OPENCL-ALAPÚ PÁRHUZAMOSÍTÁS HATÁSA A KONSTANS SZOMSZÉDOK MÓDSZERÉRE, ÖSSZEVETÉS AZ EXPLICIT EULER-MÓDSZERREL

Koics Dániel[1], Kovács Endre[2], Nehéz Károly[3]
[1]tanszéki mérnök, PhD hallgató, [2,3]egyetemi docens
[1]*Automatizálási és Infokommunikációs Intézet, Gépészmérnöki és Informatikai Kar*
[2]*Fizikai és Elektrotechnikai Intézet, Gépészmérnöki és Informatikai Kar*
[3]*Informatikai Intézet, Gépészmérnöki és Informatikai Kar*
[1]daniel.koics@uni-miskolc.hu, [2]endre.kovacs@uni-miskolc.hu,
[3]karoly.nehez@uni-miskolc.hu

ABSZTRAKT

Célom annak vizsgálata, milyen hatással van az *Open CL* [13][14] által történő párhuzamosítás a konstans szomszédok (*CNe*) módszerének [8] futásidejére. A mérési adatok egyelőre csak az explicit *Euler*-módszer adataival kerültek összevetésre. A későbbiekben szükséges lehet a módszer soros megvalósításának megmérése is, hiszen a párhuzamos és soros működésű változatokat egyazon algoritmus kategóriáján belül célszerű összehasonlítani. A félév további célja a *CpC* módszer [9] vizsgálata is, de az előadás idején ezen a területen még nincsenek eredményeim.

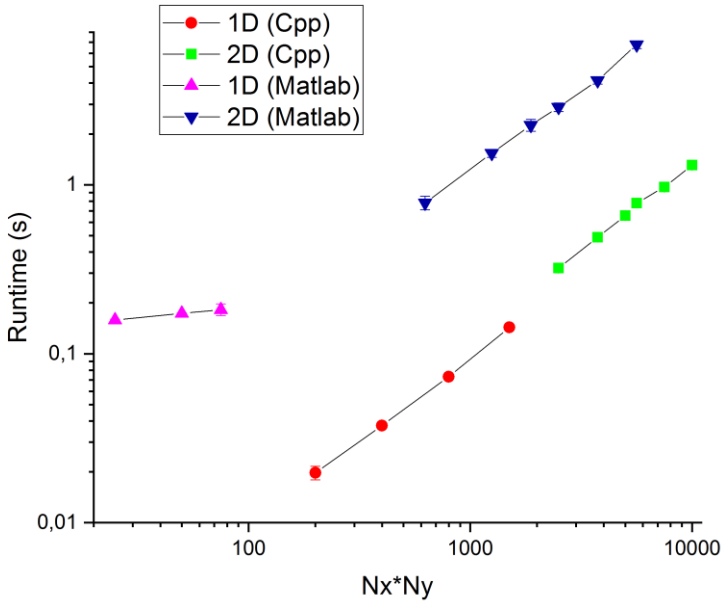
Ennek érdekében a 0. fejezetben összefoglalom korábbi munkáimat, a 0. szakaszban részletesebben ismertetem a vizsgált numerikus módszereket, a 0. pontban pedig a jelen vizsgálat során alkalmazott skálázásokat és kezdőérték-problémákat, valamint a mérés egyéb körülményeit. Az 0. fejezetben olvashatók a vizsgálat eredményei, végül a 0. szakaszban zárom gondolataimat a konklúzióval, ezután a 0. és 0. pontban csupán az irodalomjegyzék, illetve egy a helyszűke miatt a korábbiakból kimaradt példa programkód található meg.

1. KORÁBBI MUNKÁMRÓL

PhD (korábban *MSc*) tanulmányaim során a hővezetés vagy diffúzió egyenletének megoldására képes algoritmusokat vizsgálom. Célom a témavezetőm és kutatócsoportja által kifejlesztett, valamint korábban is ismert különböző módszerek párhuzamosított változatának implementálása és performanciatesztelése. Az előző félévek során csupán az explicit *Euler*-módszert vizsgáltam, előbb különböző *MATLAB* alapú megközelítésmódokat, majd pedig górcső alá vettem a *C++* alapú natív kódok performanciáját is [3]. A megelőző félév egy programozástechnikai segédeszköz, az *Open CL* [15] megismeréséről szólt [4]. Ennek a segítségével olyan párhuzamos működésű programok írhatók, melyek GPU-n vagy egyéb hardveres gyorsítóeszközökön is futtathatók. Rengeteg technikai nehézséget rejt magában az eszköz használata, melynek projektembe történő integrációja után elvégeztem a mérést, melyben a párhuzamosítás *Euler* módszerére történő hatását vizsgáltam.

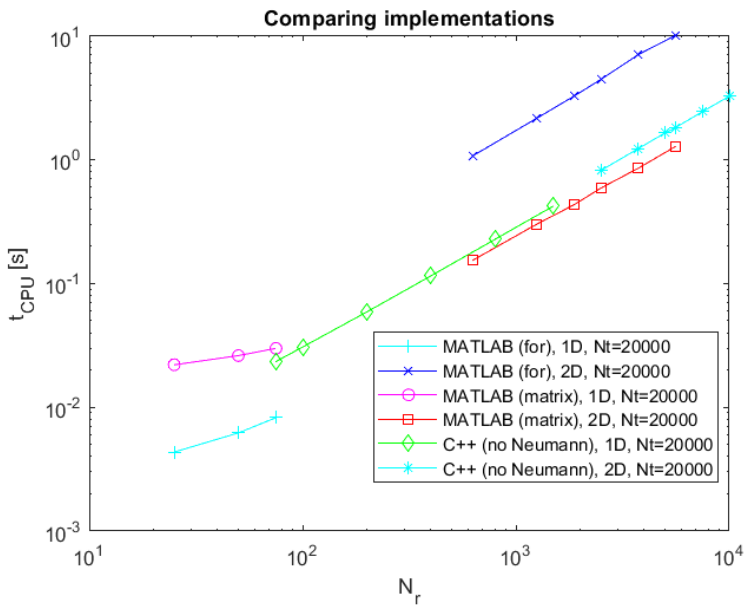
A diplomamunkám [1] során 2 dolgot tapasztaltam. Az egyik az, hogy a mátrixalgebra alkalmazásával a futásidő csupán csak 2 dimenzióban csökkenthető a `for` ciklus alapú megvalósításhoz képest, mégpedig akár 5-ödrészére is. 1 dimenzióban azonban a futásidő akár 5-szörösre is nőhet.

