Párhuzamosítás

Pál Balázs

Eötvös Loránd Tudományegyetem

Korszerű szám. módszerek a fizikában 2.
2021. november

Tematika

Kérdések

- „Hogy tudom lefuttatni ezt a kódot párhuzamosan?”
- „Mit kell beírjak Pythonban, hogy párhuzamosan fusson le a kód?”
- „Jupyter Notebookot tudok párhuzamosan futtatni valahogy?”
- „A Python kód párhuzamosan fut, ha lefuttatom?”

⋮

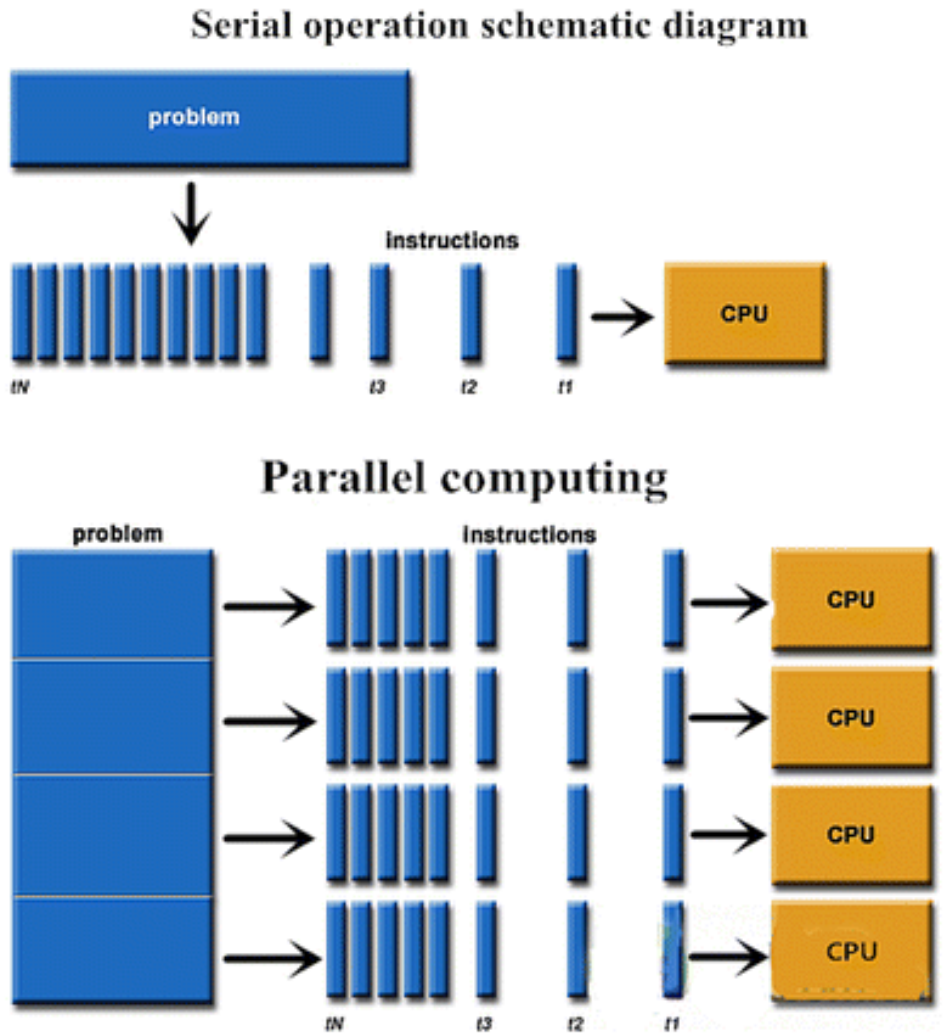
Válaszok

- Mennyi minden húzódik a „párhuzamosítás” mögött?
- Miért és mikor használjuk egyáltalán?
- Milyen esetekben és hogyan tudja egy fizikus is használni?

Áttekintés

Fontos kifejezések

- Alapfogalmak
  - Szál („Thread”)
  - Folyamat („Process”)
  - Mag („Core”)
- Számítási módszerek
  - Soros („Serial”/„Sequential”)
  - Párhuzamos („Parallel”)
- Párhuzamosítási módszerek
  - „Multi-threading”
  - „Multi-processing”
  - ⋮



