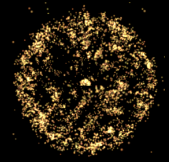
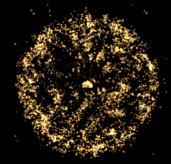


# Grafikus kártyák, mint olcsó szuperszámítógépek - I.

## RMKI GPU nap 2010

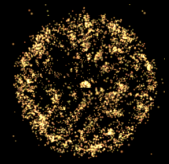
Jurek Zoltán, Tóth Gyula
MTA SZFKI, Röntgendiffrakciós csoport





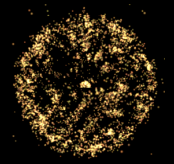
# Vázlat

- Motiváció
- Beüzemelés
- GPU számolás CUDA programozás nélkül, könyvtárak
- (C alapok)
- Hardware adottságok, CUDA programozási modell



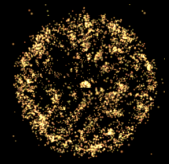
# Irodalom - CUDA

- http://heim.ifi.uio.no/~knutm/geilo2008/seland.pdf
- CUDA, Supercomputing for the Masses Part 1-15
  http://www.drdobbs.com/cpp/207200659
- CUDA official manuals
  Programming Guide, Reference Manual, Release Notes
  http://developer.nvidia.com/object/cuda_2_3_downloads.html
  http://developer.nvidia.com/object/cuda_3_0_downloads.html

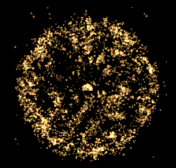


# Irodalom - Egyéb

- Kernighan, Ritchie: The C Programming Language
  http://zanasi.chem.unisa.it/download/C.pdf
  http://www.windowsceportal.hu/download/doc/cpl_hu.zip
- OpenMP tutorial
  https://computing.llnl.gov/tutorials/openMP/
- MPI tutorial
  https://computing.llnl.gov/tutorials/mpi/

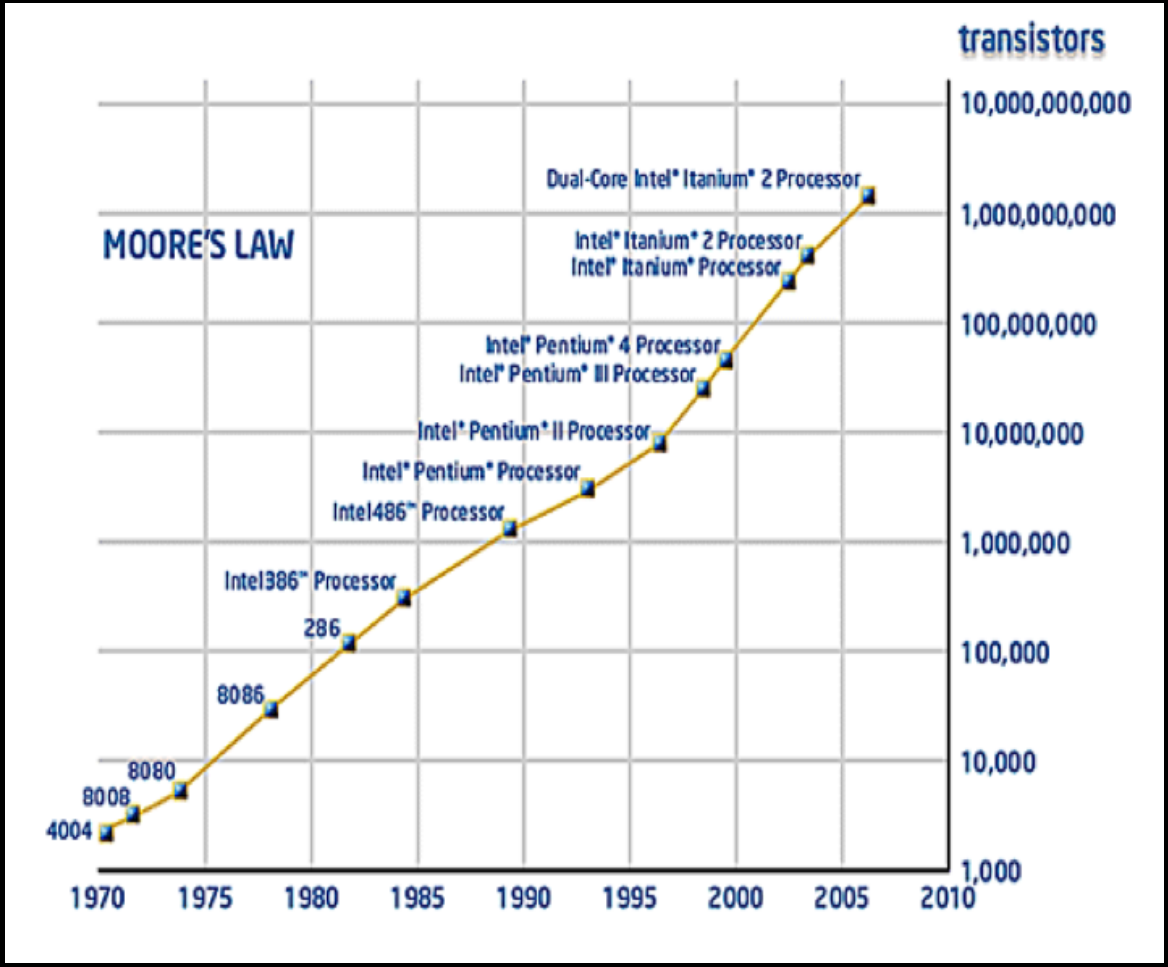


# Motiváció



# Szuperszámítógép GPU-ból

# Moore törvénye



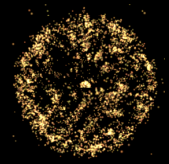

Tranzisztorok száma másfél-két évente duplázódik

Számítógép sebessége exp. növekszik

DE

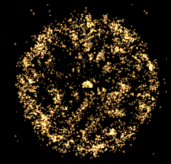
az utóbbi ~4 évben megtorpant!

3GHz, 45nm



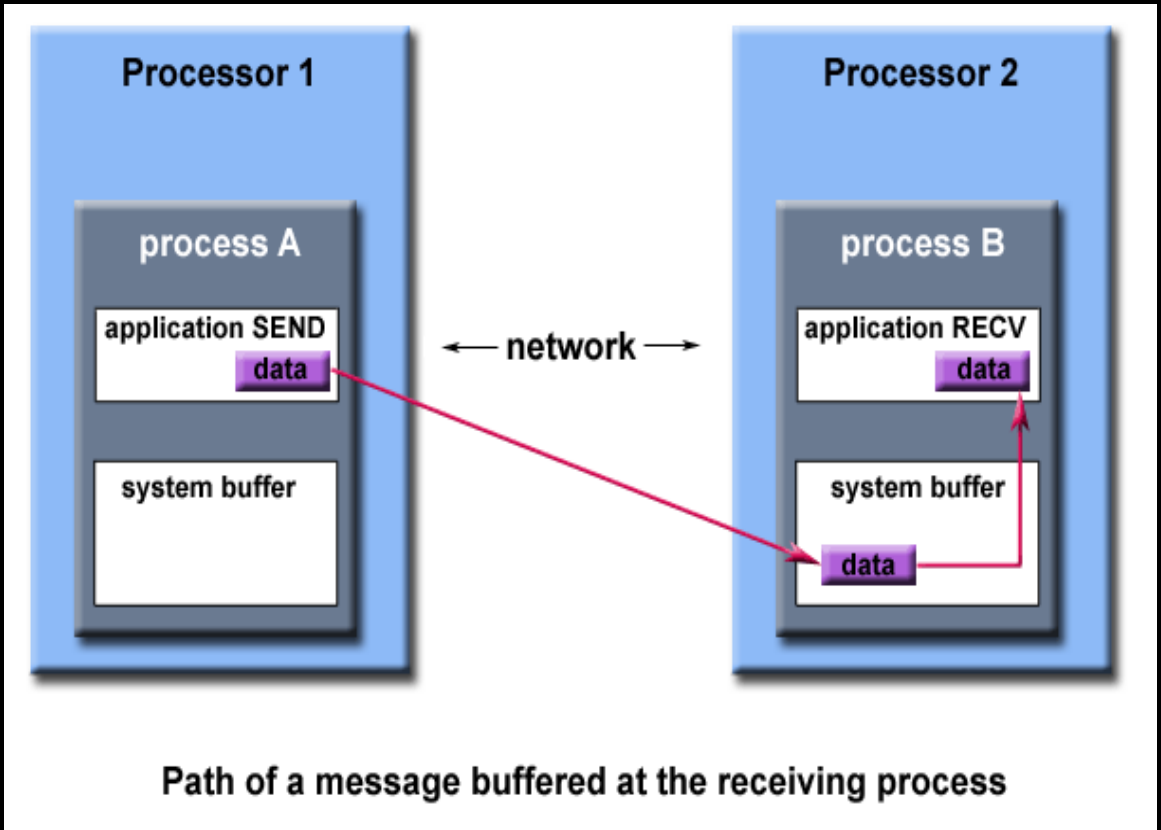
# A párhuzamosítás korszaka!

- Nem a sebesség, hanem a processzorok száma nő
- Át kell gondolni algoritmusaink szerkezetét:
  soros ⟶ párhuzamos



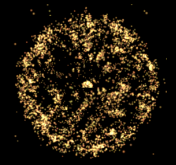
# A párhuzamosítás korszaka!

## MPI (Message Passing Interface) - process szintű




Path of a message buffered at the receiving process

- Tipikus sok gép (klaszter) esetén
- Külön memória a processznek
- Hálózati (TCP/IP) adattranszfer

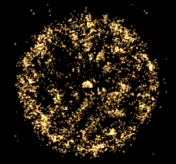


# A párhuzamosítás korszaka!

## MPI (Message Passing Interface) - process szintű

## Egy számoló core

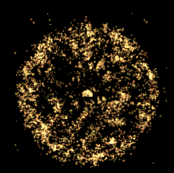
```
top - 11:14:16 up 21 days,  1:22, 84 users,  load average: 2.65, 1.04, 0.41
Tasks: 485 total,   2 running, 483 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.5%us,  0.2%sy,  0.0%ni, 93.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:     47382M total,    41908M used,     5473M free,      566M buffers
Swap:    61443M total,      160M used,    61283M free,    33619M cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9610 jurek     20   0  366m  17m 1456 R  100  0.0   1:43.95 sample.out
 4939 jurek     20   0  620m 102m  16m S    2  0.2 629:06.59 konqueror
  129 root      15  -5     0    0    0 S    1  0.0  68:46.54 kondemand/4
 1379 mt        20   0  8796 1480  856 S    1  0.0  42:57.57 top
 1825 jurek     20   0 1127m 1.1g 879m S    1  2.4  78:27.74 Xvnc
 5195 jurek     20   0  557m 154m  37m S    1  0.3  18:58.77 konqueror
 9544 jurek     20   0  8796 1484  852 R    1  0.0   0:01.35 top
21917 jurek     20   0  8784 1484  844 S    1  0.0 139:14.03 top
 1592 mt        20   0 16344 3648 3248 S    0  0.0  17:20.91 Xvnc
```



# A párhuzamosítás korszaka!

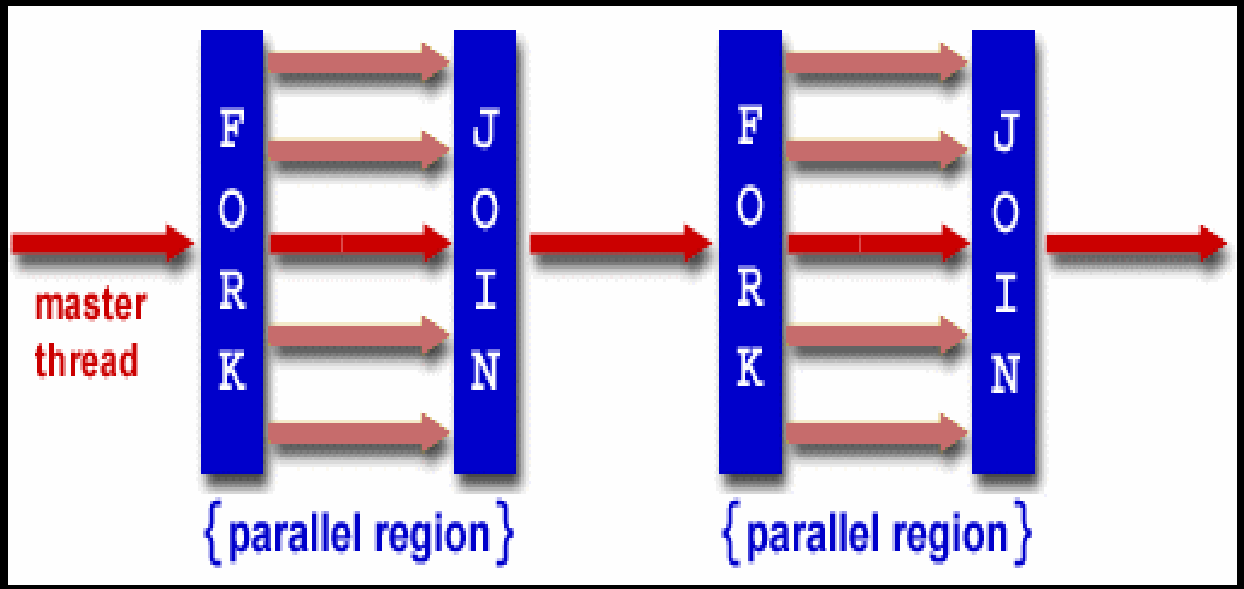
## MPI (Message Passing Interface) - process szintű
## Több számoló core

```
top - 11:13:37 up 21 days,  1:22, 84 users,  load average: 2.76, 0.84, 0.33
Tasks: 488 total,   5 running, 483 sleeping,   0 stopped,   0 zombie
Cpu(s): 25.2%us,  0.3%sy,  0.0%ni, 74.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:     47382M total,    41955M used,     5427M free,      566M buffers
Swap:    61443M total,      160M used,    61283M free,    33619M cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9607 jurek     20   0  366m  16m 1456 R  100  0.0   1:09.28 sample.out
 9609 jurek     20   0  366m  16m 1456 R  100  0.0   1:06.36 sample.out
 9610 jurek     20   0  366m  16m 1456 R  100  0.0   1:04.74 sample.out
 9608 jurek     20   0  366m  16m 1456 R  100  0.0   1:07.75 sample.out
 4939 jurek     20   0  620m 102m  16m S    1  0.2 629:05.79 konqueror
 1379 mt        20   0  8796 1480  856 S    1  0.0  42:57.34 top
 1825 jurek     20   0 1127m 1.1g 879m S    1  2.4  78:27.44 Xvnc
 5195 jurek     20   0  557m 154m  37m S    1  0.3  18:58.52 konqueror
21917 jurek     20   0  8784 1484  844 S    1  0.0 139:13.80 top
```

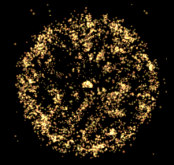


# A párhuzamosítás korszaka!

## OpenMP (Open Multi-Processing) – szál (thread) szintű



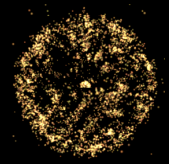

- Egy gép – sok mag esetén
- Szálak: közös memóriaterület + saját változók



# A párhuzamosítás korszaka!

## OpenMP (Open Multi-Processing) – szál (thread) szintű

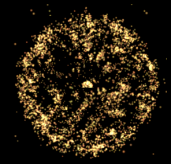
```
top - 11:15:23 up 21 days,  1:23, 84 users,  load average: 3.06, 1.44, 0.59
Tasks: 485 total,   2 running, 483 sleeping,   0 stopped,   0 zombie
Cpu(s): 24.6%us,  0.3%sy,  0.0%ni, 75.1%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:     47382M total,    41913M used,     5469M free,      566M buffers
Swap:    61443M total,      160M used,    61283M free,    33619M cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9646 jurek     20   0  397m  22m 1460 R  400  0.0   2:44.79 sample.out
 4939 jurek     20   0  620m 102m  16m S    2  0.2 629:07.92 konqueror
24131 jurek     20   0  8796 1468  844 S    1  0.0 140:47.02 top
 3633 jurek     20   0  394m  84m  23m S    1  0.2  22:46.83 dolphin
  129 root      15  -5     0    0    0 S    0  0.0  68:46.60 kondemand/4
  439 mt        20   0 3374m 2.2g 258m S    0  4.8 396:12.48 MATLAB
 1379 mt        20   0  8796 1480  856 S    0  0.0  42:57.92 top
 1586 mt        20   0 68788 3696 2224 S    0  0.0  58:02.47 icewm
 1756 gb        20   0 14096 3608 3248 S    0  0.0  17:56.28 Xvnc
```



# Grafikus kártyák

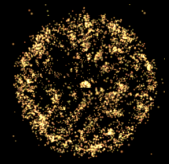
nVidia GTX280 (grafikus kártya):
- 240 db 1.3GHz processzor
- 1000MB memória