A másik megfigyelés, hogy a *C++* változat valóban akár 10-szer gyorsabb a *MATLAB* mátrix változathoz képest, bár az ábrán látható, hogy 1 dimenzióban a *C++* változat időigénye meredekebben nő a rendszerméret növelésével, mint a *MATLAB* változaté, vagyis egy ponton jó eséllyel utoléri:



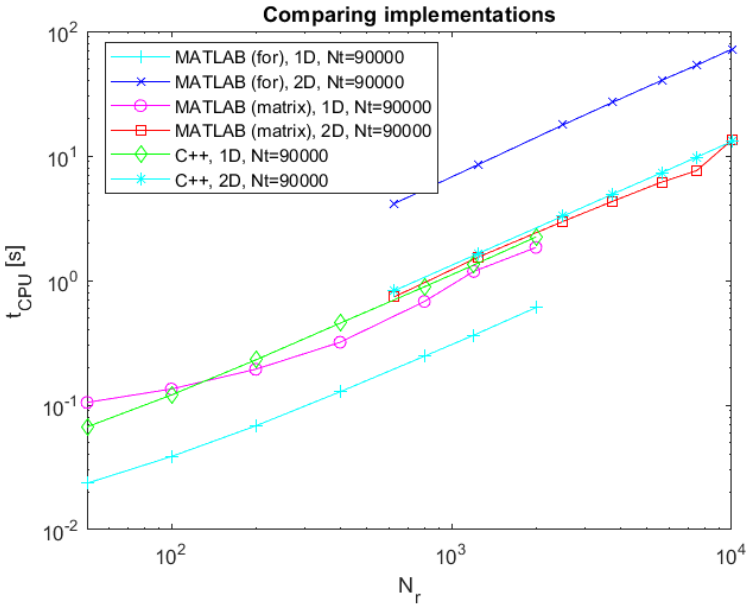
1. ábra
C++ és tisztán *MATLAB* kód futásidejének összevetése
70000 időlépésszám mellett a diplomamunkámban

Később azonban a *PhD* tanulmányaim első félévében [2] utóbbiaknak némileg ellentmondó eredményeket kaptam. Célom az lett volna, hogy a mérést megismételjem olyan módon, hogy ugyanazt a skálázást használjam minden implementáció esetében (3. ábra), valamint a térbeli skálák kezdő és végpontjai közti különbségek legyenek állandók (a diplomamunkám során a lépésközt tartottam állandó értéken). Eredményeimet szerettem volna egy helyi folyóiratban publikálni. Sajnos ezúttal a *C++* és a *MATLAB* mátrix alapú implementáció között nem adódott lényegi eltérés a futásidőt tekintve, még akkor sem, ha ugyanazokat a skálaparamétereket és ugyanazt a számítógépet használtam, mint korábban:



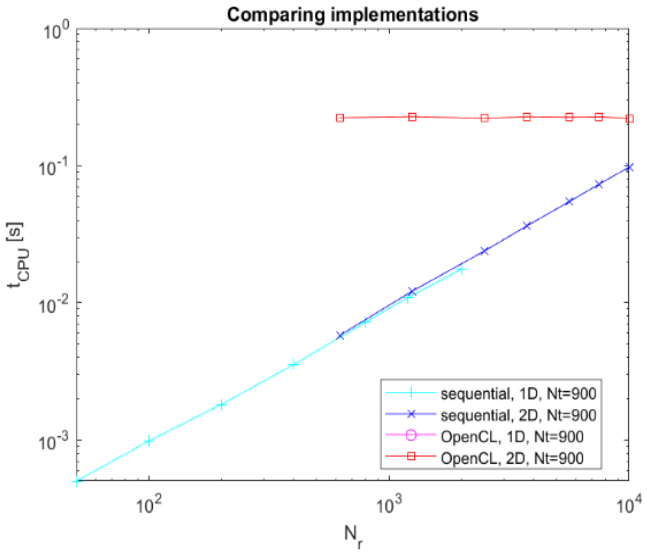
2. ábra
A *C++* és a 2-féle *MATLAB* kód futásidejének összevetése 70000 időlépésszám mellett az 1. *PhD* félévben, de a diplomamunkám során használt skálákkal és számítógépen

Az látható, hogy a *PhD* félévben készült mérés esetében a *MATLAB*-alapú számítások eleve gyorsabban futottak, mint korábban a diplomamunka idején, nagyjából olyan gyorsan, mint a *C++* alapú számítás. A két mérés közti eltérés eredete máig nem tisztázott. Végül csak a *MATLAB*-alapú számítások performanciája került publikálásra a [3] folyóiratcikkben:



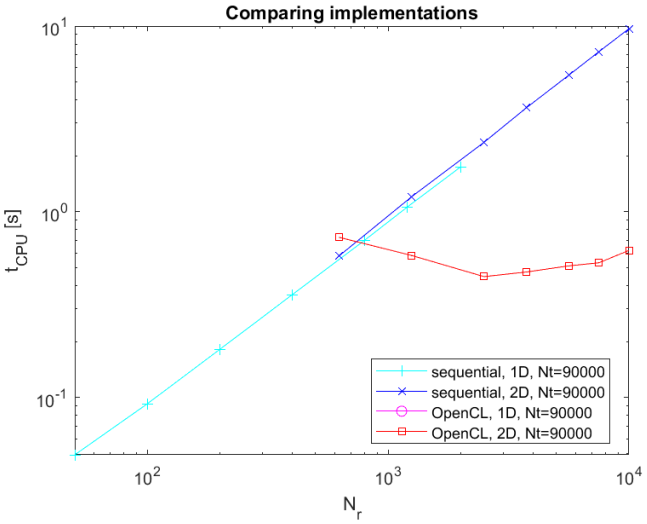
3. ábra
A *C++* és a 2-féle *MATLAB* kód futásidejének összevetése 90000 időlépésszám mellett az 1. *PhD* félévben, immár az új, azonos teljes térbeli tartományt lefedő skálákkal

A második – a jelenlegit megelőző – *PhD* félévben [4] az *Open CL* alapú párhuzamosítás vizsgálata során azt tapasztaltam, hogy az explicit *Euler*-módszer futásidő igénye csak kellően nagy rendszerméret esetén javul, mivel a futásidő – legalábbis az általam vizsgált rendszerméretek esetén – lényegében független a rendszermérettől (valószínűleg kellőképpen nagy rendszerméret már befolyásoló tényező lenne):



4. ábra
Soros megvalósítás és *Open CL* által történő párhuzamosítás futásidejének összevetése 900 időlépéses futtatások esetén

Viszont a futásidő továbbra is függ az időlépések számától, így az a rendszerméret, mely felett már érdemes a párhuzamosítást választani, nagyobb időlépésszám esetén előbb (már kisebb rendszerméretnél) eljön:



5. ábra
Soros megvalósítás és *Open CL* által történő párhuzamosítás futásidejének összevetése 90000 időlépéses futtatások esetén

Megjegyzendő azonban, hogy a 4. ábraról és az 5. ábraáról hiányzik az 1-dimenziós *Open CL*-mérések eredménye. Azokban az esetekben ugyanis a futásidőre egzaktul nulla futásidőt kaptam, ami nyilvánvalóan programhibát jelent. Az azóta eltelt időben rájöttem, hogy a problémát a *MATLAB* `tic-toc` függvényének hibás használata okozta.