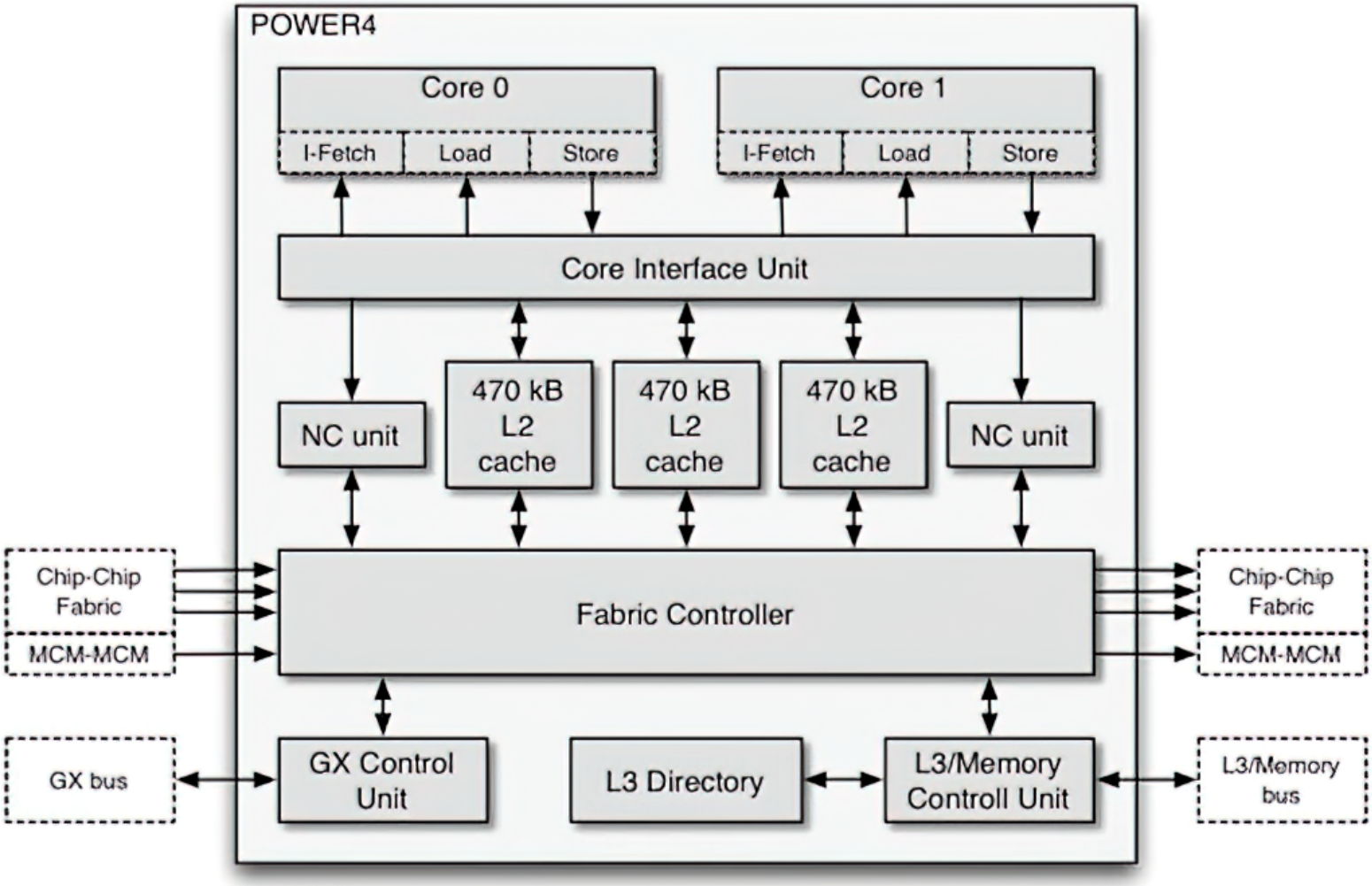
Forrás: ResearchGate

Szálak, folyamatok, magok, izék...

CPU („Central Processing Unit”) és magok

- „A számítógép agya”, ez felelős
  - az aritmetikai műveletek számításáért,
  - a logikai műveletek elvégzéséért,
  - a többi hardverkomponens működtetéséért.
- Ma már mindegyik több-magos („multi-core”)
  - Első több(két)-magos modell: POWER4 (IBM, 2001)

Szálak, folyamatok, magok, izék...



Forrás: ibm.com

Szálak, folyamatok, magok, izék...