Tesla C1060 (számításokhoz):
- 240 db 1.3GHz processzor
- 4000MB memória

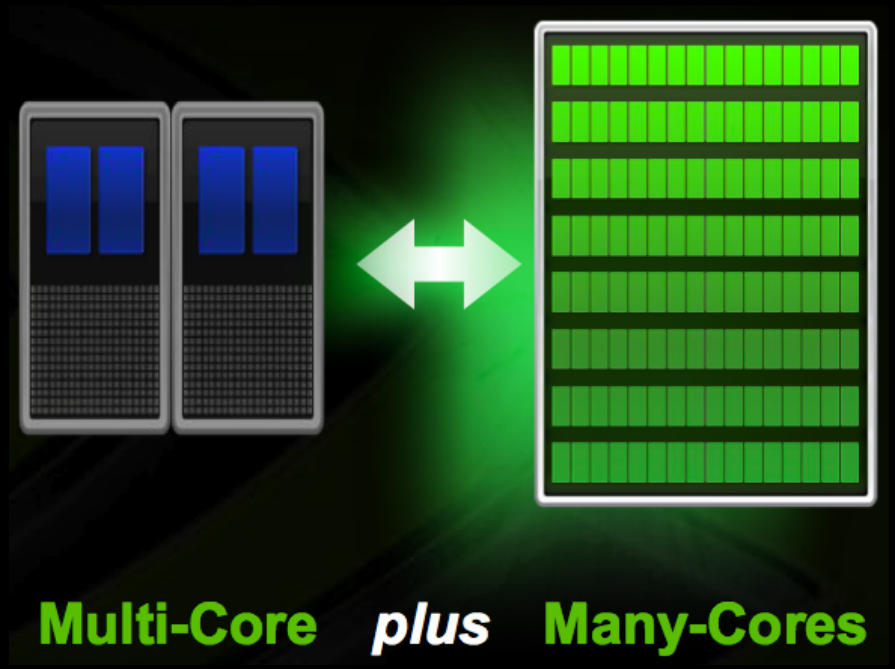


# Grafikus kártyák

- 1920 mag, 7 Tflops (float)
  (FLoating point Operations Per Second)
  100x asztali gép teljesítmény
- „Személyi szuperszámítógép”:
  1 kutató, 1 számítógép
- Elérhető: megfizethető ár
  win/linux/mac + C



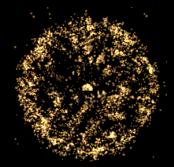
# Heterogén számolások




Host computer + Device

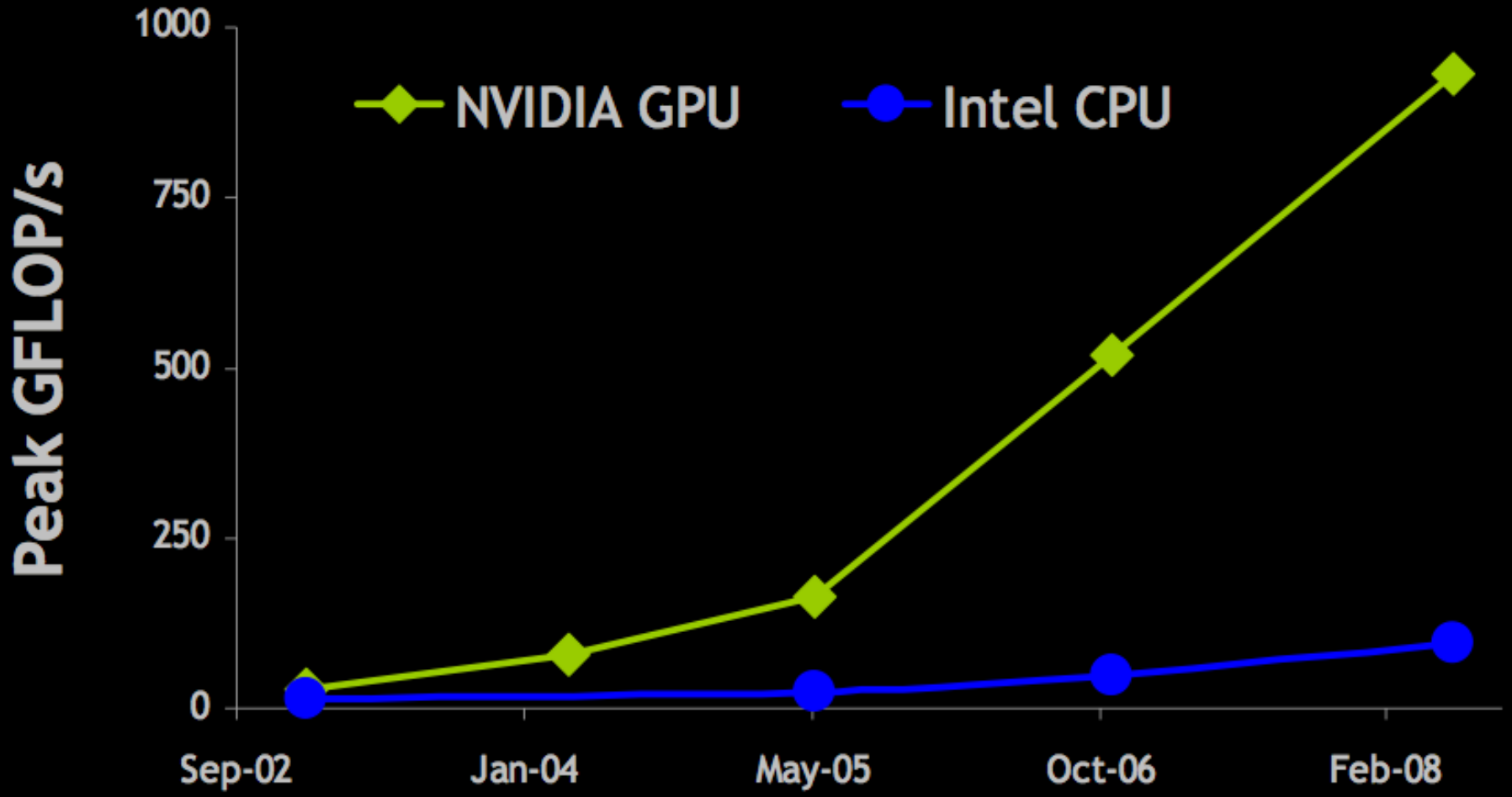
Grafikus kártya
~ coprocesszor:

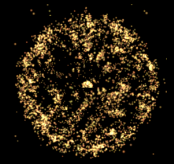
- saját processzorok
- saját memória
- host másol rá adatot
- host indítja a számolást a coprocesszoron
- host visszamásolja az eredményt



# GPU vs. CPU

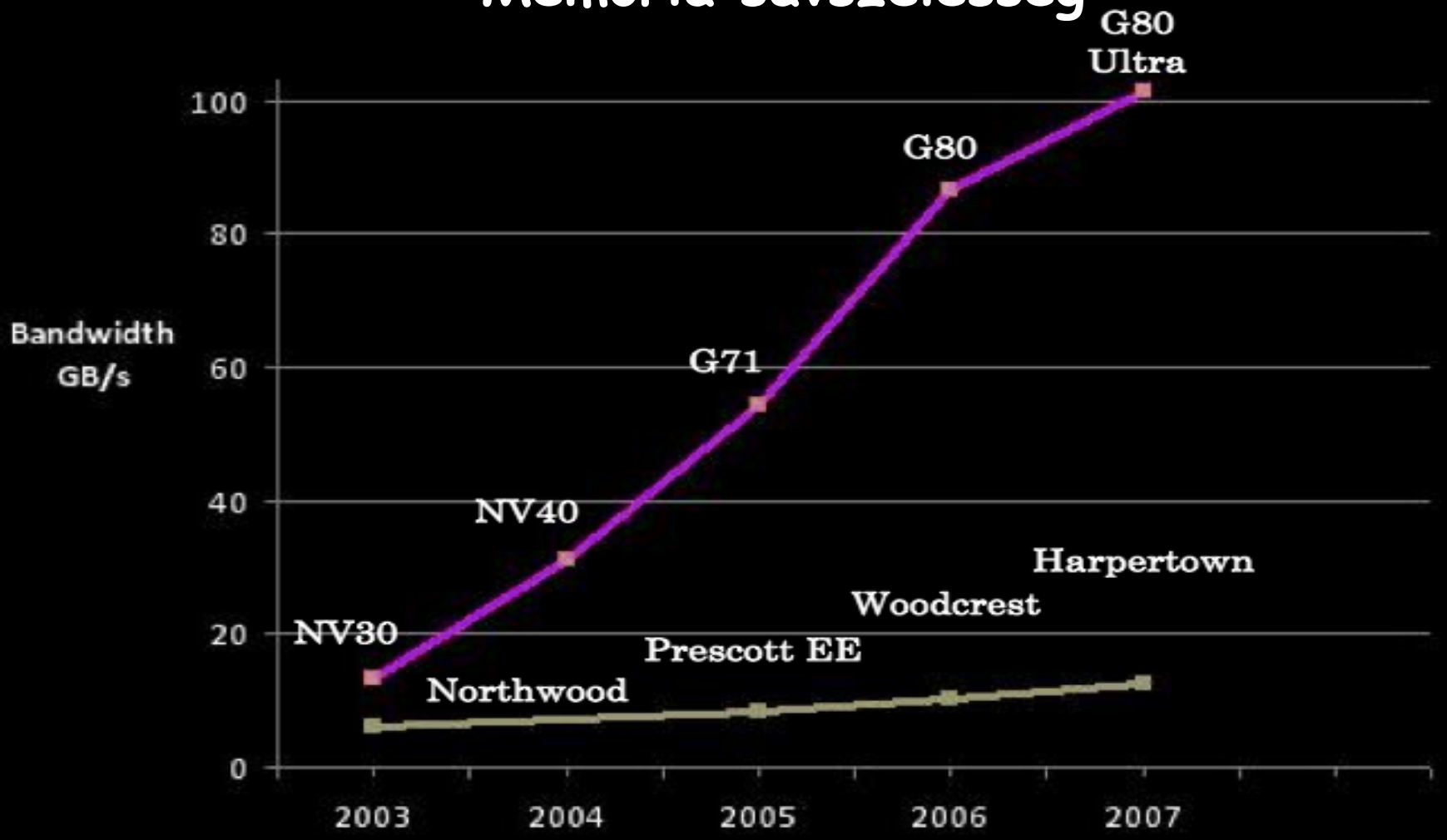
## Számítási teljesítmény





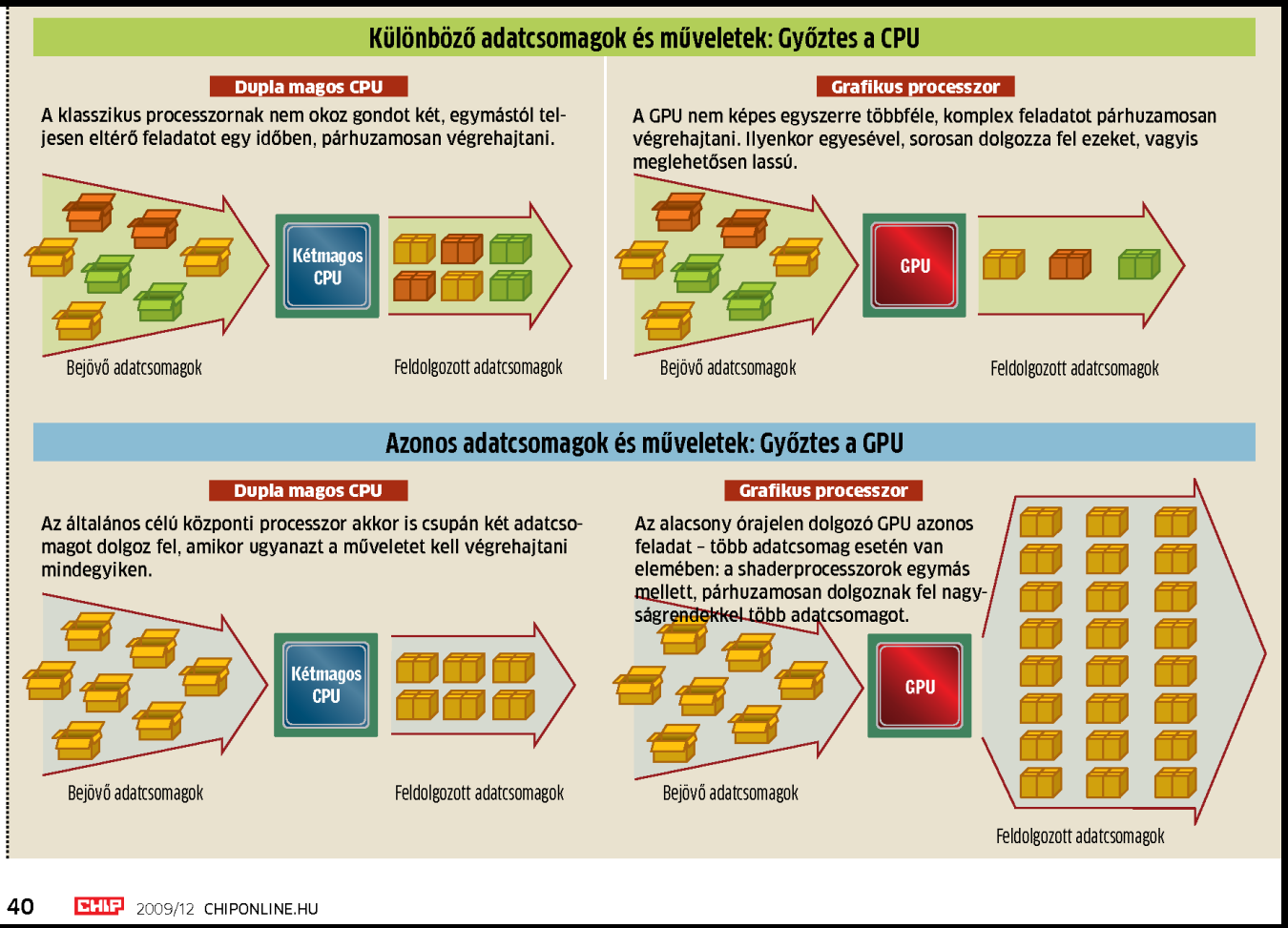
# GPU vs. CPU

## Memória sávszélesség

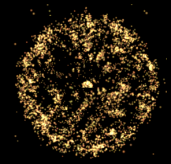


# GPU vs. CPU

## Szál-végrehajtás




- **CPU: MIMD**
  Multiple Instruction, Multiple Data
- **GPU: SIMD/SIMT**
  Single Instr., Multiple Data / Single Instr., Multiple Thread



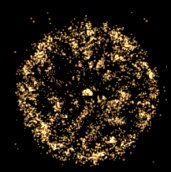
# GPU vs. CPU

A nagy memória (RAM) elérése rendkívül időigényes
(~100 órajel)

↓

gyorsítás szükséges

| CPU | GPU |
| --- | --- |
| ~MB gyorsmemória (cache) a CPU-n | kicsi (~16kB) gyorsmemória de ~128szál/mag hyperthreading |



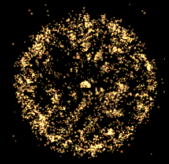
# GPGPU programozás története

- Kezdet: masszív GPU programozás

- Utóbbi években két fejlesztői környezet:
  - nVidia – Compute Unified Device Architecture, CUDA
  - AMD Firestream (ATI)

  (- intel: Larrabee, ~x86)


- OpenCL (Open Computing Language): nyílt szabvány heterogén rendszerek (pl. CPU+GPU) programozáshoz

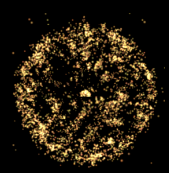


# CUDA

## Compute Unified Device Architecture

- Magas szintű kiterjesztés a C/C++ nyelvhez
- CUDA programozási és memória modell
- nvcc fordító

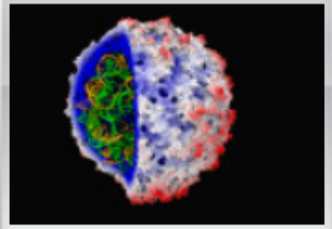
→ GPUmagok számával skálázódó programok (kód újrafordítása nélkül)



# Példák

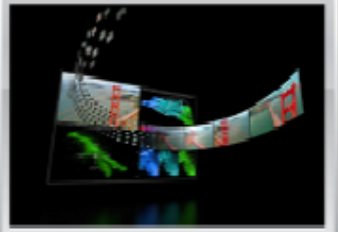
146X

Interactive visualization of volumetric white matter connectivity



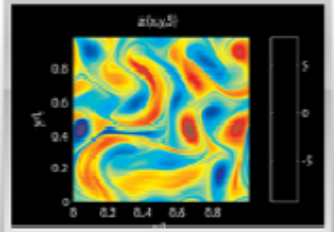
36X

Ionic placement for molecular dynamics simulation on GPU



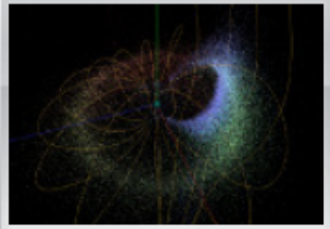
19X

Transcoding HD video stream to H.264



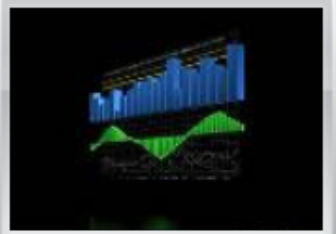
17X

Fluid mechanics in Matlab using .mex file CUDA function



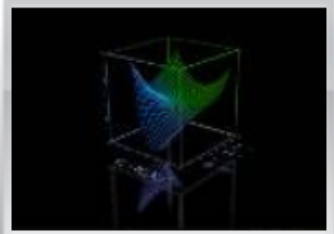
100X

Astrophysics N-body simulation



149X

Financial simulation of LIBOR model with swaptions



47X

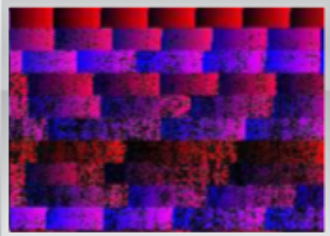
GLAME@lab: an M-script API for GPU linear algebra

20X

Ultrasound medical imaging for cancer diagnostics

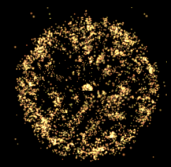
24X

Highly optimized object oriented molecular dynamics



30X

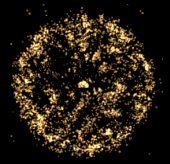
Cmatch exact string matching to find similar proteins and gene sequences



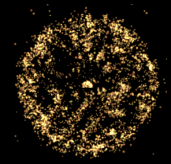
# Példák

## Molekuladinamika és kvantumkémia programok GPU-val gyorsítva

| Code | Version | Release Target | Notes |
|---|---|---|---|
| VMD | 1.8.7 | Released | Free download from VMD site |
| NAMD | 2.7 beta | September 2009 | Nightly build being rolled into beta 2 |
| HOOMD | 0.8.1 | Released | Multi-GPU support in 0.8.1 |
| HMMER | 0.9.1 | Released | GPU-HMMER available now |
| Autodock | 0.9 | Beta | From Silicon Informatics |
| CHARMM | Beta | December 2009 | |
| Amber | Alpha patch | | Generalized Born only, 1 GPU |
| GROMACS | 4.0 | Released | CUDA client based on OpenMM adds support for implicit solvent |
| LAMMPS | Alpha | Released (alpha) | 2 pair styles ported to GPU |



# CUDA - Beüzemelés



# CUDA-képes hardware

## http://www.nvidia.com/object/cuda_gpus.html

NVIDIA

Search NVIDIA USA

DOWNLOAD DRIVERS | COOL STUFF | SHOP | PRODUCTS | TECHNOLOGIES | COMMUNITIES | SUPPORT

CUDA ZONE | WHAT'S NEW | WHAT IS CUDA? | CUDA GPUs | DEVELOPERS

NVIDIA Home > Technologies > CUDA Zone > CUDA GPUs

Share this page

### CUDA GPUs

NVIDIA CUDA-enabled GPUs power millions of desktops, notebooks, workstations, and supercomputers around the world, accelerating computationally-intensive tasks for consumers, professionals, scientists, and researchers.