2. A VIZSGÁLAT TÁRGYA

Mint az már a bevezetőben említettem, jelen félévben vizsgálódásom tárgyát a konstans szomszédok (***C**onstant **Ne**ighbours*) módszere [8] képezi. A fórumelőadás idejére sikerült elvégezni a módszer *Open CL*-alapú implementációjának performanciamérését. Azonban még nem készítettem el a módszer natív *C++* alapú, soros implementációját, mely jó összehasonlítási alapot jelentene az *Open CL* és a párhuzamosítás hatásának vizsgálatához.

A következőkben a 0 alfejezetben ismertetem a hővezetés egyenletét, majd a 0 és a 0 szakaszban az összehasonlításra kerülő *Euler* és *CNe* módszereket, a 0 fejezetben pedig ezek *Open CL* által történő párhuzamosítását. Végül a 0 szakaszban említést teszek a *CpC* módszerről, mely egyelőre nem produkált értékelhető számítási eredményt.

2.1 A hővezetés/diffúzió egyenlete

A szakirodalomban gyakran találkozunk a hővezetés vagy diffúzió egyenletével, mely matematikailag ugyanolyan formájú:

$$\frac{\partial f(\boldsymbol{r},t)}{\partial t} = D \cdot \Delta f(\boldsymbol{r},t) \tag{1}$$

ahol f a hőmérséklet [°C, K]
vagy diffúzió esetén a koncentráció [m^{-3}, mol/m^3, kg/m^3]
D a diffúziós együttható [m^2/s]
$\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ a Laplace-operátor (matematikai rövidítés)

Kifejtve a Laplace-operátort:

$$\frac{\partial f(\boldsymbol{r},t)}{\partial t} = D \cdot \left(\frac{\partial^2 f(\boldsymbol{r},t)}{\partial x^2} + \frac{\partial^2 f(\boldsymbol{r},t)}{\partial y^2} + \frac{\partial^2 f(\boldsymbol{r},t)}{\partial z^2}\right) \tag{1.b}$$

2.2 Az explicit Euler-módszer

Euler módszere az

$$\frac{\partial f(\boldsymbol{r},t)}{\partial t} = \widetilde{\mathrm{D}} f(\boldsymbol{r},t) \tag{2}$$

formátumú differenciálegyenletek megoldására használható, ahol $\widetilde{\mathrm{D}}$ egy térkoordináták szerinti deriválásokat tartalmazó differenciál operátor.

A módszer az ún. véges differencia módszerek közé tartozik, emiatt megköveteli a probléma diszkretizálását mind térben, mind időtartományon. Bevezetve a térbeli koordinátatengelyek kezdőpontjaira az X_{init}, Y_{init} és Z_{init}, a végpontokra az X_{fin}, Y_{fin} és Z_{fin} jelölést, továbbá az időtartomány kezdő és végpontjára a T_{init} és T_{fin} jelölést:

$$\boldsymbol{r}_{i,j,k} = \begin{bmatrix} X_{\mathrm{init}} + i \cdot \Delta x \\ Y_{\mathrm{init}} + j \cdot \Delta y \\ Z_{\mathrm{init}} + k \cdot \Delta z \end{bmatrix} = \begin{bmatrix} X_{\mathrm{init}} \\ Y_{\mathrm{init}} \\ Z_{\mathrm{init}} \end{bmatrix} + [\Delta x \quad \Delta y \quad \Delta z] \cdot \begin{bmatrix} i \\ j \\ k \end{bmatrix}, \tag{3.a}$$

$$t_n = T_{\mathrm{init}} + n \cdot \Delta t \tag{3.b}$$

ahol:

$$i \in \{1,2 \dots N_{\mathrm{x}}\},$$
$$j \in \{1,2 \dots N_{\mathrm{y}}\},$$
$$k \in \{1,2 \dots N_{\mathrm{z}}\}, \tag{4.a}$$

és

$$n \in \{0,1,2 \dots N_{\mathrm{t}}\}, \tag{4.b}$$

a bejárt indextartományok, valamint:

$$\Delta x = (X_{\mathrm{fin}} - X_{\mathrm{init}})/(N_{\mathrm{x}} - 1),$$
$$\Delta y = (Y_{\mathrm{fin}} - Y_{\mathrm{init}})/(N_{\mathrm{y}} - 1),$$
$$\Delta z = (Z_{\mathrm{fin}} - Z_{\mathrm{init}})/(N_{\mathrm{z}} - 1), \tag{5.a}$$

és

$$\Delta t = (T_{\mathrm{fin}} - T_{\mathrm{init}})/N_{\mathrm{t}}, \tag{5.b}$$

a térbeli és időbeli lépésközök. A vizsgált függvény, illetve a (2) egyenlet jobb oldala ekkor a diszkrét mintavételi pontokban vett értékekkel közelíthető:

$$f(\boldsymbol{r},t) \cong f(\boldsymbol{r}_{i,j,k}, t_n) =: f_{i,j,k}^{(n)}, \tag{6}$$

$$\widetilde{\mathrm{D}} f(\boldsymbol{r},t) \cong \widetilde{\mathrm{D}} f(\boldsymbol{r}_{i,j,k}, t_n) =: \left[\widetilde{\mathrm{D}} f\right]_{i,j,k}^{(n)}. \tag{6.b}$$

Euler szerint a soron következő időmintavételi pontban a függvény értéke a megelőző értékek felhasználásával explicit módon közelíthető:

$$f_{i,j,k}^{(n+1)} \approx f_{i,j,k}^{(n)} + \left[\widetilde{\mathrm{D}} f\right]_{i,j,k}^{(n)} \cdot \Delta t. \tag{7}$$

A módszer lényege, hogy az időlépésen belül lineáris időfüggést tételezünk fel, melynek meredekségét a léptetés előtti állapotból számítjuk ki a (2) egyenlet jobb oldala segítségével.

A hővezetés/diffúzió esetében az egyenlet jobb oldalán lévő kifejezés:

$$\widetilde{\mathrm{D}}\, f(\boldsymbol{r},t) = D \cdot \Delta f(\boldsymbol{r},t) = D \cdot \left(\frac{\partial^2 f(\boldsymbol{r},t)}{\partial x^2} + \frac{\partial^2 f(\boldsymbol{r},t)}{\partial y^2} + \frac{\partial^2 f(\boldsymbol{r},t)}{\partial z^2}\right), \qquad (8)$$

melyet az ún. *centrális differenciálformulá*val szokás közelíteni:

$$\left[\widetilde{\mathrm{D}}\, f\right]_{i,j,k}^{(n)} = \left[D \cdot \Delta f\right]_{i,j,k}^{(n)} = D,$$
$$\cdot \left(\frac{f_{i+1,j,k}^{(n)} - 2\cdot f_{i,j,k}^{(n)} + f_{i-1,j,k}^{(n)}}{\Delta x^2} + \frac{f_{i,j+1,k}^{(n)} - 2\cdot f_{i,j,k}^{(n)} + f_{i,j-1,k}^{(n)}}{\Delta y^2} + \frac{f_{i,j,k+1}^{(n)} - 2\cdot f_{i,j,k}^{(n)} + f_{i,j,k-1}^{(n)}}{\Delta z^2}\right). \qquad (9)$$