CPU („Central Processing Unit”) és magok

- „A számítógép agya”, ez felelős
  - az aritmetikai műveletek számításáért,
  - a logikai műveletek elvégzéséért,
  - a többi hardverkomponens működtetéséért.
- Ma már mindegyik több-magos („multi-core”)
  - Első több(két)-magos modell: POWER4 (IBM, 2001)

Folyamatok, szálak

- Folyamat
  - Egy futó program elnevezése
- Szál
  - Egy futó műveletsor elnevezése
  - Egy folyamatot több szálra tudunk bontani
  - Mindig „párhuzamosan” futnak

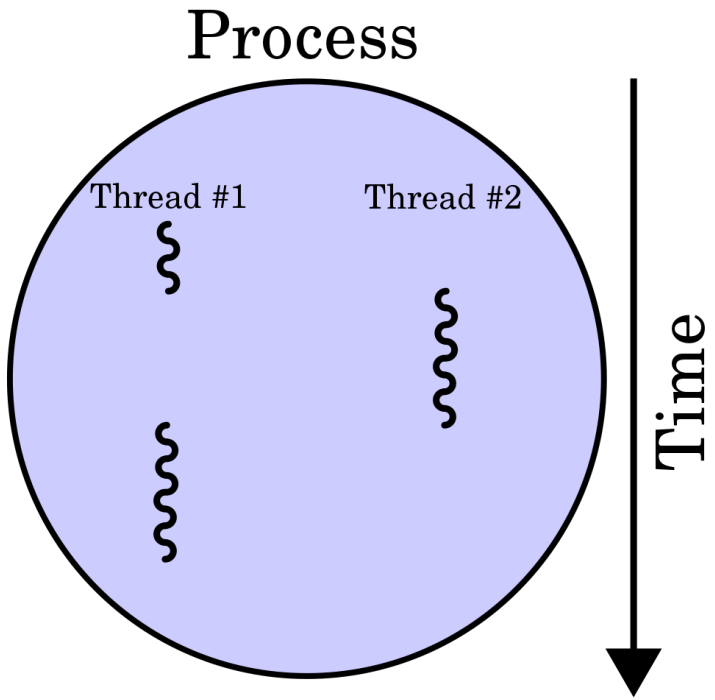
Szálak futása

Szálak kezelése

1. Programnyelv szintjén
   - A felhasználó dönti el, hogy a program melyik részét és hogyan bontja szálakra.
2. Operációs rendszer szintjén
   - Az OS szálkezelője dönti el a szálak futtatásának ütemét

Párhuzamos futás

- Több mag, mint szál esetén ténylegesen párhuzamosan futnak
- Több szál, mint mag esetén a CPU periodikusan váltogat közöttük



Forrás: Cburnett, Wikipedia

Példa párhuzamosításra

Kozmológiai N-test szimuláció

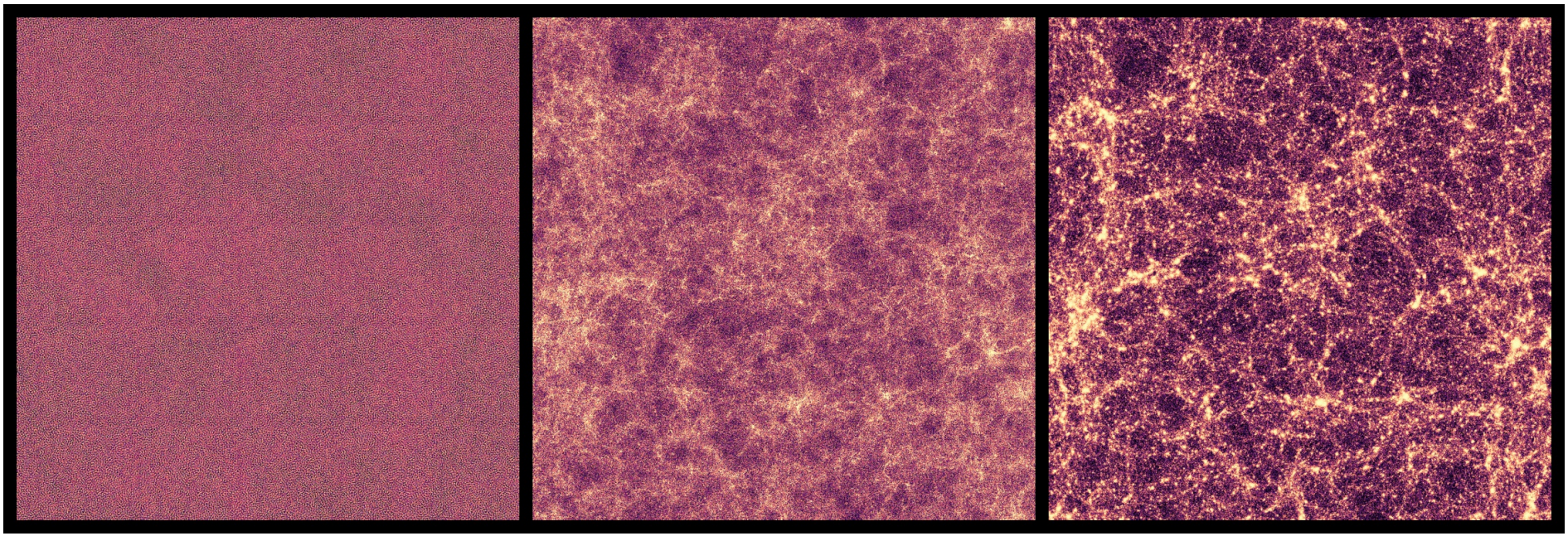
Rengeteg hasonló számítás...