1.) How can I find out which GPU is in my computer?
2.) Do I have a CUDA-enabled GPU in my computer?
3.) How do I know if I have the latest drivers?
4.) How can I get a CUDA-enabled GPU?
5.) How can I download the CUDA software development kit?

**CUDA ENABLED GEFORCE PRODUCTS**

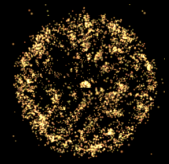
GeForce 8, 9, 100, 200-series GPUs with a minimum of 256MB of local graphics memory.

GEFORCE DESKTOP PRODUCTS

GeForce GTX 295
GeForce GTX 285
GeForce GTX 285 for Mac
GeForce GTX 280
GeForce GTX 275
GeForce 9800 GTX+
GeForce 9800 GTX
GeForce 9800 GT
GeForce 9600 GSO
GeForce 9600 GT

GEFORCE NOTEBOOK PRODUCTS

GeForce GTX 285M
GeForce GTX 280M
GeForce GTX 260M
GeForce GTS 360M
GeForce GTS 350M
GeForce 9800M GT
GeForce 9800M GTS
GeForce 9800M GS
GeForce 9700M GTS
GeForce 9700M GT



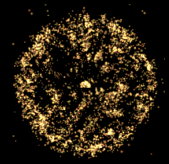
# CUDA telepítés

Hardware:

GPU: 4 x nVidia GTX295
(1920 x 1.3GHz mag)
CPU: Intel(R) Xeon(R) CPU
E5520 @ 2.27GHz
(8 mag, 16 thread)

Software:

OpenSuse 11.2 Linux
gcc-4.4.1
kernel 2.6.31.5-0.1 x86_64



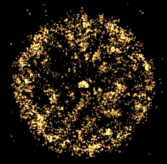
# CUDA telepítés

Hardware:

GPU: 4 x nVidia GTX295
(1920 x 1.3GHz mag)
CPU: Intel(R) Xeon(R) CPU
E5520 @ 2.27GHz
(8 mag, 16 thread)

Software:

OpenSuse 11.2 Linux
gcc-4.4.1
kernel 2.6.31.5-0.1 x86_64



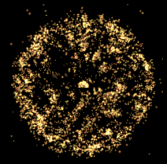
# CUDA telepítés

Hardware:

GPU: 4 x nVidia GTX295
(1920 x 1.3GHz mag)
CPU: Intel(R) Xeon(R) CPU
E5520 @ 2.27GHz
(8 mag, 16 thread)

Software:

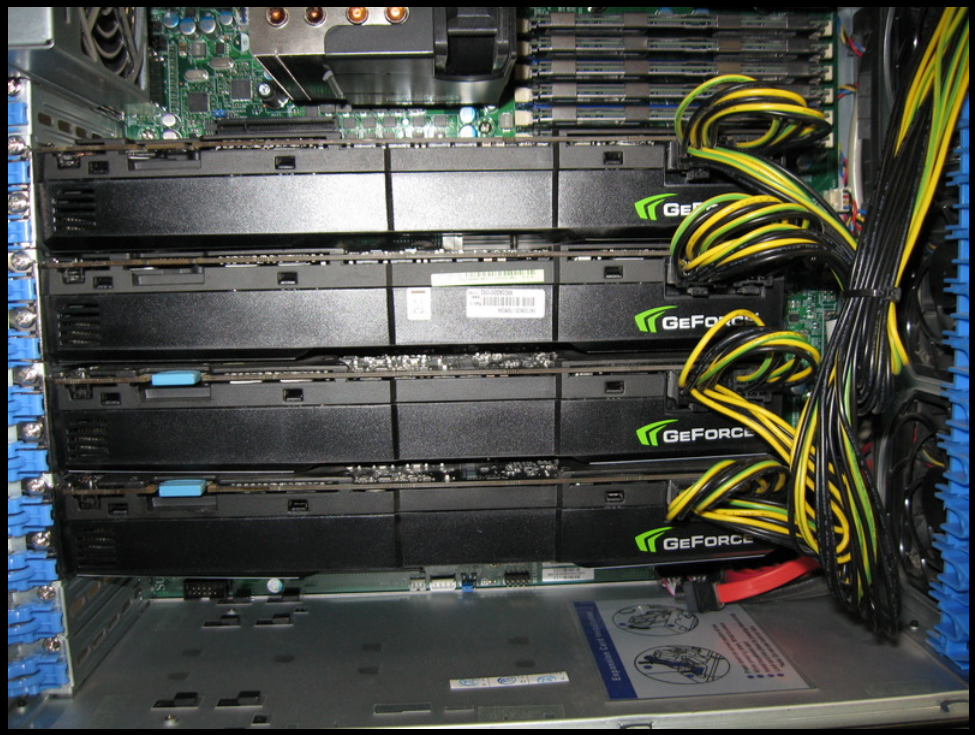
OpenSuse 11.2 Linux
gcc-4.4.1
kernel 2.6.31.5-0.1 x86_64



# CUDA telepítés

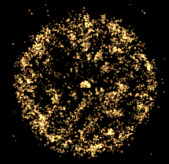
Hardware:

GPU: 4 x nVidia GTX295
(1920 x 1.3GHz mag)
CPU: Intel(R) Xeon(R) CPU
E5520 @ 2.27GHz
(8 mag, 16 thread)




Software:

OpenSuse 11.2 Linux
gcc-4.4.1
kernel 2.6.31.5-0.1 x86_64



# CUDA telepítés

**Driver** (root-ként telepíteni)
nvidia.ko; /usr/lib64/libcuda.so

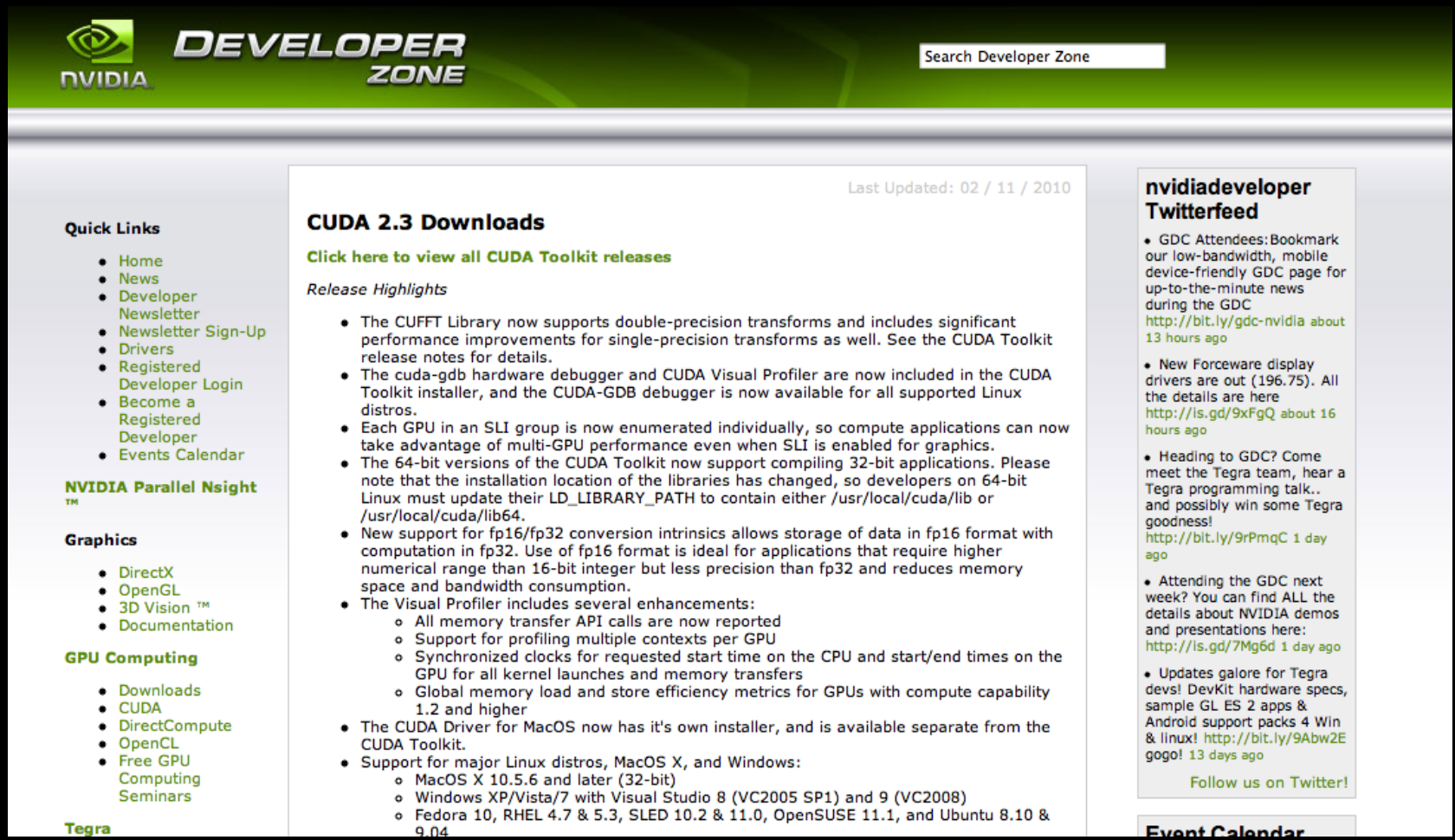
**Toolkit** (célszerű root-ként telepíteni)
nvcc compiler; CUDA FFT,BLAS; profiler; gdb
default install: /usr/local/cuda

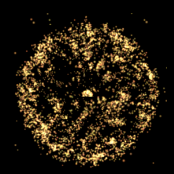
**SDK** (Software Development Kit;
lehet felhasználóként is telepíteni)
Példaprogramok; ~/NVIDIA_GPU_Computing_SDK

# CUDA letöltések

http://developer.nvidia.com/object/cuda_2_3_downloads.html

http://developer.nvidia.com/object/cuda_3_0_downloads.html





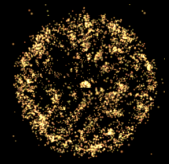
# CUDA letöltések

http://developer.nvidia.com/object/cuda_2_3_downloads.html

## Linux

| | | |
|---|---|---|
| Developer Drivers for Linux (190.53) | 32-bit 64-bit | |
| CUDA Toolkit for Fedora 10 | 32-bit 64-bit | |
| CUDA Toolkit for RedHat Enterprise Linux 5.3 | 32-bit 64-bit | |
| CUDA Toolkit for RedHat Enterprise Linux 4.7 | 32-bit 64-bit | |
| CUDA Toolkit for OpenSUSE 11.1 | 32-bit 64-bit | |
| CUDA Toolkit for SUSE Linux Enterprise Desktop 11 | 32-bit 64-bit | |
| CUDA Toolkit for SUSE Linux Enterprise Desktop 10 2 | 32-bit 64-bit | |
| CUDA Toolkit | | Documentaion |
| CUDA SDK code samples and more | download | Release Notes<br>License |

## MacOS



# CUDA telepítés

**Driver** NVIDIA-Linux-x86_64-190.53-pkg2.run

```
root@linux> sh NVIDIA-Linux-x86_64-190.53-pkg2.run
```

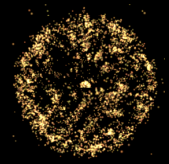
szükséges: Base-development (gcc, make)
Kernel-devel (kernel-source, headers)

**Toolkit** cudatoolkit_2.3_linux_64_suse11.1.run

```
root@linux> sh cudatoolkit_2.3_linux_64_suse11.1.run
```

**SDK** cudasdk_2.3_linux.run

```
user@linux> sh cudasdk_2.3_linux.run
```



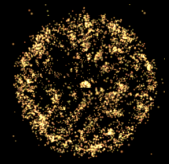
# CUDA telepítés - rendszerbeállítások

- Futtatni `release_notes_linux.txt` (CUDA download page) scriptjét (-> load module, devs)
- ~/.bashrc-be:

```
export PATH=/usr/local/cuda/bin
export LD_LIBRARY_PATH=/usr/local/cuda/lib64
```

Megj. ("nem támogatott" linux esetén):

- nvcc nem kompatibilis gcc-4.4-gyel ⟶ gcc-4.3
- SDK példaprog.-hoz kellett: freeglut freeglut-devel



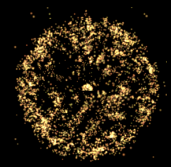
# SDK - példaprogramok

Fordítás:

```
cd ~/NVIDIA_GPU_Computing_SDK/C
make
```

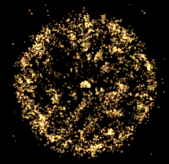
Példaprogramok:

```
cd ~/NVIDIA_GPU_Computing_SDK/C/bin/linux/release
ls
```



# SDK - példaprogramok

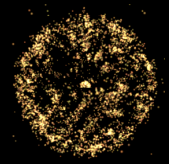
3dfd
alignedTypes
asyncAPI
bandwidthTest
bicubicTexture
binomialOptions
BlackScholes
boxFilter
clock
convolutionFFT2D
convolutionSeparable
convolutionTexture
cppIntegration
dct8x8
deviceQuery
deviceQueryDrv
dwtHaar1D
dxtc
eigenvalues
fastWalshTransform
fluidsGL
histogram
imageDenoising
lineOfSight
Mandelbrot
marchingCubes
matrixMul
matrixMulDrv
matrixMulDynlinkJIT
MersenneTwister
MonteCarlo
MonteCarloMultiGPU
nbody
oceanFFT
particles
postProcessGL
ptxjit
quasirandomGenerator
radixSort
recursiveGaussian
reduction
scalarProd
scan
scanLargeArray
simpleAtomicIntrinsics
simpleCUBLAS
simpleCUFFT
simpleGL
simpleMultiGPU
simplePitchLinearTexture
simpleStreams
simpleTemplates
simpleTexture
simpleTexture3D
simpleTextureDrv
simpleVoteIntrinsics
simpleZeroCopy
smokeParticles
SobelFilter
SobolQRNG
sortingNetworks
template
threadFenceReduction
threadMigration
transpose
transposeNew
volumeRender



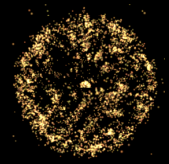
# CUDA LIVECD

CD-ről futtatható 32bites OpenSuse 11.2 + CUDA 2.3:

http://www.szfki.hu/~jurek/archive/2010_CUDAseminar/index.html



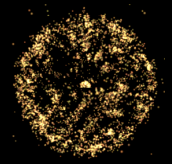
# Számítások GPU-n
# CUDA programozás nélkül



# Könyvtárak

Optimalizált könyvtárak használata jelentősen felgyorsíthatja kódfejlesztésünket.

Pl.
- BLAS (Basic Linear Algebra Subprograms)
- FFTW (Fast Fourier Transform)

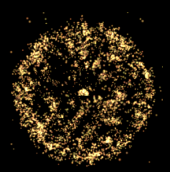


# Könyvtárak

Optimalizált könyvtárak használata jelentősen felgyorsíthatja kódfejlesztésünket.