Így a léptető formula – az egyszerűség kedvéért 2-dimenziós esetre alkalmazva:

$$f_{i,j}^{(n+1)} \approx f_{i,j}^{(n)} + D \cdot \Delta t,$$
$$\cdot \left(\frac{f_{i+1,j}^{(n)} - 2\cdot f_{i,j}^{(n)} + f_{i-1,j}^{(n)}}{\Delta x^2} + \frac{f_{i,j+1}^{(n)} - 2\cdot f_{i,j}^{(n)} + f_{i,j-1}^{(n)}}{\Delta y^2}\right). \qquad (10)$$

2.3. A konstans szomszédok módszere [8]

Az általam vizsgált módszer nevét onnan kapta, hogy – az explicit Euler-módszerrel ellentétben – az egyes időlépések során nem a vizsgált cella hőmérsékletének idő szerinti deriváltját, hanem a szomszédos cellák hőmérsékletét tekinti állandónak. Az időlépés során a vizsgált cella hőmérsékletének időfüggését nem lineáris, hanem – a *Newton*-féle lehűlési/felmelegedési törvényének megfelelő – exponenciális függvénnyel közelíti, ami a módszernek feltétel nélküli stabilitást kölcsönöz:

$$f_{i,j}^{(n+1)} = \exp\left\{-\frac{\Delta t}{\tau}\right\} \cdot f_{i,j}^{(n)} + \left(1 - \exp\left\{-\frac{\Delta t}{\tau}\right\}\right) \cdot f_{i,j}^{(n,\infty)}, \qquad (11)$$

ahol:
$$\tau = 1 \Big/ \left[2D \cdot \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\right], \qquad (11.b)$$

és
$$f_{i,j}^{(n,\infty)} = \frac{1}{2} \cdot \left(\frac{f_{i+1,j}^{(n)} - 2\cdot f_{i,j}^{(n)} + f_{i-1,j}^{(n)}}{\Delta x^2} + \frac{f_{i,j+1}^{(n)} - 2\cdot f_{i,j}^{(n)} + f_{i,j-1}^{(n)}}{\Delta y^2}\right) \Big/ \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right). \qquad (11.c)$$

Látható, hogy a modell szerint a hőmérséklet egy (τ időállandójú) exponenciális lecsengés/felfutás által tart egy az – állandónak tekintett – szomszédok által meghatározott $f_{i,j}^{(n,\infty)}$ hőmérsékletértékhez, s az új érték pedig e függvény értéke a Δt idő eltelte után. Mivel az új érték mindig a kezdeti $f_{i,j}^{(n)}$ hőmérséklet és az $f_{i,j}^{(n,\infty)}$ határérték közé esik, a módszer stabilitása garantált.

A formula levezetéséhez szükség van az említett Newton-féle lehűlési törvény ismeretére:

$$\frac{d\, f(t)}{dt} = K \cdot (f(t) - f^{\infty}), \qquad (12)$$

ahol f^{∞} a környezet vagy a kölcsönhatásban részt vevő másik, nagy hőkapacitású test (jelen esetben a szomszéd cellák) állandó hőmérséklete, míg K a test és környezete kölcsönhatására jellemző fizikai állandó. A mi esetünkben – 2 dimenziót feltételezve – a cella 4 szomszéddal áll kölcsönhatásban, ezért az egyenlet jobb oldalán az egyes szomszédokra vonatkozó tagok összege jelenik meg, mégpedig x irányú szomszédok esetén $K = D/\Delta x^2$, y irányú szomszédok esetén pedig $K = D/\Delta y^2$ hőátadási tényezővel:

$$\frac{d f_{i,j}(t)}{dt} = -\frac{D}{\Delta x^2} \cdot \left(f_{i,j}(t) - f_{i+1,j}\right) - \frac{D}{\Delta x^2} \cdot \left(f_{i,j}(t) - f_{i-1,j}\right),$$
$$-\frac{D}{\Delta y^2} \cdot \left(f_{i,j}(t) - f_{i+1,j}\right) - \frac{D}{\Delta y^2} \cdot \left(f_{i,j}(t) - f_{i-1,j}\right), \qquad (13.a)$$

röviden:
$$\frac{d f_{i,j}(t)}{dt} = D \cdot \left(\frac{f_{i+1,j} - 2 \cdot f_{i,j}(t) + f_{i-1,j}}{\Delta x^2} + \frac{f_{i,j+1} - 2 \cdot f_{i,j}(t) + f_{i,j-1}}{\Delta y^2}\right), \qquad (13.b)$$

ahol $f_{i+1,j}$, $f_{i-1,j}$, $f_{i,j+1}$ és $f_{i,j-1}$ a szomszédok állandónak tekintett értékei (vagyis rendre a $f_{i+1,j}^{(n)}$, $f_{i-1,j}^{(n)}$, $f_{i,j+1}^{(n)}$ és $f_{i,j-1}^{(n)}$ értékek). Ennek az egyenletnek a $f_{i,j}(t = 0) = f_{i,j}^{(n)}$ kezdőfeltételhez tartozó megoldása az exponenciális függvény, melybe $t = \Delta t$-t helyettesítve adódik az (11) időléptető formula.

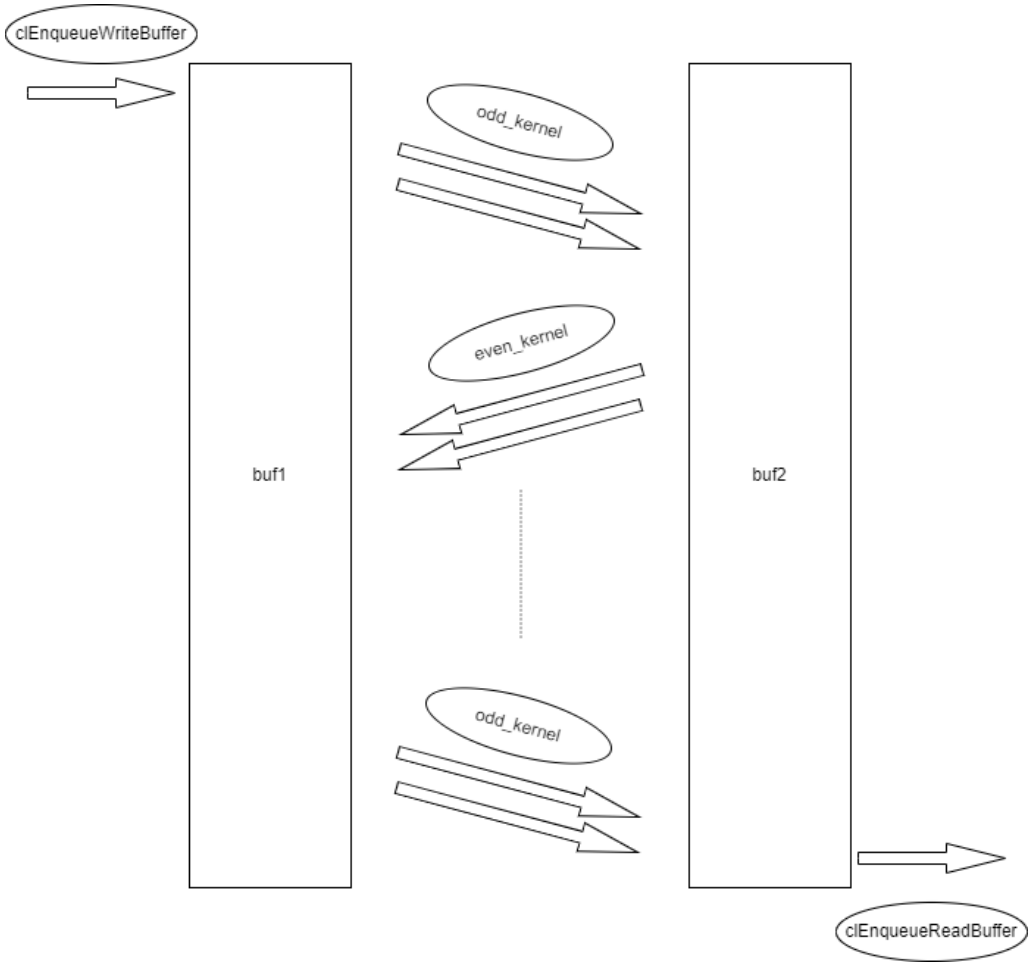
3. A MÓDSZER IMPLEMENTÁLÁSA OPEN CL HASZNÁLATÁVAL [5]