$$\Psi(\boldsymbol{q},\tau) = \int \frac{\mathrm{d}^3 k}{(2\pi)^3} e^{i\boldsymbol{k}\cdot\boldsymbol{q}} \frac{i\boldsymbol{k}}{k^2} \delta_L(\boldsymbol{k})$$

$$\boldsymbol{g}_i = -\nabla\varphi(\boldsymbol{x}_i) = G\sum_{\substack{j=1\\ j\neq i}}^{N} m_j \mathcal{F}(\boldsymbol{x}_i - \boldsymbol{x}_j)$$

$$\boldsymbol{x} = \boldsymbol{q} + \Psi(\boldsymbol{q},\tau)$$

$$\dot{\boldsymbol{x}} = \frac{\dot{D}(a)}{D(1)} \Psi(\boldsymbol{q},\tau)$$



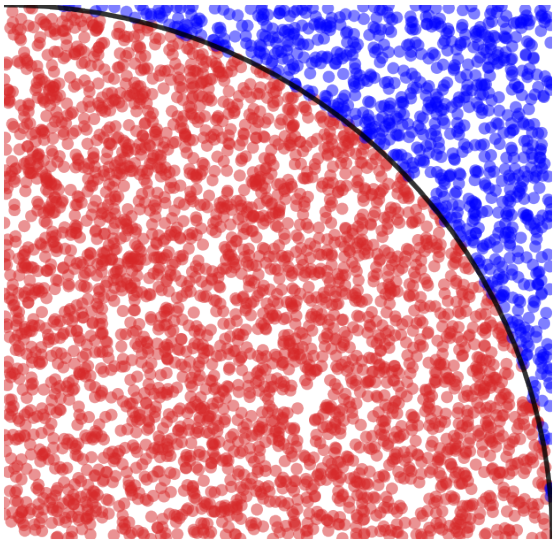
De még sok más is...

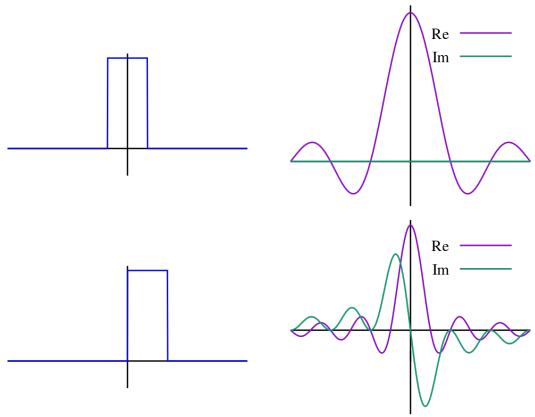
Példák párhuzamosítható problémákra

- Monte–Carlo-szimulációk
- Numerikus integrálás
- Machine learning algoritmusok (pl. random forest)

  ⋮
- Számítógépes vizualizációk és modellezés
- Képfeldolgozási problémák
- Diszkrét Fourier-transzformáció

  ⋮





Race condition

Párhuzamosítás a gyakorlatban

Több eszköz is nagyban megkönnyíti már párhuzamos kódok programozását:

- Python: `threading`, `multiprocessing` stb.
- C++17: Standard könyvtárak
- C/C++/Fortran: `OpenMP`, `CUDA`, stb.

Race condition

Párhuzamosítás a gyakorlatban

Több eszköz is nagyban megkönnyíti már párhuzamos kódok programozását:

- Python: `threading`, `multiprocessing` stb.
- C++17: Standard könyvtárak
- C/C++/Fortran: `OpenMP`, `CUDA`, stb.

Megoldandó problémák

- Rengeteg lenne a fenti eszközök nélkül, de még így is vannak...
- Legfontosabb talán a „race condition”
  - Két szál azonos memóriaterülethez akar egyszerre hozzábabrálni
  - Egyik tipikus esete a rettegett „undefined behaviour”-nek
  - Kritikus hiba, ami teljesen el tudja rontani bármilyen szoftver működését

Race condition

Egyszerű (bár bugyuta) példa

race-condition-serial.cpp

```cpp
int addition(int &x) {

  int new_x = x + 1;

  return new_x;
}

int main(int argc, char const *argv[])
{
  // Starting number
  int x = 0;

  // Adding +1 to it 4 times
  for(int i = 0; i < 4; i++) {
    x = addition(x);
    std::cout << "Current value of `x` is " << x << std::endl;
  }

  return 0;
}
```

```
└─$ ./race_condition_serial
Current value of `x` is 1
Current value of `x` is 2
Current value of `x` is 3
Current value of `x` is 4
```