Pl.
- BLAS (Basic Linear Algebra Subprograms)
- FFTW (Fast Fourier Transform)

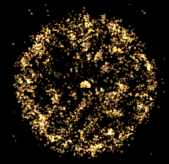
CUDA megvalósítás (Toolkit része):
- CUBLAS
- CUFFT



# Pl. CUBLAS

## Alapmodell

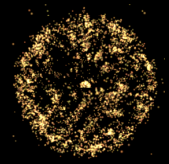
- Létrehozzuk a vektor/mátrix objektumokat a GPU-n (host -> device adattranszfer)
- Meghívjuk a CUBLAS függvényeket (számolás GPU-n)
- Visszaolvassuk az eredményeket (device -> host adattranszfer)
- Általános stratégia: minél tovább számolni a GPU-n



# Portland accelerator

```
int main(void)
{
   …

   {
     ...
   }

   …
}
```

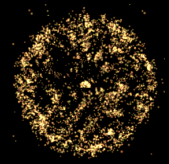
```
#include <accelmath.h>
int main(void)
{
   …

   #pragma acc region
   {
     ...
   }

   …
}
```



# Portland accelerator

**Fordítás:**

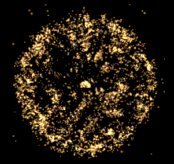
```
pgcc sample.c -O3 -ta=nvidia -Minfo -fast
```



# Portland accelerator

Fordítási üzenet:

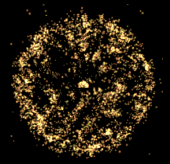
```
237, Accelerator restriction: size of the GPU copy
of an array depends on values computed in this loop
238, Accelerator restriction: size of the GPU copy
of 'm' is unknown
     Accelerator restriction: size of the GPU copy
of 'xi' is unknown
     Accelerator restriction: one or more arrays
have unknown size
     Loop not vectorized: data dependency
```



# Portland accelerator

Fordítási üzenet:

```
235, Generating copyin(xi[0:natom-1][0:2])
     Generating copyin(m[0:natom-1])
     Generating copyout(ai[0:natom-1][0:2])
     Generating compute capability 1.0 kernel
     Generating compute capability 1.3 kernel
237, Loop is parallelizable
     Accelerator kernel generated
    237, #pragma acc for parallel, vector(32)
         Non-stride-1 accesses for array 'xi'
         Non-stride-1 accesses for array 'ai'
244, Complex loop carried dependence of 'ai'
prevents parallelization
     Loop carried reuse of 'ai' prevents
parallelization
     Inner sequential loop scheduled on accelerator
```

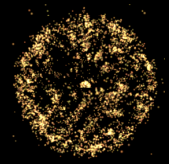


# Portland accelerator

A CPU kód szervezésétől erősen függhet a GPUs gyorsítás hatékonysága!

Irodalom pl.:

http://www.pgroup.com/resources/accel.htm
- http://www.pgroup.com/lit/articles/insider/v1n1a1.htm
- http://www.pgroup.com/lit/articles/insider/v1n2a1.htm
- http://www.pgroup.com/lit/articles/insider/v1n3a1.htm

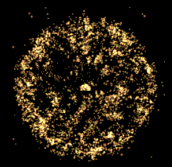


# Magasabb szintü nyelvek + CUDA

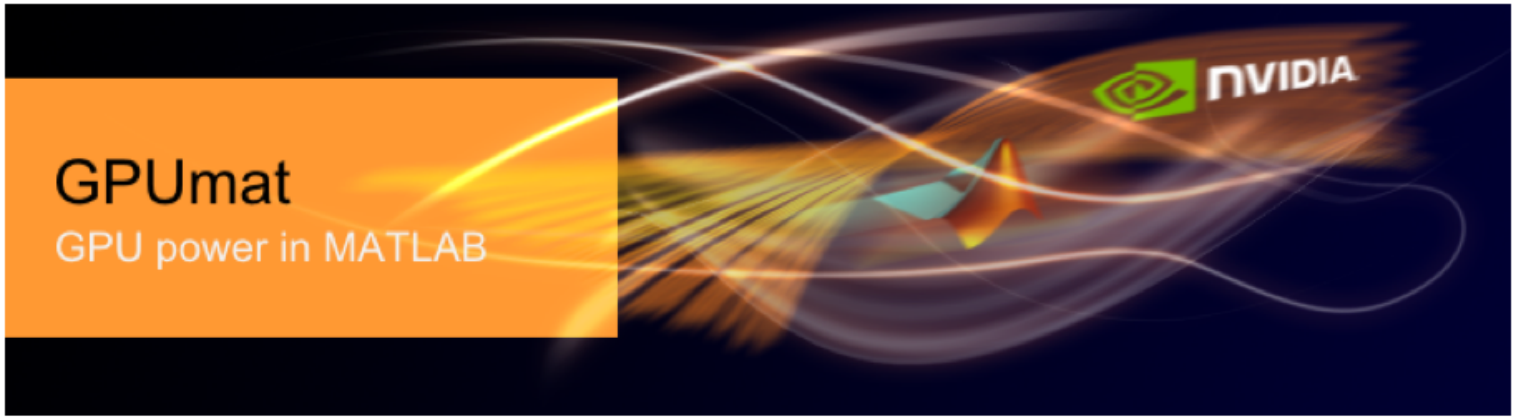
Természetes a CUDA kiterjesztés, ha lehetőség van C-ben írt modulok használatára

- Python – PyCUDA
- Matlab
  - mex -> nvmex
    http://developer.nvidia.com/object/matlab_cuda.html
  - GPUmat / Jacket

Ügyelni kell a nyelvek közötti adatátvitel többletidejére!



# Matlab + CUDA




GPUmat allows standard MATLAB code to run on GPUs. The execution is transparent to the user as shown in the following example:

```
A = GPUsingle(rand(100)); % A is on GPU memory
B = GPUdouble(rand(100)); % B is on GPU memory
C = A+B; % executed on GPU.
D = fft(C); % executed on GPU
```

**Executed on GPU**

```
A = single(rand(100)); % A is on CPU memory
B = double(rand(100)); % B is on CPU memory
C = A+B; % executed on CPU.
D = fft(C); % executed on CPU
```

**Executed on CPU**

# Matlab + CUDA

GPUmat allows sta ing example:

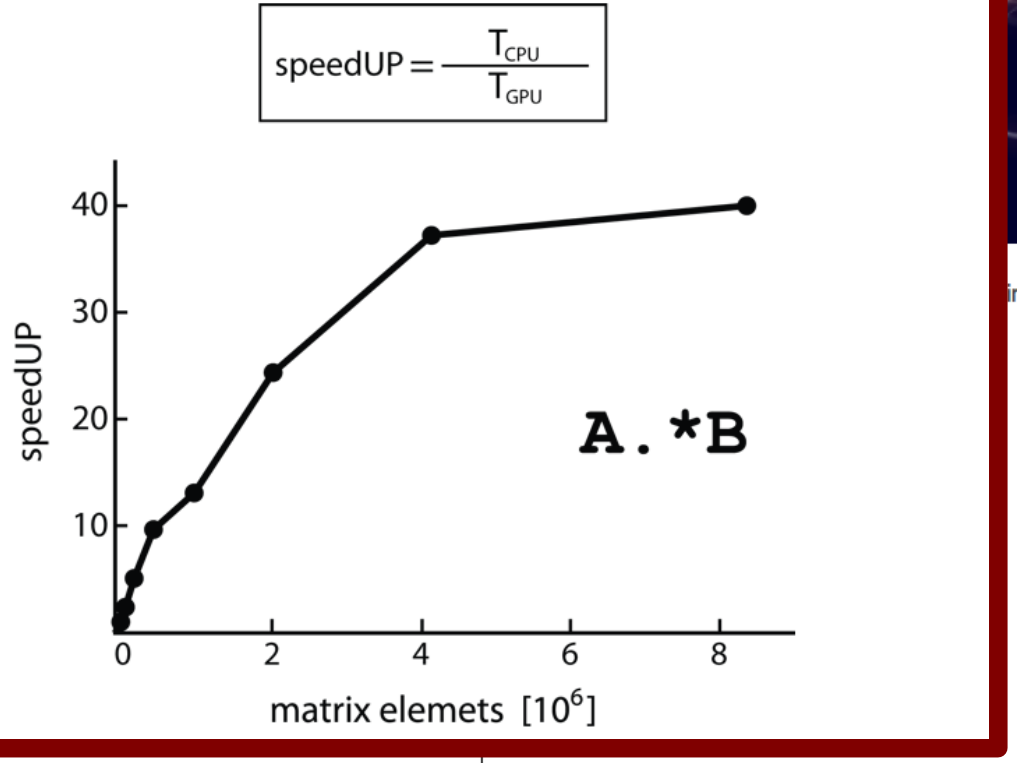
```
A = GPUsingle
B = GPUdouble
C = A+B; % ex
D = fft(C);
```

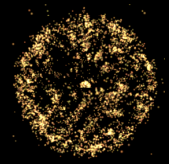
Executed on GP

```
A = single(ra
B = double(ra
C = A+B; % ex
D = fft(C); % executed on CPU
```

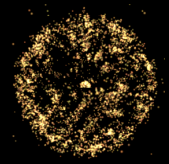
Executed on CPU

$$speedUP = \frac{T_{CPU}}{T_{GPU}}$$





# C gyorstalpaló

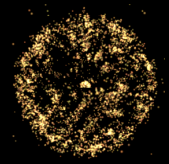


# C gyorstalpaló

## main függvény, változók

```
int main(void)          // main fg. definiálás
{
   int i;               // lokális változó def.

   i=1;                 // értékadás

   return(i);           // fg. visszatérési érték
}

// Ez egy komment
```

**Adattípusok**: int, float, double, char, ...



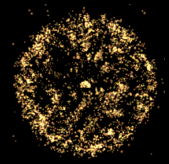
# C gyorstalpaló

Fordítás:

```
gcc -o test.out test.c
```

Futtatás:

```
./test.out
```

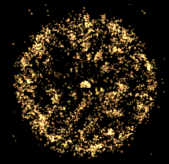


# C gyorstalpaló

## Külső függvények

```
#include <stdio.h>   // küls   fg-ek deklarálása
void main(void)      // main fg. definiálás
{
   int i;            // lokális változó def.

   i=1;              // értékadás

   printf(”%d\n”,i); // küls   fg. hívása
}

// Ez egy komment
```

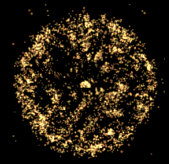
Adattípusok: int, float, double, char, ...



# C gyorstalpaló

## Külső függvények

```
#include <math.h>    //
#include <stdio.h>

void main(void)
{
   double x;

   x = sqrt(2.0);    // Küls   fg.

   printf(”%e\n”,x);
}
```



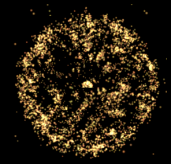
# C gyorstalpaló

Fordítás:

```
gcc -o test.out test.c -lm
```

Futtatás:

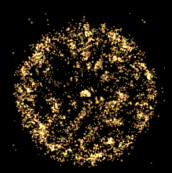
```
./test.out
```



# C gyorstalpaló

## Függvények

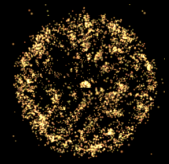
```
int foo(int i, int j) {   // fg. definiálás
   return(j*i);           // visszatérési érték
}

int main(void)            // main fg. definiálás
{
   int i=1;               // lokális változó def.

   i = foo(2,i);          // függvényhívás

   return(i);             // fg. visszatérési érték
}
```



# C gyorstalpaló

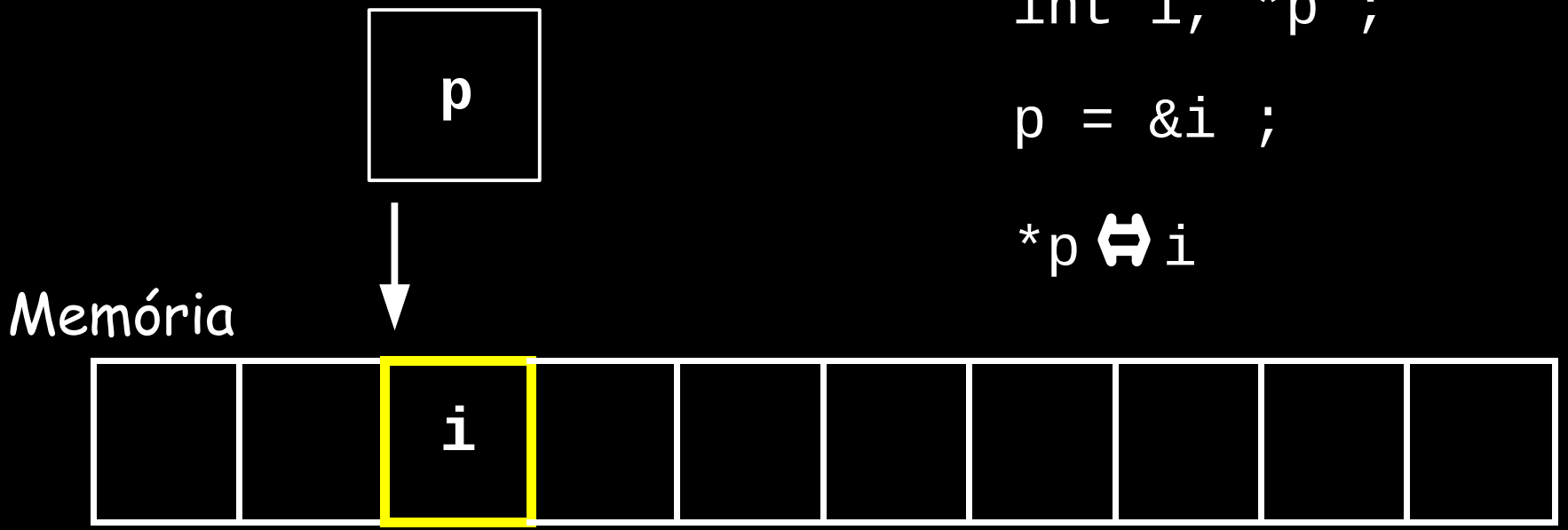
## Függvények

```
int foo(int i, int j);    // fg. deklarálás

int main(void) {          // main fg. definiálás
   int i=1;               // lokális változó def.

   i = foo(2,i);          // függvényhívás

   return(i);             // fg. visszatérési érték
}

int foo(int i, int j) {   // fg. definiálás
   return(j*i);           // visszatérési érték
}
```

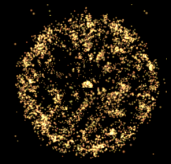


# C gyorstalpaló

## Mutatók



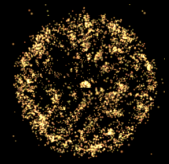

```
int i, *p ;

p = &i ;

*p ⇔ i
```



# C gyorstalpaló

## Mutatók

```
#include <stdio.h>
void main(void)
{
   int i, *p;                    // mutató definiálás

   i=1;

   p = &i;                       // mutató i címére
   printf("%d %d\n",i,*p);// p értékének kiírása

   *p = 2;                     // értékadás p értékére
   printf("%d %d\n",i,*p);   // kiiratás
}
```



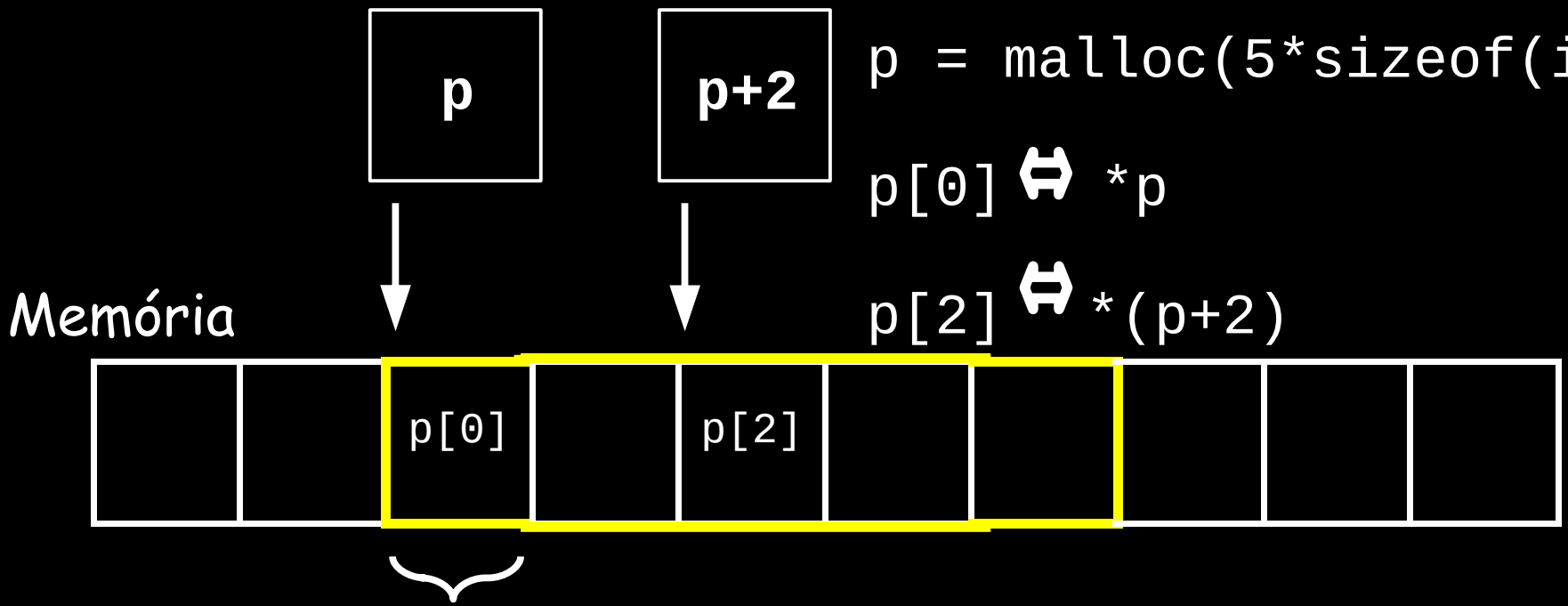
# C gyorstalpaló

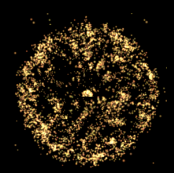
## Mutatók - tömbök

```
int *p ;
p = malloc(5*sizeof(int));
```

p[0] ⇔ *p

p[2] ⇔ *(p+2)