Az *Open CL* egy – a *Khronos Groups* által megalkotott – specifikáció, melynek célja lehetővé tenni párhuzamos működésű programkódok fejlesztését a modern számítógépekbe beépíthető különféle hardveres gyorsítóeszközökre. Segítségével komoly gyorsulás érhető el. Egyrészt meghatároz egy *C/C++* alapú programozási nyelvet, melyet használva lefejleszthetjük a gyorsítóeszközön – a mi esetünkben a GPU-n – futtatni kívánt programrészt – az ún. kernel kódot. Másrészt specifikál egy API-t, melynek segítségével a CPU-n futó programrész – a gazdakód – képes a kernel buildelésére, a gyorsító memóriájának menedzselésére, a kernel függvényeinek parametrizálására és a menedzselt memóriaterületekhez történő hozzárendelésére, valamint a felparametrizált kernelfüggvények hívásának ütemezésére, továbbá gazdakód és kernel közti szinkronizálási pontok definiálására. A kernelhívás ütemezésekor indextartományt kell megadni, a kernelfüggvény minden egyes indexre, párhuzamos módon kerül meghívásra, s lehetősége van az aktuális index lekérdezésére. További érdekessége az *Open CL*-nek a vektortípusok (`float2`, `float3`, `float4`, `float8`, `float16`, `double2`, `double3`, `double4`, `double8`, `double16`…), melyek több azonos típusú változó elemi műveletként történő feldolgozását teszik lehetővé. Bár a szabványt a *Khronos Groups* tartja fenn, az API-t, valamint a kernel buildeléséhez szükséges C/C++ fordítót az egyes gyorsítók gyártói maguk implementálják. Jómagam az *Open CL* 3. verziójának *Intel* implementációját használom. Telepítéséről és konfigurálásáról bővebben az előző féléves beszámolóban lehet olvasni [4].

3.1. Explicit Euler-módszer

Bár az *Euler*-módszer párhuzamosítása az előző félév feladata volt [4], az egyértelműség kedvéért ismét összefoglalom az akkor alkalmazott alapelveket.

Az explicit Euler-módszer esetében adott időlépésen belül a különböző adatpontok egymással párhuzamosan számíthatók. Ezért létrehoztam egy kernelfüggvényt, mely egy adatpont egyetlen időlépésének kiszámítására képes. Mivel az új érték függ a szomszédos adatpontok régi értékétől, fontos, hogy a párhuzamos számítás során az egyes cellák számítása ne változtassa meg egymás szomszédjait idő előtt, ezért az új értékeket egy másik memóriaterületen tárolni, mint a régieket. Ezért két memóriatömböt foglaltam le a GPU-n, a kernelt pedig kétféleképp parametrizáltam fel: Páratlan időlépésben az első tömböt használva forrásként a második tömbbe, míg páros időlépés esetén a második tömböt használva forrásként az első tömbbe kerül az eredmény.



6. ábra
Euler módszerének működése az *Open CL* keretei közt

Az alkalmazott kernelfüggvény 2-dimenziós esetben (részlet a [5] repozitórium tartalmából):

```
__kernel void calc_cell_2D(__global const real *before,
__global real *after,
      real2 coeffs, uint2 size, long4 neighbours) {
   const size_t index = get_global_id(0) + size.x *
get_global_id(1);
   __global const real *old_cell = before + index;
   after[index] = (1 - 2 * (coeffs.x + coeffs.y)) * *old_cell
      +coeffs.x * (old_cell[neighbours.even.x] +
old_cell[neighbours.odd.x])
      +coeffs.y * (old_cell[neighbours.even.y] +
old_cell[neighbours.odd.y]);
}
```

ahol `real` és `real2` a `float` és `float2` vagy a `double` és `double2` típus álnevei attól függően, hogy támogatott-e a 64-bites lebegőpontos architektúra:

```
#ifdef cl_khr_fp64
```

```
    #warning "Using Khronos-style double precision floating
points"
    #pragma OPENCL EXTENSION cl_khr_fp64 : enable
    typedef double real;
    typedef double2 real2;
    typedef double3 real3;
#elif defined cl_amd_fp64
    #warning "Using AMD-style double precision floating
points"
    #pragma OPENCL EXTENSION cl_amd_fp64 : enable
    typedef double real;
    typedef double2 real2;
    typedef double3 real3;
#else
    #warning "Double precision floating points not supported.
Defaulting to single precision mode."
    typedef float real;
    typedef float2 real2;
    typedef float3 real3;
#endif
```

Ennek buildelésére és kétféle parametrizálására a gazdakódban a 0. kódjegyzékben látható példa. A példát használva *Euler* ciklusa (*Dirichlet*-féle határfeltétel mellett) a következőképp valósítható meg:

```
size_t offset[2] = {1, 1};
size_t size[2] = {scaling.Nx - 2, scaling.Ny - 2};
cl_int err = clEnqueueWriteBuffer(queue, buf1, CL_TRUE,
        0, scaling.nr_tot() * sizeof(double), data,
        0, NULL, NULL);
if (err) {
        printErr("Filling memory buffer", err);
        return;
}
cl_mem resbuf = buf1;
cl_kernel kernel = NULL;
for (size_t nt = 0; nt <= Nt; ++nt) {
        if (kernel != even_kernel) {
                kernel = even_kernel;
                resbuf = buf2;
        }
        else {
                kernel = odd_kernel;
                resbuf = buf1;
        }
```

```
        err = clEnqueueNDRangeKernel(queue, kernel,
                2, offset, size, NULL,
                0, NULL, NULL);
        if (err != CL_SUCCESS) {
                printErr("enqueuing kernel", err);
                return;
        }
}
err = clEnqueueReadBuffer(queue, buffer, CL_TRUE,
        0, scaling.nr_tot() * sizeof(double), data,
        0, NULL, NULL);
if (err != CL_SUCCESS) {
        printErr("copying back result from device", err);
        return;
}
```

ahol `double* data` a numerikus adattömbre mutató pointer.