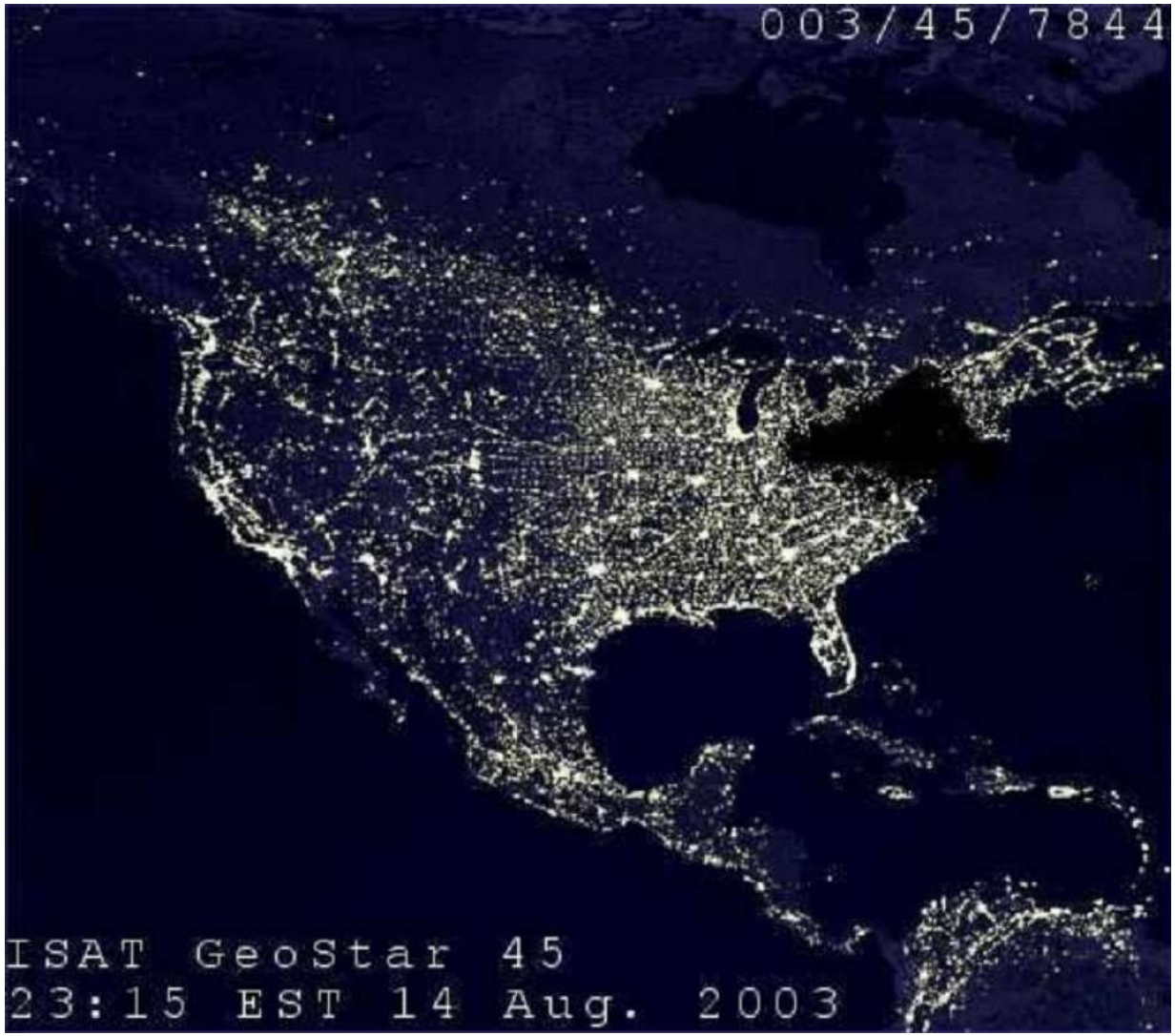
race-condition-parallel.cpp

```cpp
int addition(int x)
{
  int new_x = x + 1;

  return new_x;
}

int main(int argc, char const *argv[])
{
  // Starting number
  int x = 0;

  // Parallelization
  std::vector<std::future<int>> future_vec;

  // Adding +1 to it 4 times
  for(int i = 0; i < 4; i++)
  {
    future_vec.push_back(std::async(std::launch::async, addition, x));
  }

  for(int i = 0; i < 4; i++)
  {
    auto new_x = future_vec[i].get();
    std::cout << "Current value of `x` is " << new_x << std::endl;
  }

  return 0;
}
```

```
└─$ ./race_condition_parallel
Current value of `x` is 1
Current value of `x` is 1
Current value of `x` is 1
Current value of `x` is 1
```

Race condition

2003-as észak-amerikai áramszünet



Forrás: clevescene.com

Amdahl-törvénye

Motiváció

Mennyi időt nyerhetünk valós esetekben a párhuzamosítással?