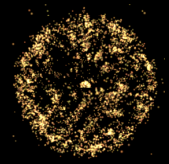




# C gyorstalpaló

## Tömbök - vektorok

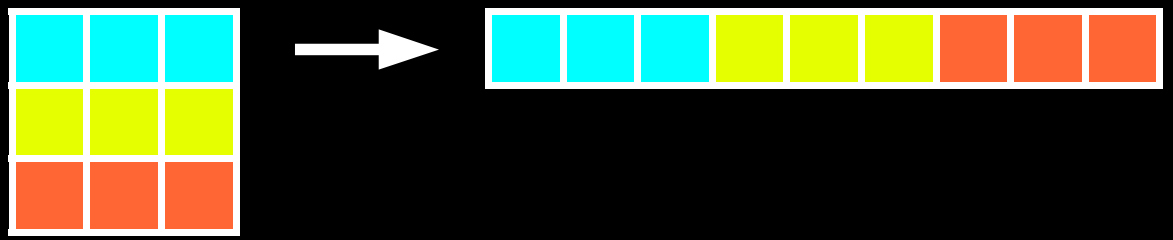
```
#include <stdlib.h>
void main(void)    {
   int i, *p, N=10;

   p = (int*)malloc(N*sizeof(int)); //din.mem.f.

   for (i=0; i<N; i=i+1)          // tömbfeltöltés
   {
      p[i] = 2*i;                 // vektorelem
   }                  // 2.0: double, 2.0f: float

   free(p);           // mem. felszabadítás
}
```

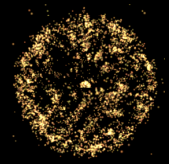


# C gyorstalpaló

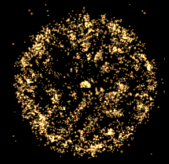
## Tömbök - mátrixok

```
#include <stdlib.h>
void main(void) {
   int i, j, *p, N=10, M=20;

   p = (int*)malloc(N*M*sizeof(int)); //mem.f.

   for (i=0; i<N; i=i+1) {
     for (j=0; j<M; j=j+1) {
        p[i*M+j] = 2*(i*M+j);   // tömbfeltöltés
     }
   }
   free(p);
}
```





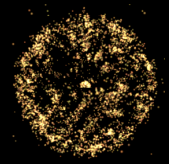
# GPU hardware felépítés
# és
# CUDA programozási modell



# Programozási modell

- CUDA programkód (".cu" kiterjesztéssel)
- Fordítás nvcc fordítóval
- Futtatás

Megj.: célszerű egy könyvtárat létrehozni az adott részfeladat GPU-n történő elvégzéséhez, majd azt CPU-s programunkhoz kapcsolni ("linkelni").

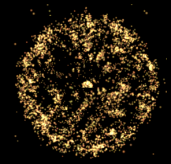


# Programozási modell

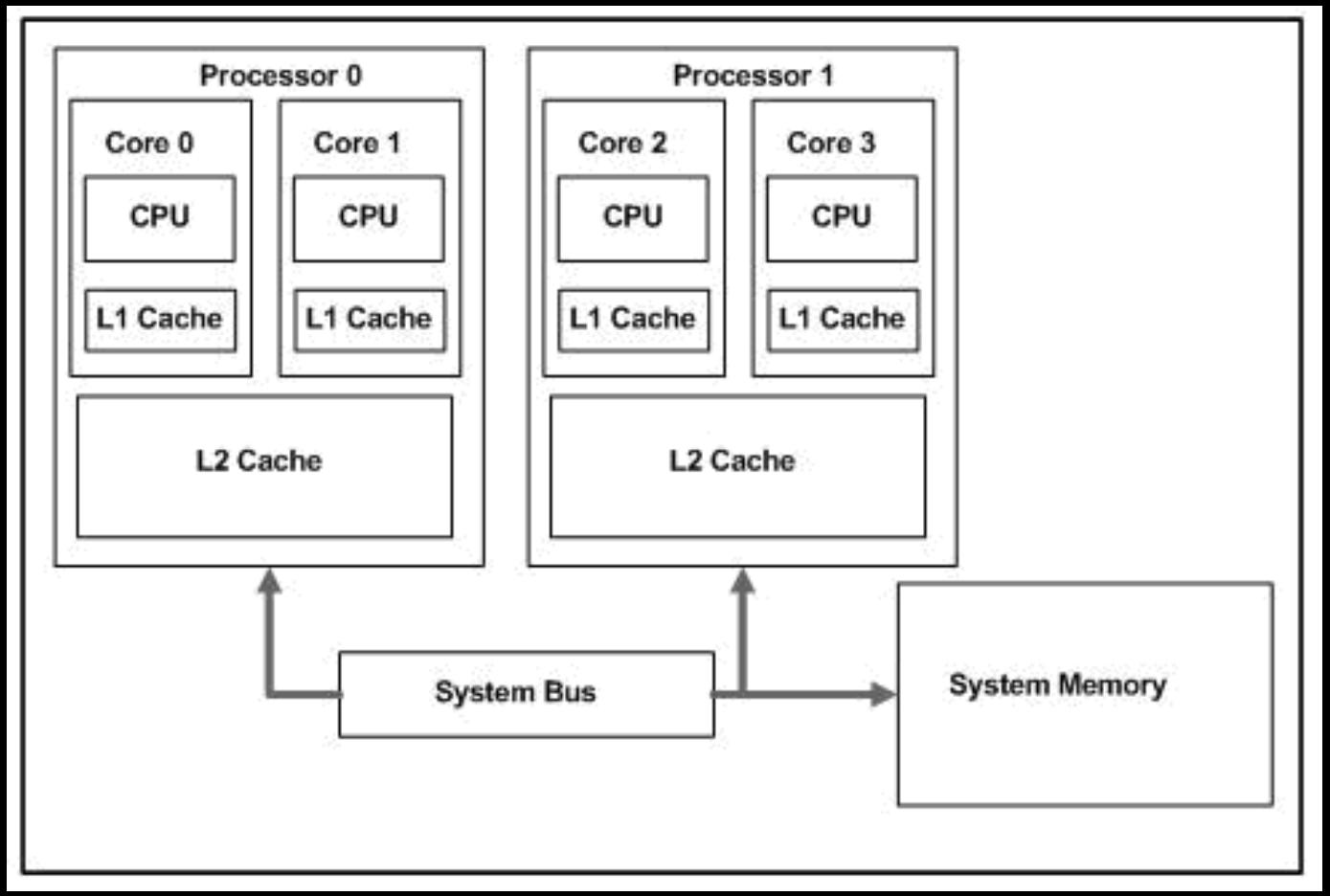
CUDA programkód:

- GPU-n futó rész (**kernel**): fg., ami a számolási feladat időigényes részét számolja (általában eredetileg röveidebb CPU kódrész)

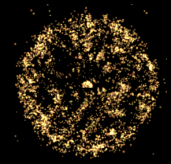
- CPU-n futó rész: adatelőkészítés
  adatmásolás a GPU-ra
  GPU **kernel futtatás**a
  adatmásolás a GPU-ról



# CPU - Fizikai felépítés

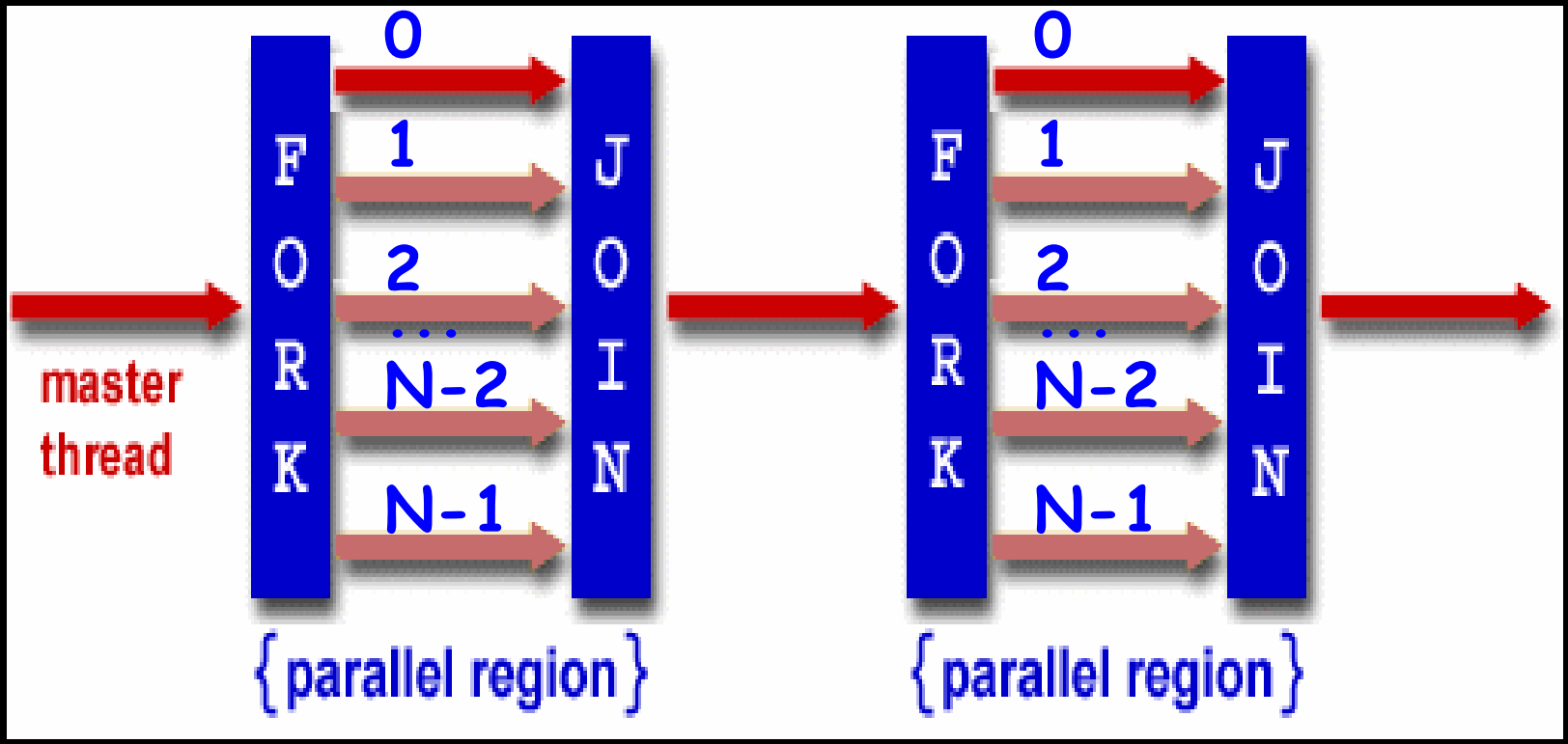



Egyenértékű processzormagok: 0, 1, 2, ..., N-1

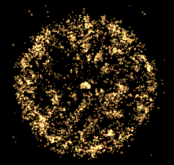


# CPU - Futtatási modell

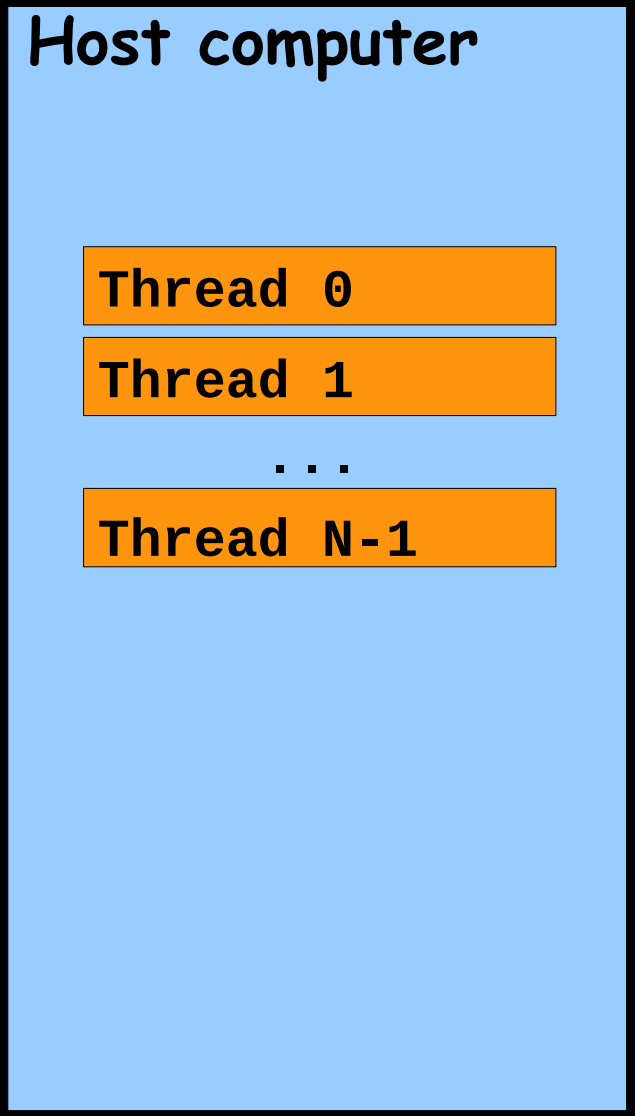
## OpenMP a CPU-n



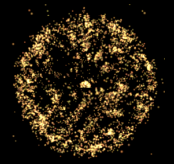

Egyenértékű szálak: 0, 1, 2, ..., N-1



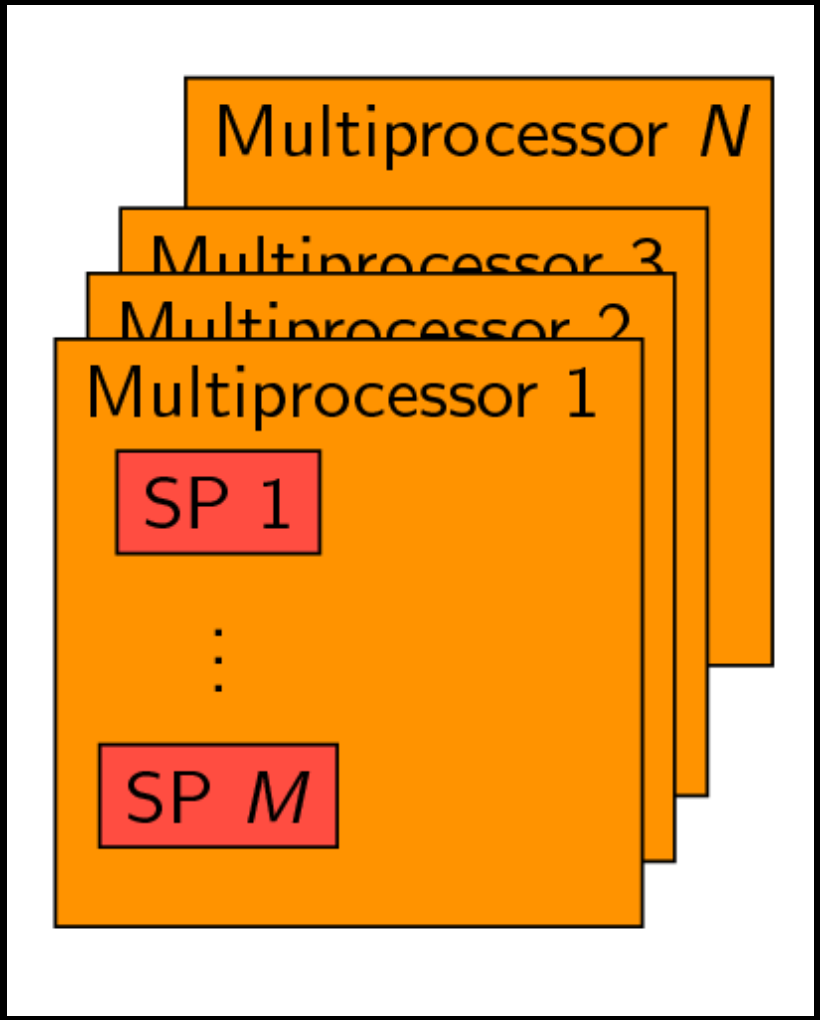
# CPU - Futtatási modell




- Legjobb kihasználtság tipikusan:
  threadek száma = CPUmagok száma
- A threadek száma beállítható:
  `omp_set_num_threads(N) ;`
- Azonosítás: lekérdezhető ID:
  `id = omp_get_thread_num() ;`