A gyakorlatban mindemellett még szükség van a lefoglalt erőforrások felszabadítására (clReleaseKernel, clReleaseMem, clReleaseProgram, clReleaseQueue, clReleaseContext, clReleaseDevice…) is.

Emellett megjegyzendő, még, hogy az [5] repozitóriumban kicsit más szemlélettel oldottam meg a problémát:

– Létrehoztam egy statikus könyvtárat, melyen keresztül az *Open CL*-t kezelem. Ide kerül a platform inicializálása (elérhető gyorsítók lekérdezése, GPU kiválasztása, 64-bites aritmetika támogatottságának ellenőrzése), továbbá a kernelkód menedzselését segítő függvények (a kernelkód fájlból történő beolvasása és buildelése, adott kernelfüggvény lekérdezése és parametrizálása), a hibadetektálás és a hibaüzenetek kiírásai, valamint az erőforrások nyomkövetése és felszabadítása.
– A számítást megvalósító programfájlban a kernelkód buildelését és a kernelfüggvények parametrizálását – *Euler* ciklusáról leválasztva – külön függvénybe szerveztem, s függvénypointerként adom át a ciklusnak, hogy az később más módszerhez is könnyen adaptálható legyen.
– Ezenkívül igyekeztem az erőforrásokat úgy menedzselni, hogy a kernelkódot elég legyen a program – pontosabban az algoritmusspecifikus kódokat tartalmazó *diffusionlib.dll* – betöltése után egyszer buildelni, majd a létrejött struktúrapointert (cl_program) – a GPU-t és a felépített kontextust reprezentáló struktúrapointerekkel (cl_device_id, cl_context) együtt – egy static tárolási osztályú változóban tárolom. Hogy ez biztosan ne okozzon memóriaszivárgást (pl. *MATLAB*-ból *MEX API*-n keresztül történő hívást követő `clear mex` vagy `clear all` parancs kiadásakor), a legújabb verzióban ezen erőforrások felszabadítását a *.dll* elengedésekor – vagyis a BOOL APIENTRY DllMain(HMODULE, DWORD, LPVOID) függvényből, annak DWORD ul_reason_for_call = DLL_PROCESS_DETACH értékkel történő meghívása esetén – kezdeményezem.

3.2. Konstans szomszédok (CNe) módszere

Összevetve a (10) és a (11) formulákat látható, hogy a *CNe* módszer esetén továbbra is lineáris a kapcsolat az új és a régi adatrács értékei közt, sőt ugyanaz a kernelfüggvény használható, mint az *Euler*-módszer esetében, csupán arra kell figyelni, hogy a kernelnek átadott coeffs együtthatóvektor számításakor a megfelelő képleteket használjuk CPU oldalon. Egyszerűbbnek láttam átalakítani a (11.b) és (11.c) képleteket:

$$\tau = \frac{1}{2Dk^2}, \tag{14.a}$$

és

$$f_{i,j}^{(n,\infty)} = \frac{1}{2k^2} \cdot \left(\frac{f_{i+1,j}^{(n)} - 2 \cdot f_{i,j}^{(n)} + f_{i-1,j}^{(n)}}{\Delta x^2} + \frac{f_{i,j+1}^{(n)} - 2 \cdot f_{i,j}^{(n)} + f_{i,j-1}^{(n)}}{\Delta y^2} \right), \tag{14.b}$$

ahol:

$$k^2 = \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}. \tag{14.c}$$

Ezalapján a coeffs vektor helyébe $-\ 0{,}5 \cdot D \cdot \Delta t \cdot [1/\Delta x^2\,;\quad 1/\Delta y^2]$ helyett $-\ 0{,}5 \cdot (1 - \exp\{-\Delta t/\tau\}) \cdot [1/\Delta x^2\,;\quad 1/\Delta y^2]$ helyettesítendő:

```
...
double D_times_dt = D * dt;
double k_square = 1 / pow(scaling.dx(), 2) + 1 / pow(scaling.dy(), 2);
double expramp = (1 - exp(-2 * D_times_dt * k_square)) / (2 * k_square);
cl_double2 coeffs;
coeffs.x = expramp / pow(scaling.dx(), 2);
coeffs.y = expramp / pow(scaling.dy(), 2);
cl_uint2 cl_size {scaling.Nx, scaling.Ny};
cl_long4 neighbours {
        -1, 1,  // x neighbours diff
        -(cl_long)scaling.Nx, (cl_long)scaling.Nx  // y neighbours diff
};
odd_kernel = clCreateKernel(program, name, &err);
if (err != CL_SUCCESS) {
        printErr("selecting Kernel for name", err);
        return;
}
...
```

4. A CPC MÓDSZER [9] VIZSGÁLATÁRÓL

A fentiek mellett vizsgálom még a módszer továbbfejlesztett, 2-iterációs változatát is, a *CpC* módszert, mely a konstans szomszédok módszerét egy másodrendű *Runge-Kutta* sémához hasonló módon alkalmazza. Bár az implementációt nagyjából elkészítettem, az előadás pillanatában a kódom értékelhetetlen számítási eredményt produkál, vagyis további hibakeresésre van szükség. Emiatt jelen írás ezzel kapcsolatos eredményeimet nem tartalmazza.

5. A MÉRÉS FUTTATÁSI KÖRÜLMÉNYEI

A teszteléshez használt, általam fejlesztett keretrendszert lásd az [6] repozitóriumban. A programhiba, mely miatt az előző félévben nem kaptam futásidő értékeket az 1-dimenziós esetekben, a tesztelő keretrendszerben volt, jelen félévben ezt javítottam.

Az alábbiakban először a 0 fejezetben a skálázási paramétereket és az azok mögött lévő megfontolásokat mutatom be, majd a 0 szakaszban a vizsgált kezdőérték problémákat, végül a 0 pontban ismertetem a vizsgálathoz használt számítógép műszaki paramétereit.

5.1 Skálázási paraméterek, stabilitás-vizsgálat

Bár jelen félévben *Euler* módszere mellett feltétel nélküli stabilitással bíró módszerrel (is) foglalkozom, az összehasonlíthatóság végett alábbiakban feltűntetem az *Euler*-módszer stabilitását is. A skálázási paramétereket az előző félévhez képest három szempontból módosítottam:

- Egyfelől az időlépésszám nagyobb értékekre felmegy, növelve ezzel a vizsgált esetek számát.
- Másfelől a vizsgált időintervallum hosszát, s ezáltal az időlépések méretét megnöveltem, hogy a szimuláció kimenete érdemben térjen el a kezdő adatsortól. (Ez korábban kritikus probléma volt.)
- A korábbiaktól eltérően most eltérő időskálát használok az 1-, illetve 2-dimenziós esetekben. Ezáltal – miközben a szimuláció kimenete 1 dimenzióban is eltér a kezdő adatsortól – a 2-dimenziós esetben is képes vagyok ugyanolyan sok, az *Euler*-módszer szempontjából instabil esetet vizsgálni.