Amdahl-törvénye

Motiváció

Mennyi időt nyerhetünk valós esetekben a párhuzamosítással?

- Egy program futásideje kifejezhető az alábbi módon:

$$T = T \cdot S + T \cdot P$$

ahol $S + P = 1$ és melyek rendre egy algoritmus kizárólag sorosan futtatható, valamint párhuzamosítható részeinek arányát jelöli.

Amdahl-törvénye

Motiváció

Mennyi időt nyerhetünk valós esetekben a párhuzamosítással?

- Egy program futásideje kifejezhető az alábbi módon:

$$T = T \cdot S + T \cdot P$$

ahol $S + P = 1$ és melyek rendre egy algoritmus kizárólag sorosan futtatható, valamint párhuzamosítható részeinek arányát jelöli.

- Ha a párhuzamosítható (P) részt több (N) szálra osztjuk szét, akkor a program futásideje lecsökken:

$$T_{új} = T \cdot S + T \cdot \frac{P}{N}$$

Amdahl-törvénye

Motiváció

Mennyi időt nyerhetünk valós esetekben a párhuzamosítással?

- A felgyorsulás (Q) így felírható az alábbi módon:

$$Q(N) = \frac{T}{T_{új}} = \frac{\cancel{T}}{\cancel{T} \cdot S + \cancel{T} \cdot \frac{P}{N}} = \frac{1}{S + \frac{P}{N}}$$

Amdahl-törvénye

Motiváció

Mennyi időt nyerhetünk valós esetekben a párhuzamosítással?

- A felgyorsulás (Q) így felírható az alábbi módon:

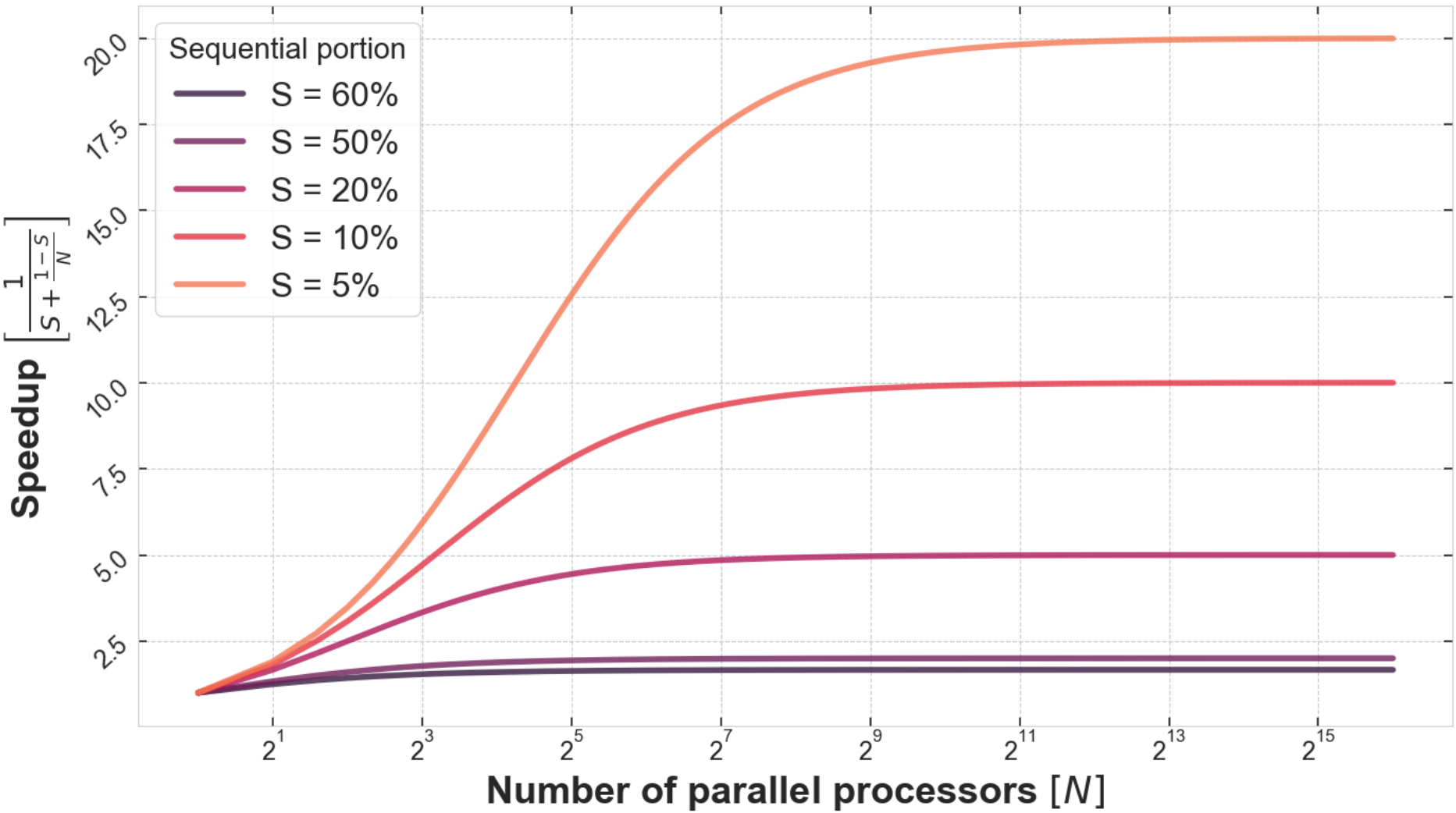
$$Q(N) = \frac{T}{T_{új}} = \frac{\cancel{T}}{\cancel{T} \cdot S + \cancel{T} \cdot \frac{P}{N}} = \frac{1}{S + \frac{P}{N}}$$

