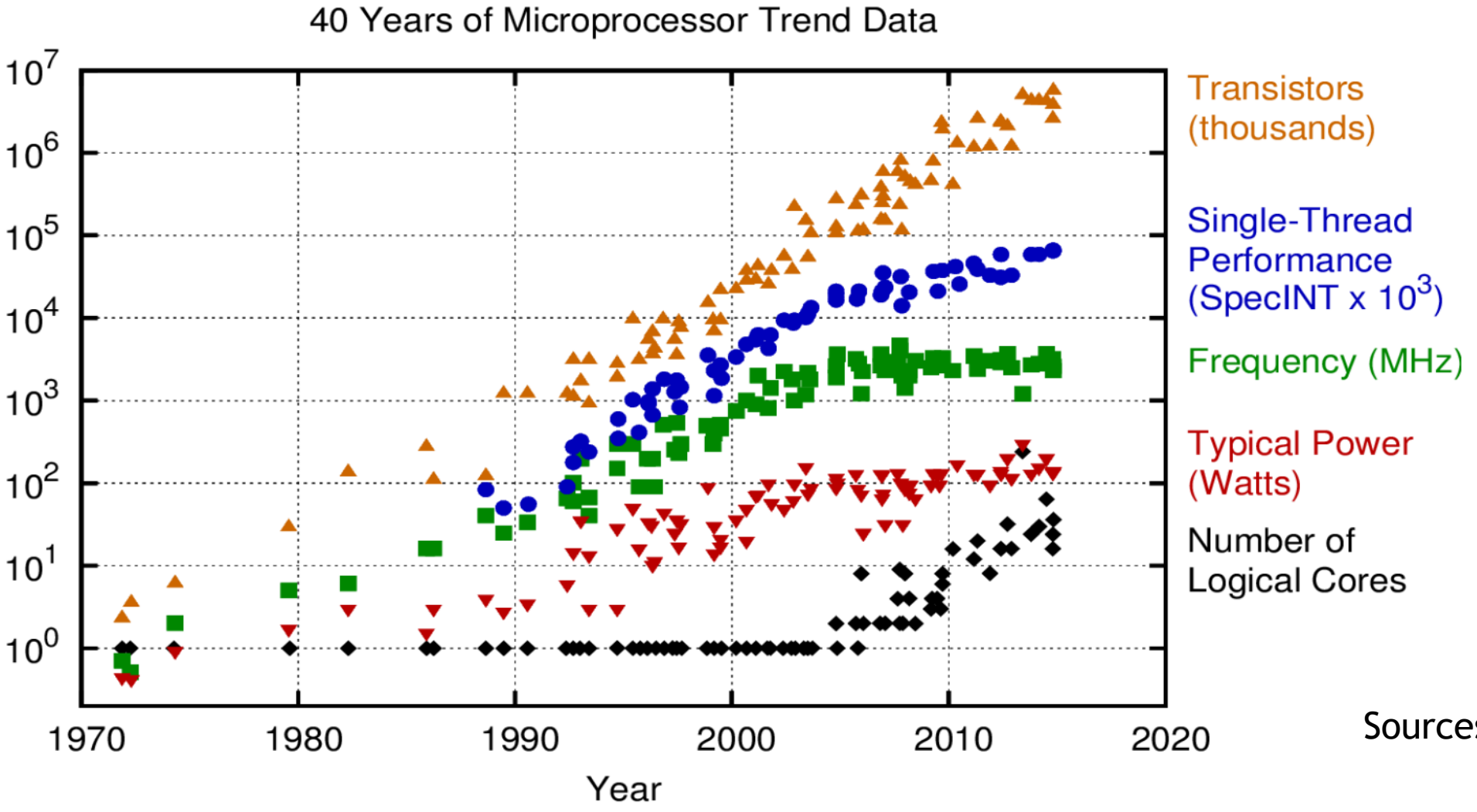
  Ha egy tetszőleges ponton levő nyelvet ismerő programozó „lefelé” néz, tudja, hogy azok „kevésbé jó” nyelvek

  De amikor „felfelé” néz, akkor csak furcsa dolgokat lát, amik szerinte szükségtelenek