5.2 Az 1-dimenziós eset

Az 1-dimenziós esetekben használt időskálák:

1. táblázat
Időskálák 1-dimenziós esetben

N_t	Δt	T_{init}	T_{fin}
500	**2,0040E-03**	0,0	1,0
900	**1,1123E-03**	0,0	1,0
1500	**6,6711E-04**	0,0	1,0
3000	**3,3344E-04**	0,0	1,0
5000	**2,0004E-04**	0,0	1,0
9000	**1,1112E-04**	0,0	1,0
15000	**6,6671E-05**	0,0	1,0
30000	**3,3334E-05**	0,0	1,0
50000	**2,0000E-05**	0,0	1,0
90000	**1,1111E-05**	0,0	1,0
150000	**6,6667E-06**	0,0	1,0
300000	**3,3333E-06**	0,0	1,0
500000	**2,0000E-06**	0,0	1,0
900000	**1,1111E-06**	0,0	1,0

Az 1-dimenziós térbeli skálák:

2. táblázat
Az 1-dimenziós térhálók

N_x	Δx	X_{init}	X_{fin}
50	**2,0408E-01**	0,0	10,0
100	**1,0101E-01**	0,0	10,0
200	**5,0251E-02**	0,0	10,0
400	**2,5063E-02**	0,0	10,0
800	**1,2516E-02**	0,0	10,0
1200	**8,3403E-03**	0,0	10,0
2000	**5,0025E-03**	0,0	10,0

Ekkor az *Euler*-módszer stabilitása:

3. táblázat

Az *Euler*-módszer stabilitása 1-dimenzióban, 500-tól 15000 időlépésszámig

	N_t	500	900	1500	3000	5000	9000	15000
N_x	$\setminus \Delta t$	2,0040	1,1123	6,6711	3,3344	2,0004	1,1112	6,6671
	$\Delta x \setminus$	E-03	E-03	E-04	E-04	E-04	E-04	E-05
50	2,0408 E-01	4,8116 E-02	2,6707 E-02	1,6017 E-02	8,0060 E-03	4,8030 E-03	2,6681 E-03	1,6008 E-03
100	1,0101 E-01	1,9641 E-01	1,0902 E-01	6,5384 E-02	3,2681 E-02	1,9606 E-02	1,0891 E-02	6,5344 E-03
200	5,0251 E-02	7,9361 E-01	4,4050 E-01	2,6418 E-01	1,3205 E-01	7,9218 E-02	4,4006 E-02	2,6402 E-02
400	2,5063 E-02	3,1904 E+00	1,7709 E+00	1,0620 E+00	5,3085 E-01	3,1847 E-01	1,7691 E-01	1,0614 E-01
800	1,2516 E-02	1,2794 E+01	7,1012 E+00	4,2588 E+00	2,1287 E+00	1,2771 E+00	7,0941 E-01	4,2563 E-01
1200	8,3403 E-03	2,8810 E+01	1,5991 E+01	9,5904 E+00	4,7936 E+00	2,8758 E+00	1,5975 E+00	9,5846 E-01
2000	5,0025 E-03	8,0080 E+01	4,4449 E+01	2,6658 E+01	1,3324 E+01	7,9936 E+00	4,4405 E+00	2,6642 E+00

4. táblázat

Az *Euler*-módszer stabilitása 1-dimenzióban, 30000-től 900000 időlépésszámig

	N_t	30000	50000	90000	150000	300000	500000	900000
N_x	$\setminus \Delta t$	3,3334	2,0000	1,1111	6,6667	3,3333	2,0000	1,1111
	$\Delta x \setminus$	E-05	E-05	E-05	E-06	E-06	E-06	E-06
50	2,0408 E-01	8,0036 E-04	4,8021 E-04	2,6678 E-04	1,6007 E-04	8,0034 E-05	4,8020 E-05	2,6678 E-05
100	1,0101 E-01	3,2671 E-03	1,9602 E-03	1,0890 E-03	6,5340 E-04	3,2670 E-04	1,9602 E-04	1,0890 E-04
200	5,0251 E-02	1,3201 E-02	7,9204 E-03	4,4002 E-03	2,6401 E-03	1,3200 E-03	7,9202 E-04	4,4001 E-04
400	2,5063 E-02	5,3069 E-02	3,1841 E-02	1,7689 E-02	1,0613 E-02	5,3067 E-03	3,1840 E-03	1,7689 E-03
800	1,2516 E-02	2,1281 E-01	1,2768 E-01	7,0934 E-02	4,2560 E-02	2,1280 E-02	1,2768 E-02	7,0934 E-03
1200	8,3403 E-03	4,7922 E-01	2,8753 E-01	1,5974 E-01	9,5841 E-02	4,7920 E-02	2,8752 E-02	1,5973 E-02
2000	5,0025 E-03	1,3320 E+00	7,9922 E-01	4,4401 E-01	2,6640 E-01	1,3320 E-01	7,9920 E-02	4,4400 E-02

5.3. A 2-dimenziós eset

A 2-dimenziós esetekben használt időskálák:

5. táblázat
Időskálák 2-dimenziós esetben

N_t	Δt	T_{init}	T_{fin}
500	2,0040E-04	0,0	0,1
900	1,1123E-04	0,0	0,1
1500	6,6711E-05	0,0	0,1
3000	3,3344E-05	0,0	0,1
5000	2,0004E-05	0,0	0,1
9000	1,1112E-05	0,0	0,1
15000	6,6671E-06	0,0	0,1
30000	3,3334E-06	0,0	0,1
50000	2,0000E-06	0,0	0,1
90000	1,1111E-06	0,0	0,1
150000	6,6667E-07	0,0	0,1
300000	3,3333E-07	0,0	0,1
500000	2,0000E-07	0,0	0,1
900000	1,1111E-07	0,0	0,1

A 2-dimenziós térhálók:

6. táblázat
A 2-dimenziós térhálók

Nx	Ny	N_xN_y	Δx	Δy	$\Delta x\Delta y$
25	25	625	4,1667E-02	4,1667E-02	1,7361E-03
25	50	1250	4,1667E-02	2,0408E-02	8,5034E-04
50	50	2500	2,0408E-02	2,0408E-02	4,1649E-04
50	75	3750	2,0408E-02	1,3514E-02	2,7579E-04
75	75	5625	1,3514E-02	1,3514E-02	1,8262E-04
75	100	7500	1,3514E-02	1,0101E-02	1,3650E-04
100	100	10000	1,0101E-02	1,0101E-02	1,0203E-04