- Tovább egyszerűsítve felírhatóvá válik az Amdahl-törvény megszokott alakja:

$$Q(S, N) = \frac{1}{S + \frac{P}{N}} = \frac{1}{S + \frac{1-S}{N}},$$

mely megadja, hogy a felgyorsulás kizárólag a sorosan futtatandó részek arányától és a szálak számától függ.

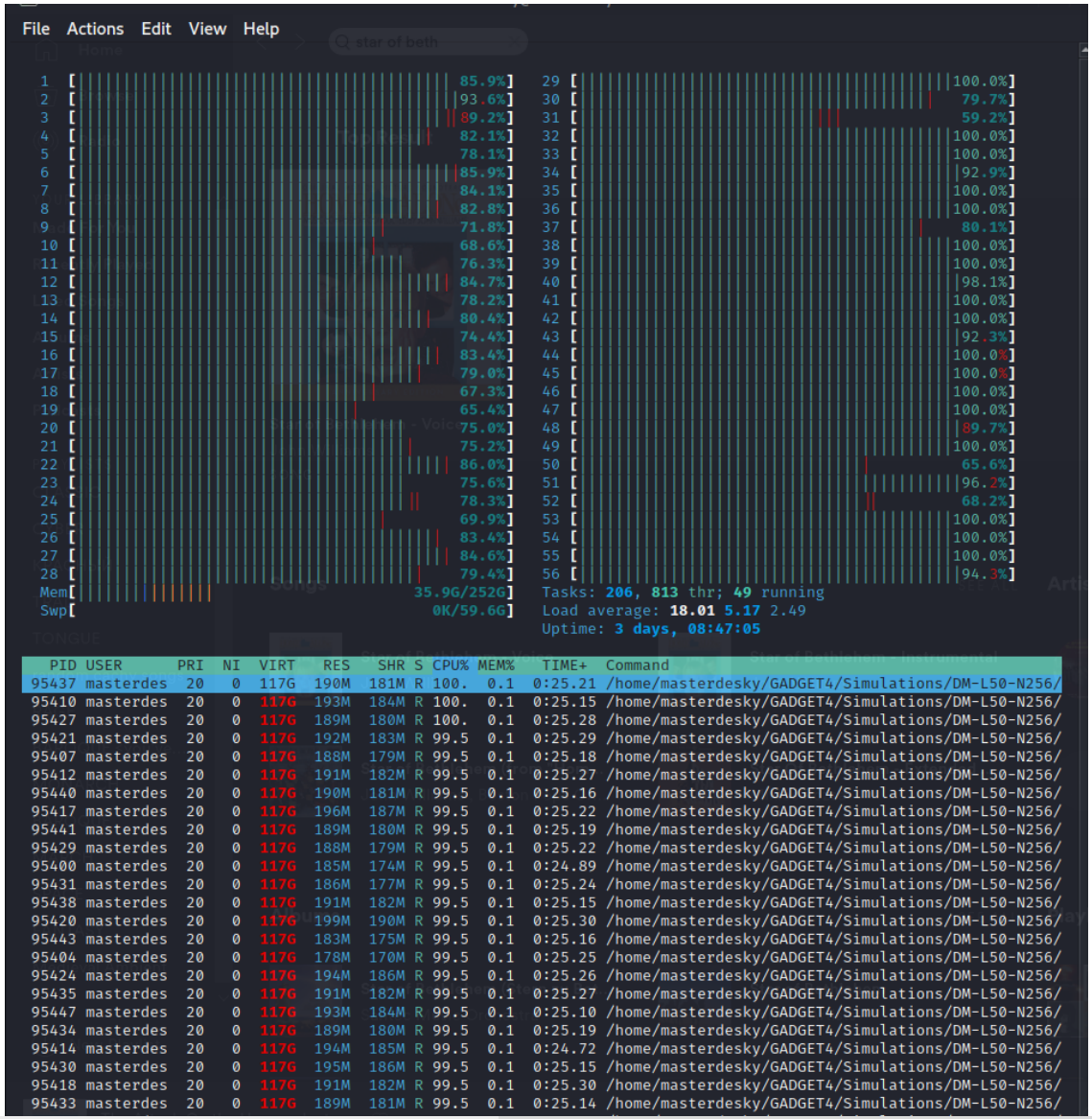
Amdahl-törvénye



Párhuzamosítás fizikában

Általánosságban

- Szimulációs szoftverek futtatása (pl. Szám. szim., MSc – OpenFoam, GADGET4, HOOMD-blue)
- Szervereken történő munka (pl. ELTE-n az onco2, tesla, atys, veo1 (ismertebb nevén ‘kooplex‘), stb.)
- Szerver-klasztereken történő futtatás (pl. ELTE *Atlasz* klaszter)



Párhuzamosítás fizikában

Szerveren és klaszteren

```
#!/bin/bash -l
#SBATCH --job-name="G4M2Norm"
#SBATCH --partition=hpc2019
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=18
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --exclusive

echo
echo "Running on hosts: $SLURM_NODELIST"
echo "Running on $SLURM_NNODES nodes."
echo "Running on $SLURM_NPROCS processors."
echo

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/users/lordpb666/opt/gsl-2.6/lib/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/users/lordpb666/opt/fftw-3.3.9/lib/

mpirun -np $SLURM_NPROCS /users/lordpb666/apps/GADGET4/Gadget4 \
                         /users/lordpb666/apps/GADGET4/Simulations/DM-N65536-M2-L200-GFT5/GADGET4_GTF5.param
```

```
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            314254   hpc2019  7-28-61 hanyeczo  R 23-13:29:58      1 cn18-25
            314255   hpc2019  7-62-95 hanyeczo  R 23-13:29:52      1 cn18-26
            314256   hpc2019 7-96-128 hanyeczo  R 23-13:29:49      1 cn18-07