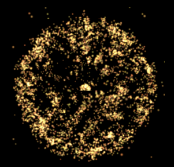


# Fizikai felépítés

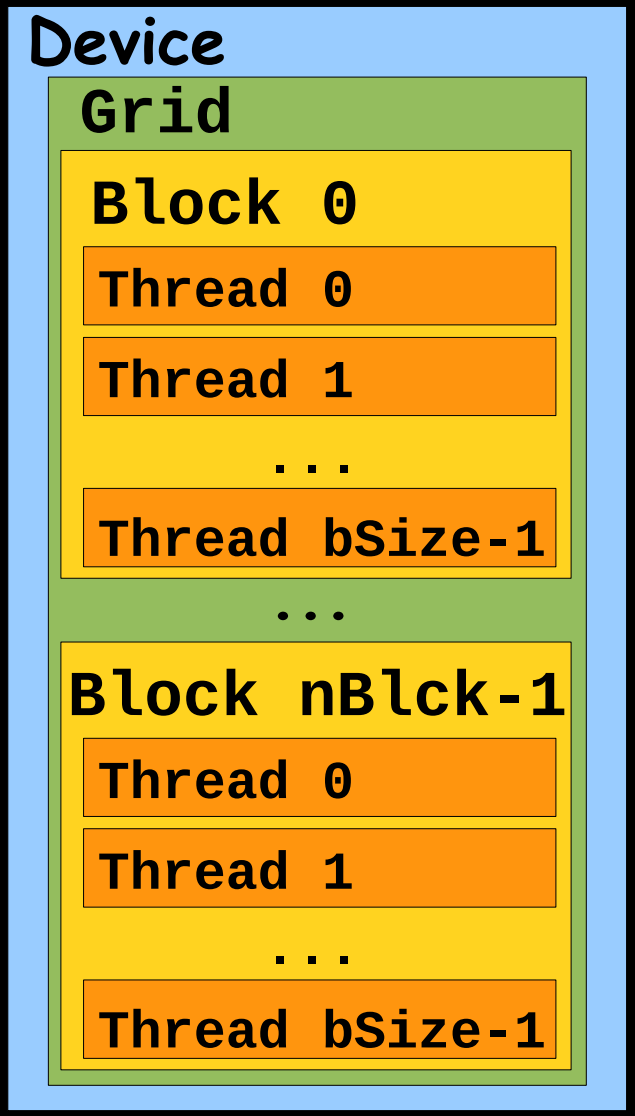



- N db **multiprocessor** (MP)
- M db **singleprocessor**(SP) / MP

| | GeForce 9400M | GTX280 |
|---|---|---|
| N | 2 | 30 |
| M | 8 | 8 |
| Órajel | 0.2 GHz | 1.3 GHz |

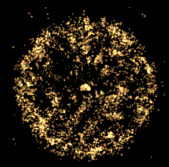


# Futtatási modell




- **Kernel**: GPU-n futó objektum
- A threadek a kernelt hajtják végre
- A grid thread blockokból áll
- A blockok számát és méretét (execution configuration) a host alkalmazás állítja be
- Mindegyik thread azonosítja magát blockIdx, threadIdx (blockon belül), blockDim, gridDim

```
idx = blockIdx.x * blockDim.x +
      threadIdx.x
```



# Futtatási modell

`gridDim.x = 3 , blockDim.x = 4`

`blockIdx.x=0` `blockIdx.x=1` `blockIdx.x=2`

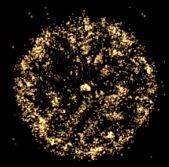
threadIdx.x:

| 0 | 1 | 2 | 3 | | 0 | 1 | 2 | 3 | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

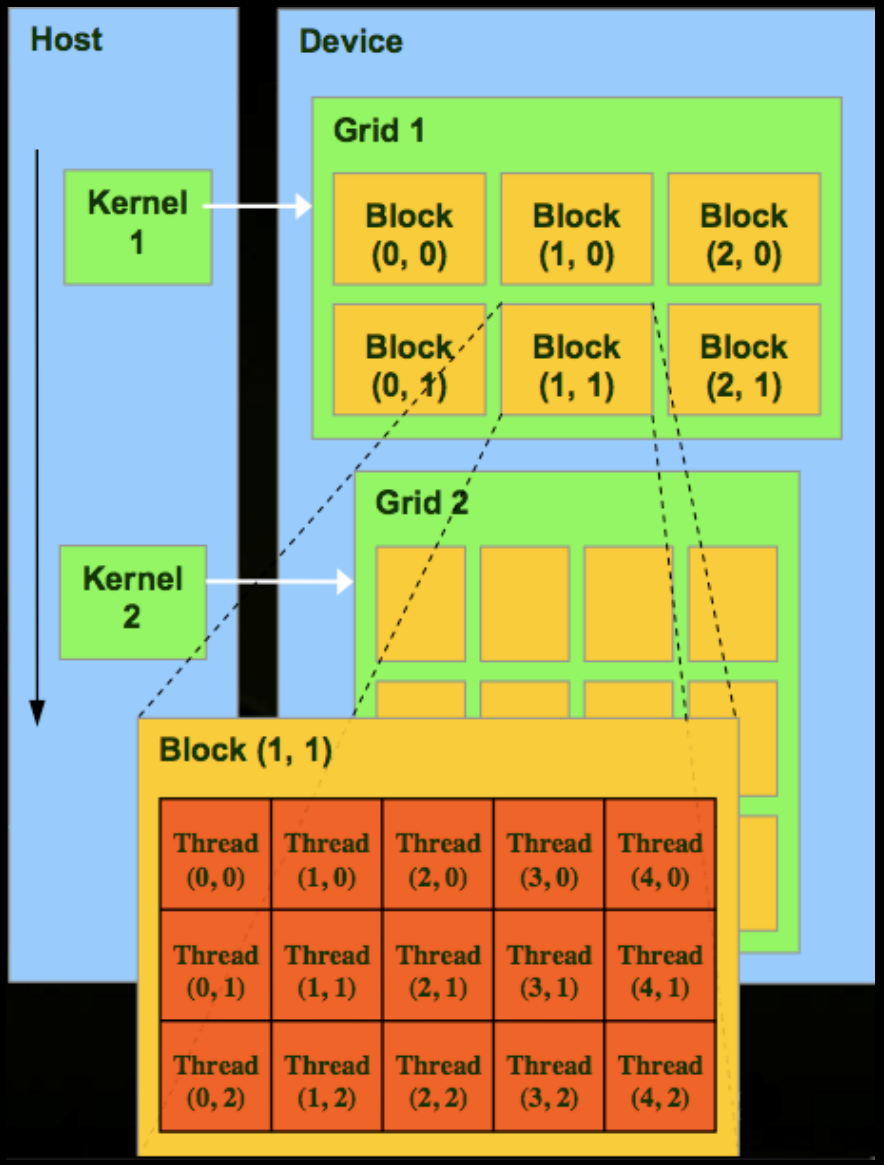
Global threadID:

`idx = blockIdx.x * blockDim.x + threadIdx.x`

| 0 | 1 | 2 | 3 | | 4 | 5 | 6 | 7 | | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# Futtatási modell




- A grid és a blockok méretét a host alkalmazás állítja be

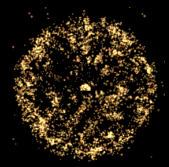
```
tkernel <<<g,th>>> (p,x)
```

ahol pl.

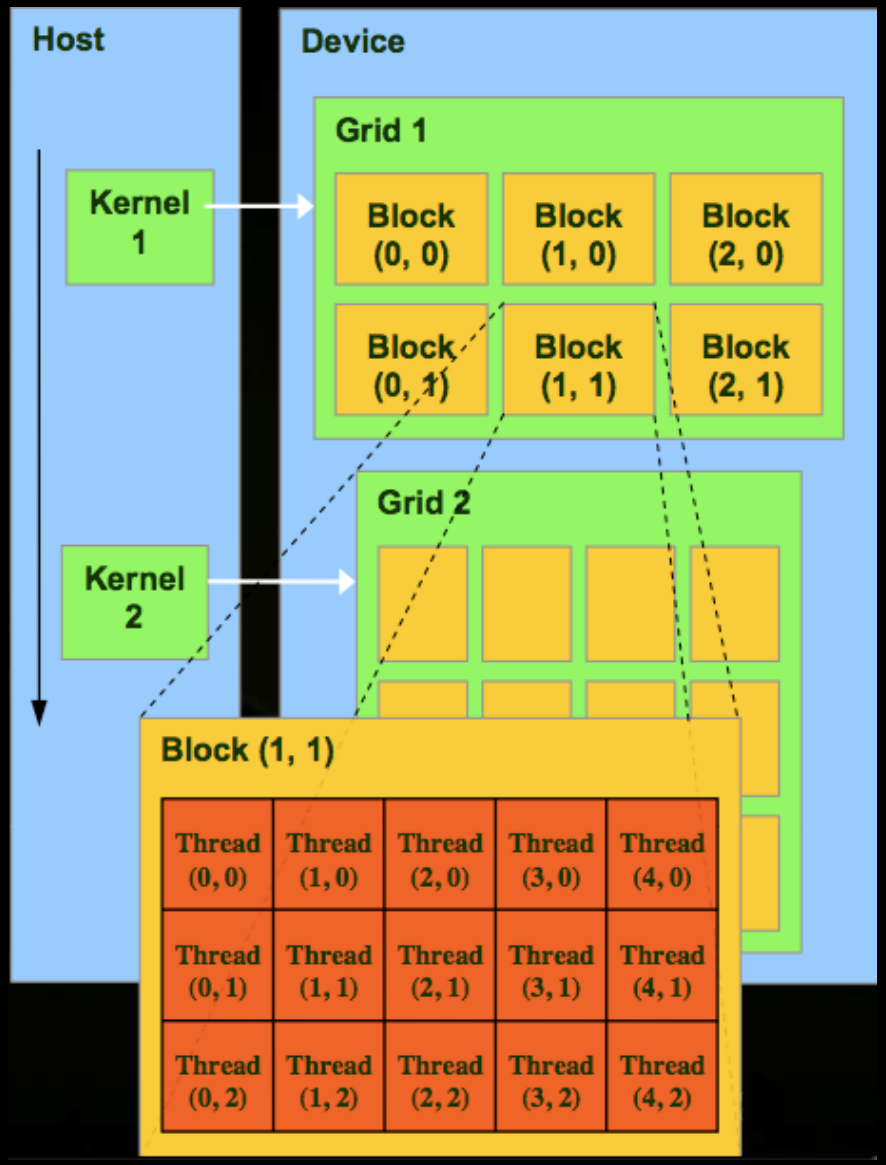
```
__global__ void tkernel(int *p,int x)
```

- Grid dim.: 1 vagy 2
- Block dim.: 1, 2 vagy 3

pl. `threadIdx.x`
`threadIdx.y`
`threadIdx.z`



# Futtatási modell




- Pl. 512 szál / Grid

| blockDim.x | 32 | 64 |
|---|---|---|
| gridDim.x | 16 | 8 |

- Grid dim.: 1 vagy 2

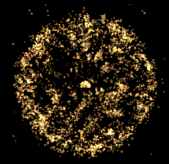
`blockIdx.x,blockIdx.y`

- Block dim.: 1, 2 vagy 3

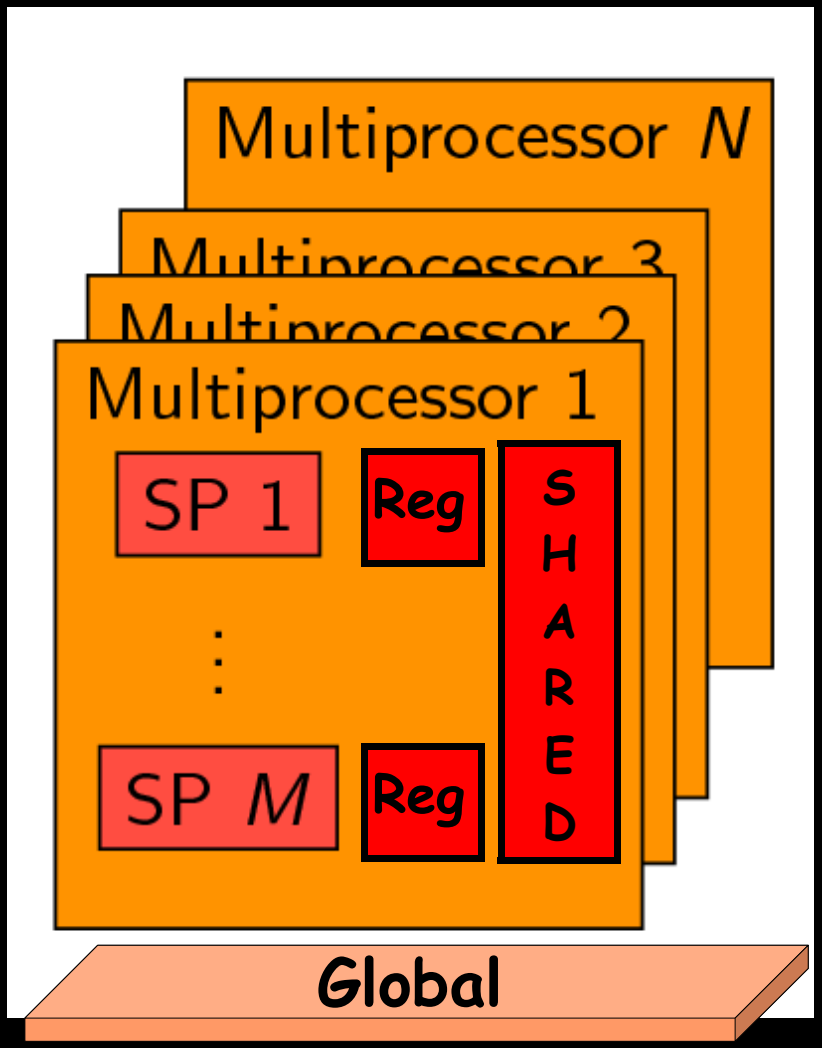
`threadIdx.,threadIdx.y,threadIdx.z`

- Pl. 64 szál / Block

| blockDim.x | 64 | 32 |
|---|---|---|
| blockDim.y | 1 | 2 |



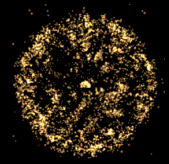
# Fizikai felépítés



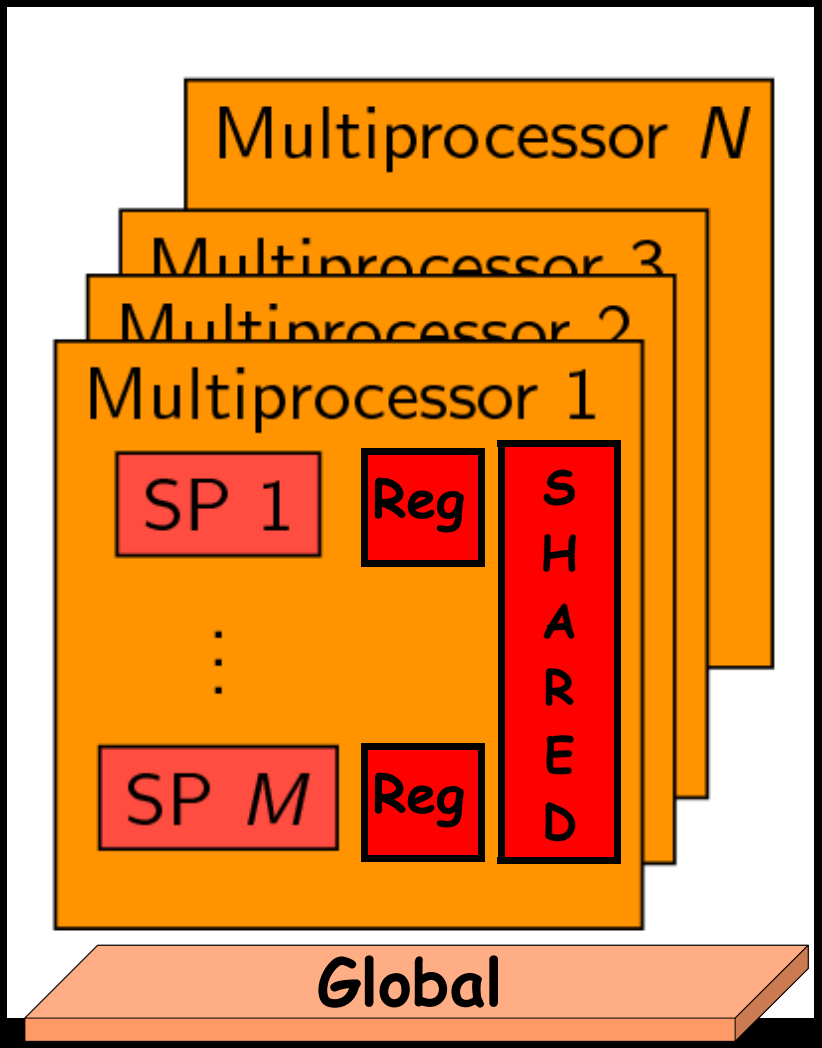

- N db **multiprocessor** (MP)
- M db **singleprocessor**(SP) / MP

(Számunkra) legfontosabb memóriák:

| Mem. | láthatóság | sebesség |
|---|---|---|
| Global | grid | lassú |
| Shared | block | gyors |
| Register | thread | gyors |