Bevezetés: párhuzamosság

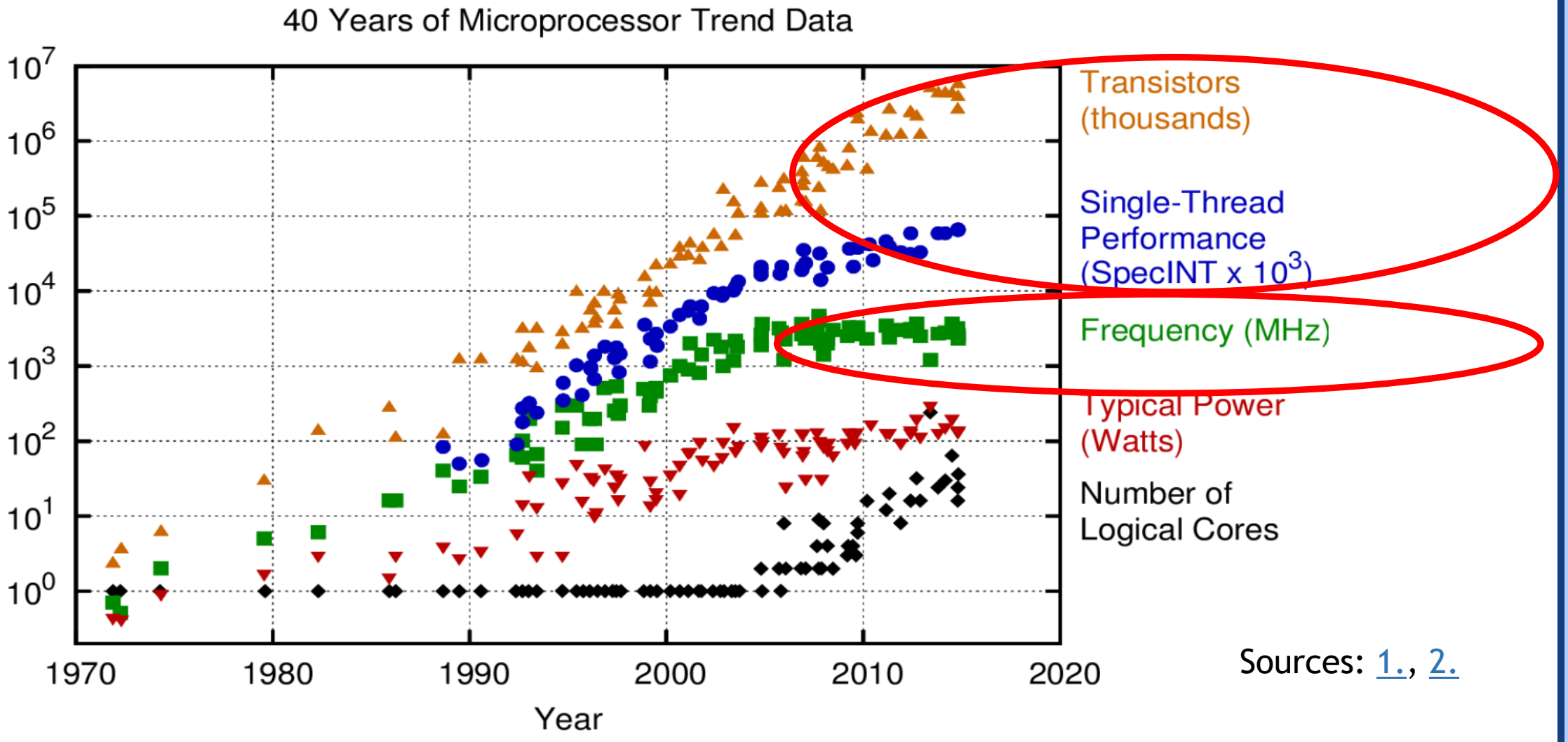
- Miért kell nekünk párhuzamosság?

Bevezetés: párhuzamosság



Sources: 1., 2.

Bevezetés: párhuzamosság



Sources: 1., 2.

Bevezetés: miért C++?

Nyelvválasztás = kompromisszum

- Nem fizetünk azért a nyelvi szolgáltatásért, amit nem használunk
- Egyszerre kellően alacsonyszintű, hogy hatékony programokat lehessen benne írni, de
- Meglepően magasszintű absztrakciók is építhetőek benne
- Dinamikusan fejlődő, az egész ipar által támogatott nyelv

Bevezetés: miért OpenCL / SYCL?

- Bizonyítottan a fontos a hordozhatóság
  Számos esetben demonstrálódott, hogy az egyetlen megoldáshoz kötődés csapdahelyzetet teremthet
- Széles ipari összefogás áll ezeknek a fejlesztése mögött is (Khronos)
- Ezekben is le lehet menni kellően alacsony szintre, hogy maximálisan kihasználjuk a hardver nyújtotta képességeket