            314257   hpc2019  14-1-42 hanyeczo  R 23-13:26:30      1 cn18-08
            314258   hpc2019 14-43-84 hanyeczo  R 23-13:25:27      1 cn18-09
            314259   hpc2019 14-85-12 hanyeczo  R 23-13:24:23      1 cn18-10
          316125_8   hpc2019 evo22.sb skiszkao  R 12-10:50:51      1 cn18-24
            325390   hpc2019 atlasz_b geobarbi  R 5-06:35:54      5 cn18-[04,11-12,16,18]
            326521   hpc2019 atlasz_b geobarbi  R 5-12:35:29      1 cn18-06
            326839   hpc2019 atlasz_b geobarbi  R 5-06:07:18      1 cn18-35
            327902   hpc2019 atlasz_b geobarbi  R 4-04:12:25      2 cn18-[05,27]
            327919   hpc2019 atlasz_b geobarbi  R 3-16:49:20      1 cn18-33
            327942   hpc2019 atlasz_b geobarbi  R 3-10:09:39      1 cn18-03
```

Párhuzamosítás fizikában

Pythonban

Előfeltételek

- Pl. a részben már említett `threading`, `multiprocessing`, `joblib`, `subprocess` stb. csomagok
- Több, rendelekzésre álló CPU mag (a Kooplex esetében ez pl. 2 db)

```
_ = Parallel(n_jobs=n_jobs)(delayed(create_frame)(i, P,
                                                  N, n_steps, gl,
                                                  outdir, ffmt, fdpi) for i, gl in enumerate(tqdm(grid_lims)))
```

```
# Takes lot of time to write all files and ping all sites!
for target in df_n['URL']:
    ping_command = 'ping -D -c {0} -i {1} -s {2} {3}'.format(n_packet, interval, packet_size, target)
    output = "{0}{1}.txt".format(data, target)
    with open(output, 'w') as f:
        # Using `Popen` here to run pings "parallel"
        #print('Pinging {}...'.format(target))
        process = subprocess.Popen(ping_command.split(' '), stdout=f)
```