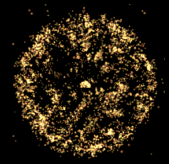
# Fizikai felépítés



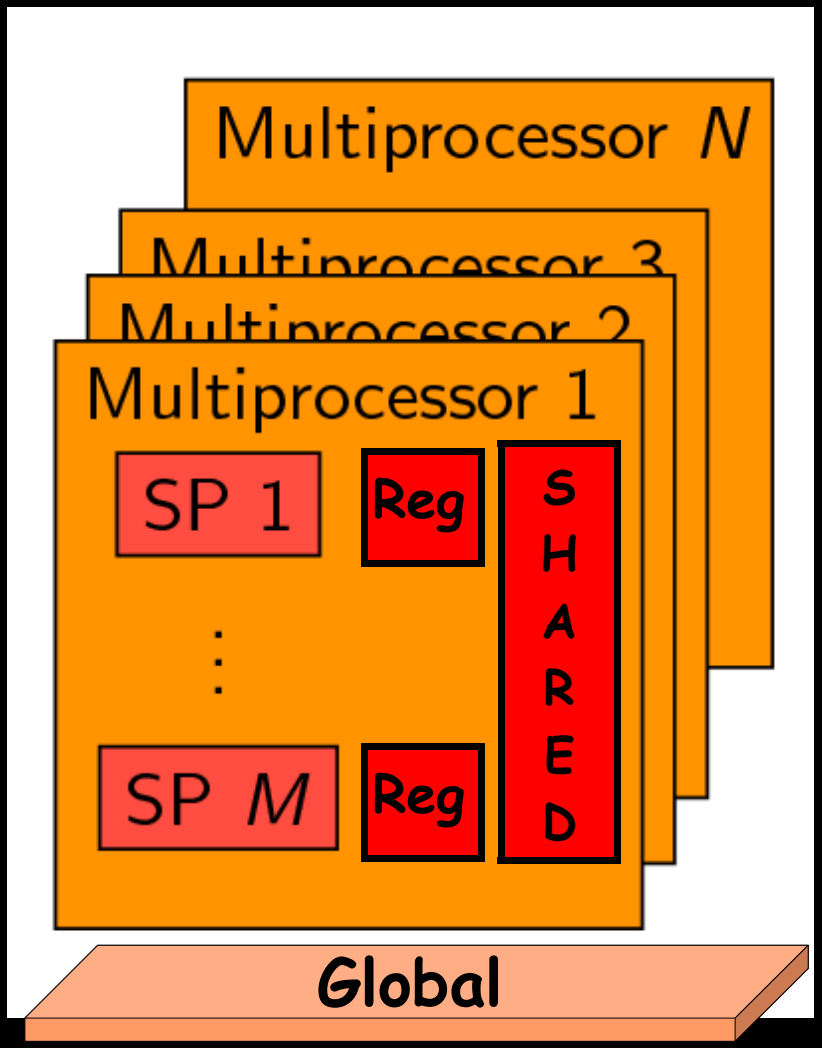

**Global** (~GB):

- lassú elérés (400-600 órajel)
- host és device írja, olvassa
- elérése gyorsítható, ha a szálak rendezetten olvasnak/írnak

pl. `cudaMalloc`
`cudaMemcpy`



# Fizikai felépítés

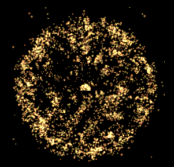



**Register** (16384 db):

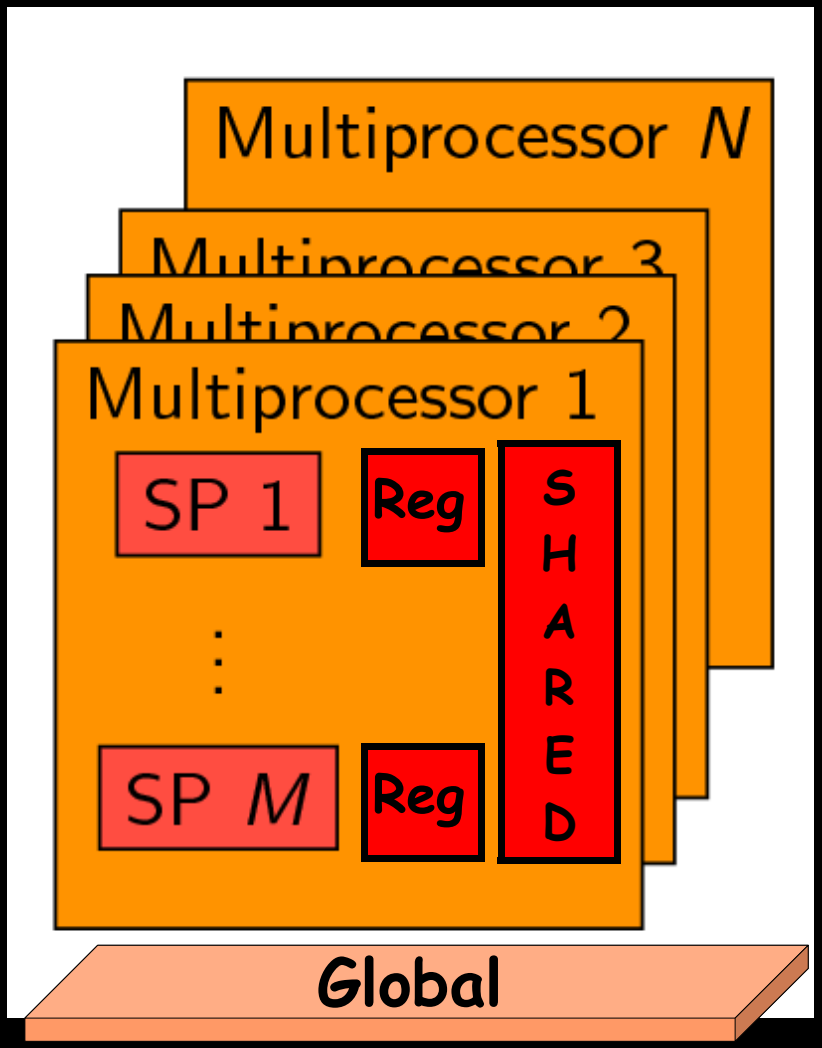
- leggyorsabb memória
- egy thread látja
- csak device írja/olvassa

pl. kernel (`__global__`) lokális változói:

```
int id, N, i;
```



# Fizikai felépítés

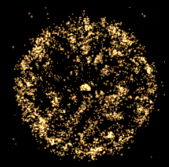



**Shared** (16kB):

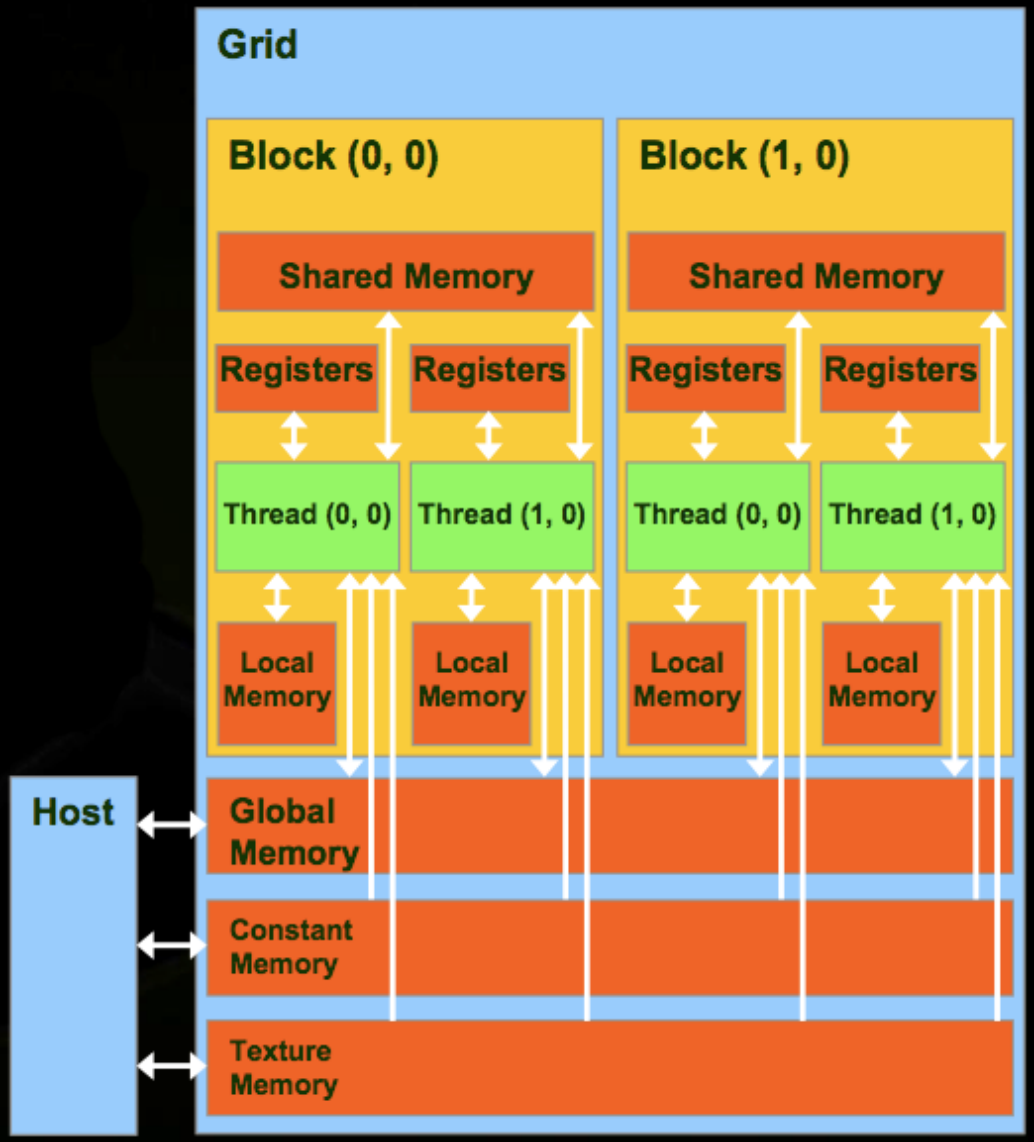
- gyors
- egy Block összes threadje látja
- csak device írja/olvassa

pl. kernelben:

```
__shared__ float x[8]
```



# GPU memória




Global

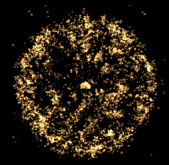
Shared

Register

Constant

Texture

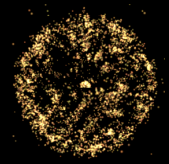
Local



# SDK - deviceQuery

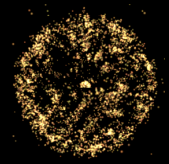
## nVidia GTX295 hardware tulajdonságai

```
There are 8 devices supporting CUDA

Device 0: "GeForce GTX 295"
  CUDA Driver Version:                           2.30
  CUDA Runtime Version:                          2.30
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         3
  Total amount of global memory:                 939261952 bytes
  Number of multiprocessors:                     30
  Number of cores:                               240
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 16384
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.24 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     No
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Default (multiple host threads can use this device simultaneously)
```



# GPU memória

| Memória | láthatóság | R/W | sebesség |
|---|---|---|---|
| Global | grid | RW | lassú |
| Shared | block | RW | gyors |
| Register | thread | RW | gyors |
| Local | thread | RW | lassú |
| Constant | grid | RO | lassú/cache gyorsít |
| Texture | grid | RO | lassú/cache gyorsít |



# GPU memória

## Memória - foglalás

**Global: pl. host (CPU) kódrészben:**

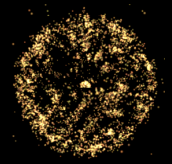
```
cudaMalloc( (void**)&g, memSize);
```

**Register: a kernelben (__global__ függvény)**

**pl. változó deklarálása:** `float r;`

**Shared: kernelben**

**pl.** `__shared__ float s[1000][4];`



# GPU memória

## Memória - adattranszfer

Értékadással, pl.

- Global -> Register

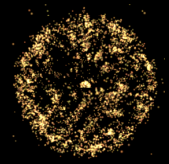
```
r = g[2] ;
```

- Global -> Shared

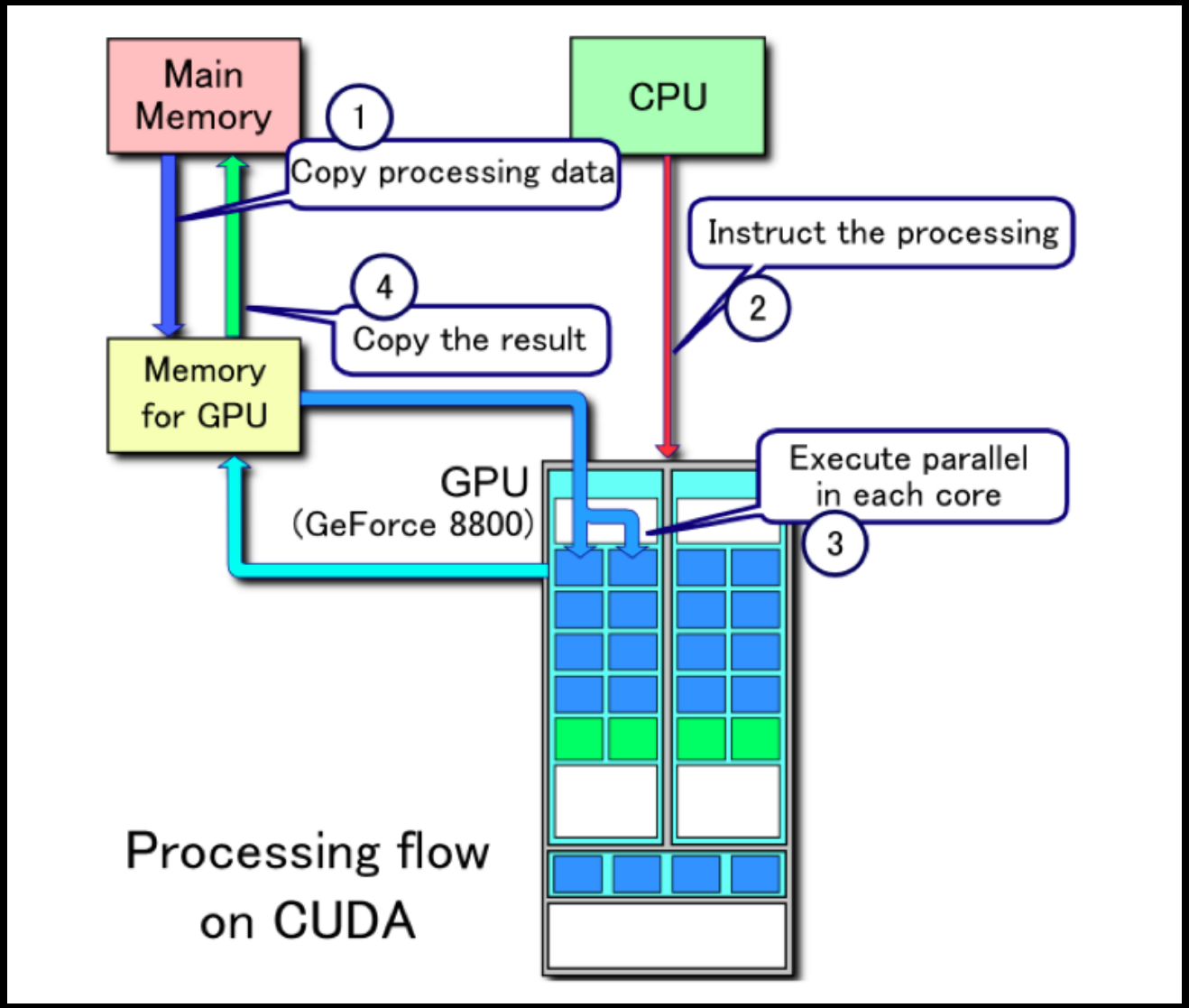
```
s[0][0] = g[1] ;
```

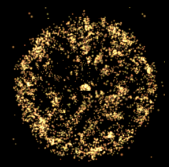
- Shared -> Global

```
g[0] = s[0][0] ;
```



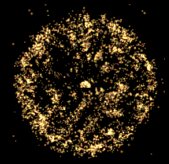
# Számítási folyam





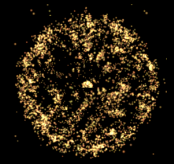
# CUDA kódszerkezet

```
__device__ void mykernel(float *v, int D, float c) {…}

int main(void) {
        …
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
        …
  dim3 gridD     (30);
  dim3 blockD  (16,16);
  mykernel <<< gridD, blockD >>> (d, D, 2.0f);
        …
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```



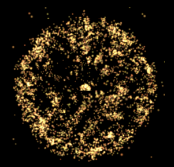
# Példaprogram - soros CPU

```
#include <stdlib.h>


void myfunc(float *v, int D, float c) {
   ...
}

int main(void) {
   int i, D=1000000, s;
   float *h;
   s = D * sizeof(float);      // size in bytes
   h = (float*) malloc(s);
   for (i=0; i<D; i=i+1) { h[i] = (float)i; }
   myfunc (h, D, 2.0f);
   free(h);
}
```



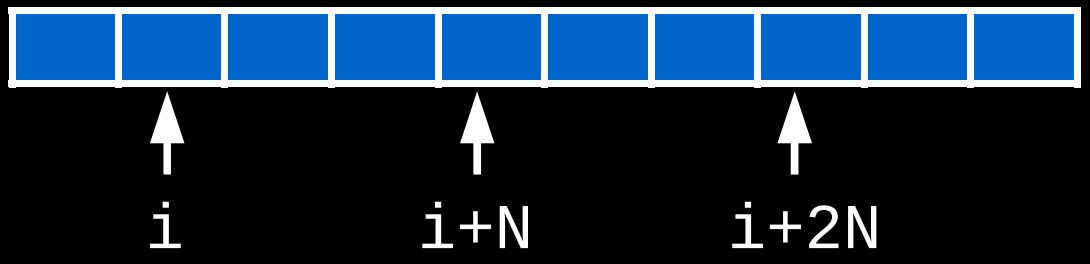
# Példaprogram - soros CPU

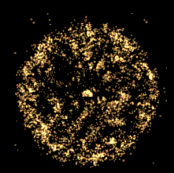
```
#include <stdlib.h>


void myfunc(float *v, int D, float c) {
  int i;
  for (i=0; i<D; i=i+1) {
    v[i] = v[i] * c;
  }
}

int main(void) {
  ...
}
```



# Példaprogram - párhuzamos CPU

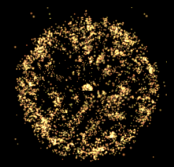
```
#include <stdlib.h>
#include <omp.h>

void myfunc(float *v, int D, float c) {
  int id,N,i;
  N = omp_get_num_procs();
  omp_set_num_threads(N);
  #pragma omp parallel private(id,i)
  {
    id = omp_get_thread_num();
    for (i=id; i<D; i=i+N) {
      v[i] = v[i] * c;
    }
  }
}
```





# Példaprogram - CUDA

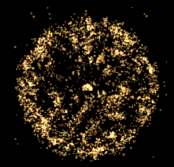
```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```



# Példaprogram - CUDA

```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```

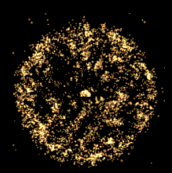
```
// var. deklarációk
// tömb hossza byte-ban
```



# Példaprogram - CUDA

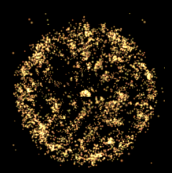
```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```

```
// mem. foglalás
// host-on és device-on
```



# Példaprogram - CUDA

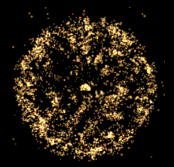
```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {                    // tömb inicializálás
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```



# Példaprogram - CUDA

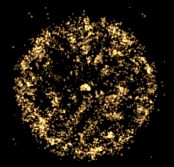
```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```

// tömb másolása device-ra



# Példaprogram - CUDA

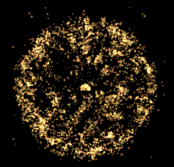
```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {                    // grid definiálás
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```



# Példaprogram - CUDA

```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```

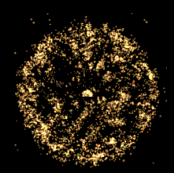
```
// kernel aszinkron futtatása
//  device-on
// futtatási konfig. megadása
```



# Példaprogram - CUDA

```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```

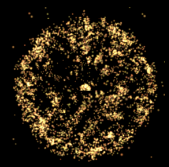
```
// eredmény visszamásolása
```



# Példaprogram - CUDA

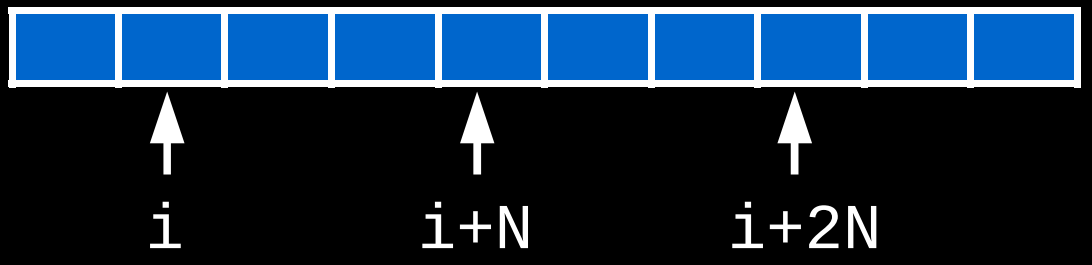
```
__global__ void mykernel(float *v, int D, float c) {…}

int main(void) {
  int i, D=1000000, s;
  float *h, *d;
  s = D * sizeof(float);
  cudaMallocHost((void**) &h, s);
  cudaMalloc( (void**) &d, s) ;
  for (i=0; i<D; i=i+1) { h[i] = (float)i; }
  cudaMemcpy( d, h, s, cudaMemcpyHostToDevice);
  dim3 grid     (30);
  dim3 threads  (32);
  mykernel <<< grid, threads >>> (d, D, 2.0f);
  cudaMemcpy( h, d, s, cudaMemcpyDeviceToHost);
  cudaFreeHost(h);  cudaFree(d);
}
```

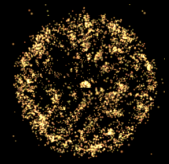
```
// memóriafelszabadítás
// host-on és device-on
```



# Példaprogram - CUDA

```
__global__ void mykernel(float *v, int D, float c)
{
  int id,N,i;

  id = blockIdx.x * blockDim.x + threadIdx.x ;
  N = gridDim.x * blockDim.x;

  for (i=id; i<D; i=i+N) {
    v[i] = v[i] * c;
  }
}


int main(void) {
   ...
}
```

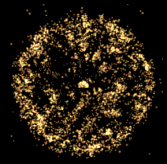




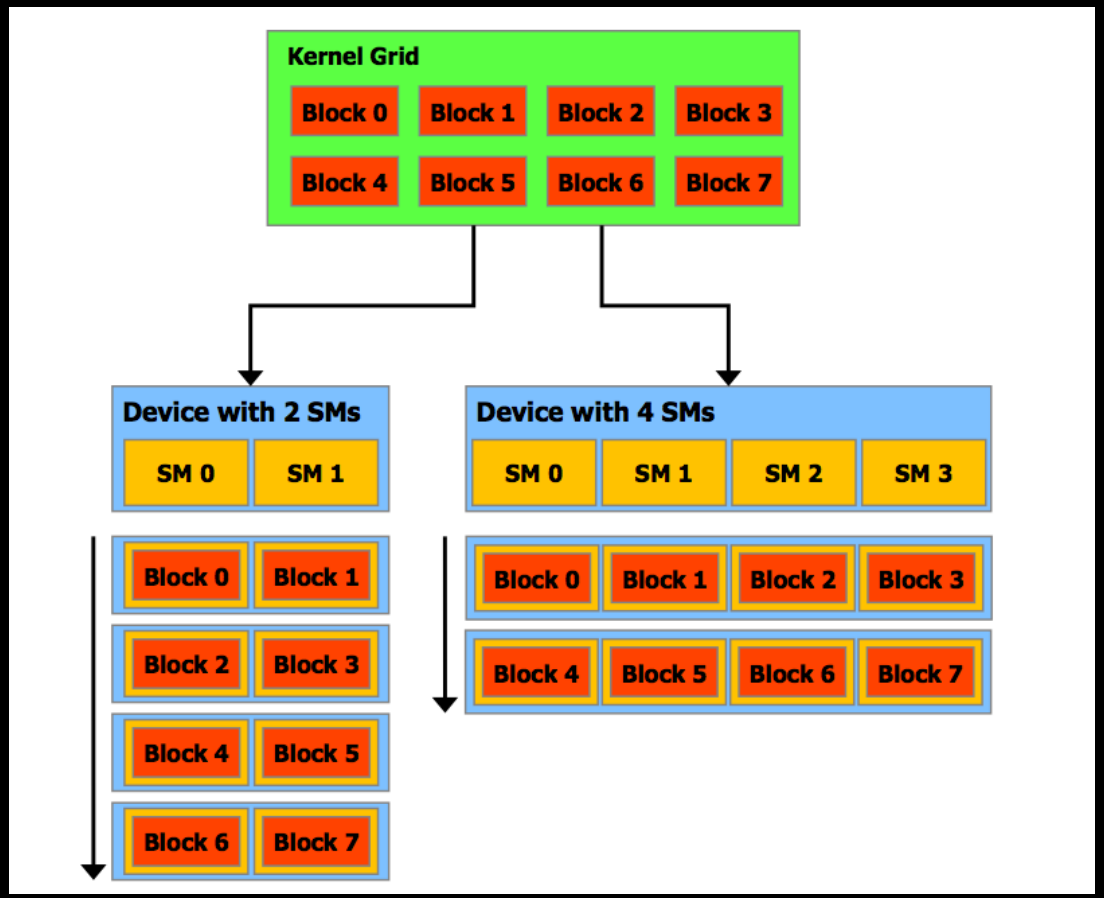
# Példaprogramok

Fordítás:

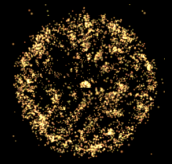
```
gcc -o testserial.out testserial.c

gcc -o testomp.out testomp.c -fopenmp -lgomp

nvcc -o testcuda.out testcuda.c
```



# Fizikai végrehajtás

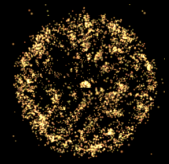



- Egy Blockot egy MP számol
- Minden Block SIMD csoportokra van osztva:
  **warp**
  a warp threadjei fizikailag egyszerre hajtódnak végre
- a warp (ma) 32 szálból áll (0-31,32-63,...)



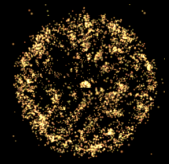
# Aritmetika időigénye

- 4 clock cycles:
  - Floating point: add, multiply, fused multiply-add
  - Integer add, bitwise operations, compare, min, max
- 16 clock cycles:
  - reciprocal, reciprocal square root, `__log(x)`, 32-bit integer multiplication
- 32 clock cycles:
  - `__sin(x)`, `__cos(x)` and `__exp(x)`
- 36 clock cycles:
  - Floating point division (24-bit version in 20 cycles)
- Particularly costly:
  - Integer division, modulo
  - Remedy: Replace with shifting whenever possible
- Double precision (when available) will perform at half the speed

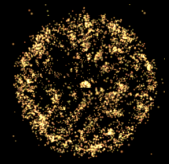


# Következmények

- # Block / # MP > 1
  minden MP számol
- # Block / # MP > 2
  egy MP egyszerre több Blockot is számol
- # Block / # MP > 100
- egy Block erőforrás-igénye ≤ MP teljes erőforrása
  shared memória, register
  egy MP egyszerre több Blockot is számolhat
- egy warpon belüli thread-divergencia kerülendő
  a különböző ágak sorbarendezve hajtódnak végre



# Tippek

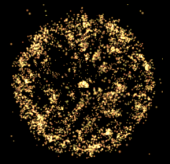


# Tippek

- nvcc flagek:
  - -emu : emuláció CPU-n (-> printf,debug)
  - -keep : megőrzi a fordítás közbenső fájljait
  - -arch sm_13 ; -arch compute_13 : double support
  - -arch sm_10 ; -arch compute_10 : lekorlátozás

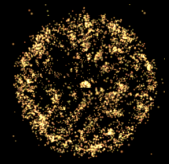
- Több végrehajtási konfiguráció (számítási háló, computational grid) kipróbálása (gridDim, blockDim)

- Hibakezelés (ld. Supercomputing for the masses)



# Tippek

- Profiler (cudatoolkit)
- Debugger (cudatoolkit)
- Kártyalefagyás esetén:
  kill -9 <application PID>
  várni perceket
  root-ként: rmmod nvidia ; modprobe nvidia
- lib készítése (ld.: példaprogramok)

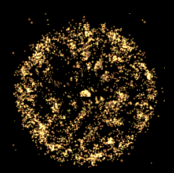


# Tippek: időmérés

**GPU-n:**

```
cudaEvent_t start, stop;

cudaEventCreate( &start ) ;
cudaEventCreate( &stop ) ;

cudaEventRecord( start, 0 );

{ … }

cudaEventRecord( stop, 0 );

cudaEventElapsedTime( &elapsedTimeInMs, start, stop);
```

**ld. pl. bandwidthTest.cu forráskód (SDK)**

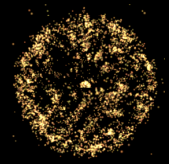


# Tippek: időmérés

**CPU-n:**

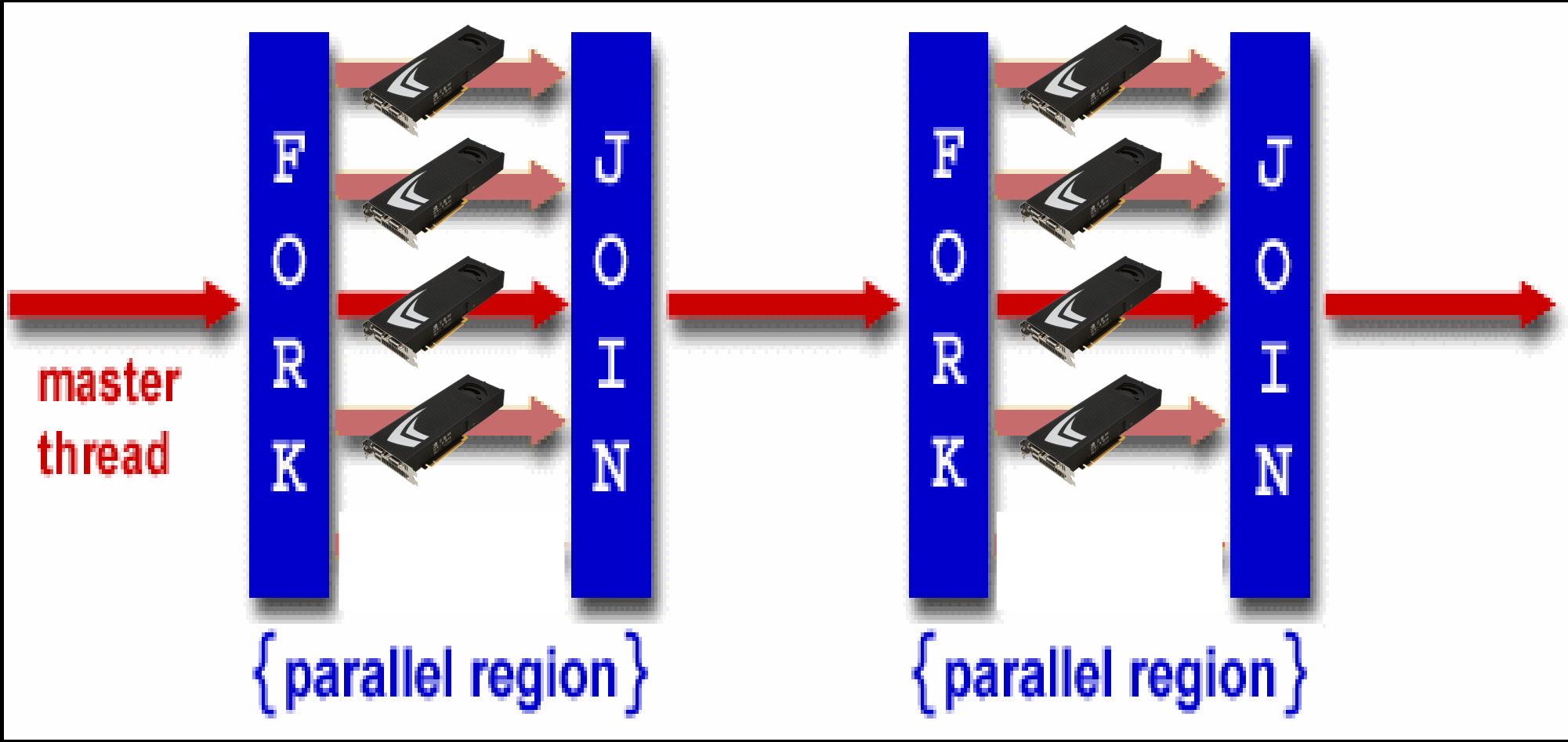
```
struct timespec start, end;
clock_gettime(CLOCK_REALTIME, &start);
{ … }
cudaThreadSynchronize();   //non-blocking GPU kernel!
clock_gettime(CLOCK_REALTIME, &stop);

time_elapsed = (stop.tv_sec - start.tv_sec) +
(double)(stop.tv_nsec - start.tv_nsec) / 1000000000 );
```

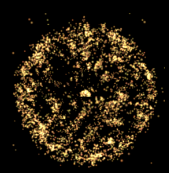
Megj: GPU kernel aszinkron végrehajtás ->
CPU/GPU párhuzamos számolás



# Tippek: több kártya egy gépben

## OpenMP + CUDA

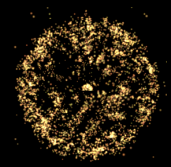




# Tippek: több kártya egy gépben

```
int num_gpus, gpu_id, cpu_thread_id, num_cpu_threads;

cudaGetDeviceCount(&num_gpus);
omp_set_num_threads(num_gpus);

#pragma omp parallel private(cpu_thread_id,gpu_id)
{
  cpu_thread_id = omp_get_thread_num();
  num_cpu_threads = omp_get_num_threads();

  cudaSetDevice(cpu_thread_id % num_gpus);
  cudaGetDevice(&gpu_id);

  dim3 gpu_threads(…);
  dim3 gpu_blocks(…);
  cuda_kernel <<< gpu_blocks,gpu_threads >>> (d,x);
}
```

https://www.wiki.ed.ac.uk/display/ecdfwiki/Use+multiple+GPU+devices+with+OpenMP+and+CUDA



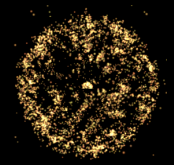
# Példa: mátrixösszeadás

## CPU Program

```
void add_matrix
  ( float* a, float* b, float* c, int N ) {
  int index;
  for ( int i = 0; i < N; ++i )
    for ( int j = 0; j < N; ++j ) {
      index = i + j*N;
      c[index] = a[index] + b[index];
    }
}

int main() {
  add_matrix( a, b, c, N );
}
```

## CUDA Program

```
__global__ add_matrix
  ( float* a, float* b, float* c, int N ) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int index = i + j*N;
  if ( i < N && j < N )
    c[index] = a[index] + b[index];
}

int main() {
  dim3 dimBlock( blocksize, blocksize );
  dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
  add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```



# Példa: mátrixösszeadás

## CPU Program

```
void add_matrix
  ( float* a, float* b, float* c, int N ) {
  int index;
  for ( int i = 0; i < N; ++i )
    for ( int j = 0; j < N; ++j ) {
      index = i + j*N;
      c[index] = a[index] + b[index];
    }
}

int main() {
  add_matrix( a, b, c, N );
}
```

## CUDA Program

```
__global__ add_matrix
  ( float* a, float* b, float* c, int N ) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int index = i + j*N;
  if ( i < N && j < N )
    c[index] = a[index] + b[index];
}

int main() {
  dim3 dimBlock( blocksize, blocksize );
  dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
  add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```