Ekkor az Euler-módszer stabilitása:

7. táblázat

Az *Euler*-módszer stabilitása 2-dimenzióban, 500-tól 15000 időlépésszámig

		N_t	500	900	1500	3000	5000	9000	15000
N_x	N_y	$\setminus \Delta t$ $\Delta x \Delta y \setminus$	2,0040 E-03	1,1123 E-03	6,6711 E-04	3,3344 E-04	2,0004 E-04	1,1112 E-04	6,6671 E-05
25	25	1,7361 E-03	2,3086 E-01	1,2814 E-01	7,6851 E-02	3,8413 E-02	2,3045 E-02	1,2801 E-02	7,6805 E-03
25	50	8,5034 E-04	5,9659 E-01	3,3115 E-01	1,9860 E-01	9,9266 E-02	5,9552 E-02	3,3081 E-02	1,9848 E-02
50	50	4,1649 E-04	9,6232 E-01	5,3415 E-01	3,2035 E-01	1,6012 E-01	9,6059 E-02	5,3361 E-02	3,2015 E-02
50	75	2,7579 E-04	1,5786 E+00	8,7620 E-01	5,2548 E-01	2,6265 E-01	1,5757 E-01	8,7532 E-02	5,2517 E-02
75	75	1,8262 E-04	2,1948 E+00	1,2182 E+00	7,3062 E-01	3,6519 E-01	2,1908 E-01	1,2170 E-01	7,3018 E-02
75	100	1,3650 E-04	3,0615 E+00	1,6993 E+00	1,0191 E+00	5,0940 E-01	3,0560 E-01	1,6976 E-01	1,0185 E-01
100	100	1,0203 E-04	3,9283 E+00	2,1804 E+00	1,3077 E+00	6,5362 E-01	3,9212 E-01	2,1782 E-01	1,3069 E-01

8. táblázat

Az Euler-módszer stabilitása 2-dimenzióban, 30000-től 900000 időlépésszámig

		N_t	300000	500000	900000	1500000	3000000	5000000	9000000
N_x	N_y	$\setminus \Delta t$ $\Delta x \Delta y \setminus$	3,3334 E-06	2,0000 E-06	1,1111 E-06	6,6667 E-07	3,3333 E-07	2,0000 E-07	1,1111 E-07
25	25	1,7361 E-03	3,8401 E-03	2,3040 E-03	1,2800 E-03	7,6801 E-04	3,8400 E-04	2,3040 E-04	1,2800 E-04

25	50	8,5034 E-04	9,9237 E-03	5,9541 E-03	3,3078 E-03	1,9847 E-03	9,9234 E-04	5,9540 E-04	3,3078 E-04
50	50	4,1649 E-04	1,6007 E-02	9,6042 E-03	5,3356 E-03	3,2014 E-03	1,6007 E-03	9,6040 E-04	5,3356 E-04
50	75	2,7579 E-04	2,6258 E-02	1,5754 E-02	8,7523 E-03	5,2514 E-03	2,6257 E-03	1,5754 E-03	8,7522 E-04
75	75	1,8262 E-04	3,6508 E-02	2,1904 E-02	1,2169 E-02	7,3014 E-03	3,6507 E-03	2,1904 E-03	1,2169 E-03
75	100	1,3650 E-04	5,0925 E-02	3,0555 E-02	1,6975 E-02	1,0185 E-02	5,0924 E-03	3,0554 E-03	1,6974 E-03
100	100	1,0203 E-04	6,5342 E-02	3,9205 E-02	2,1780 E-02	1,3068 E-02	6,5340 E-03	3,9204 E-03	2,1780 E-03

6. KEZDŐÉRTÉK-PROBLÉMÁK ÉS REFERENCIAMEGOLDÁSOK

Bár jelen félévben lényegében ugyanazokat a kezdőérték-problémákat és referenciamegoldásokat használom, mint a megelőző félévben, az egyértelműség kedvéért most ezeket ismét ismertetem. Csak és kizárólag *Dirichlet*-féle határfeltétellel dolgozom. Mind 1 dimenzióban, mind pedig 2 dimenzióban 2-2 problémát vizsgálok meg:

- Szinusz függvényen alapuló kezdőérték-probléma, melynek analitikus megoldása ismert
- Olyan probléma, melynek nem ismert analitikus megoldása. Ebben az esetben – a korábbiakhoz hasonlóan – a *MATLAB* `ode15s` függvénye segítségével, valamint az – MSc diplomamunkám [1] során megalkotott – `KerMat_Diffusion_Isolated` függvény által generált ritkamátrix felhasználásával állítok elő numerikus referenciát.

6.1. Az 1-dimenziós probléma analitikus megoldással

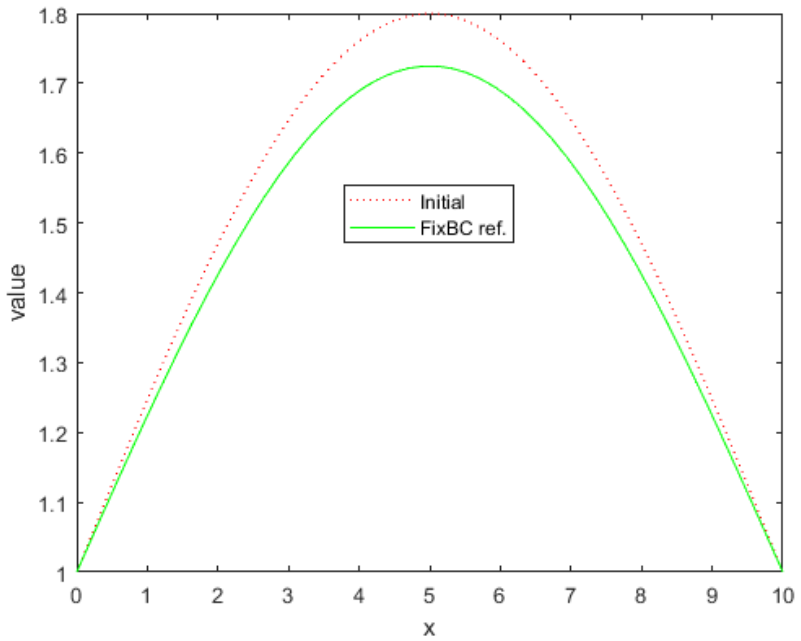
Ebben az esetben egy olyan színuszfüggvényt vizsgálok, melynek nyugvópontjai épp a vizsgált tartomány szélső pontjaira esnek:

$$f(x;t=0) = 0{,}8 \cdot \sin\left(\pi \frac{x}{X_{fin}}\right) + 1. \tag{15}$$

Ennek analitikus megoldása:

$$f(x;t) = 0{,}8 \cdot \exp\left\{-\pi^2 \frac{D \cdot t}{X_{\text{fin}}{}^2}\right\} \cdot \sin\left(\pi \frac{x}{X_{\text{fin}}}\right) + 1. \tag{16}$$

Ábrázolva:



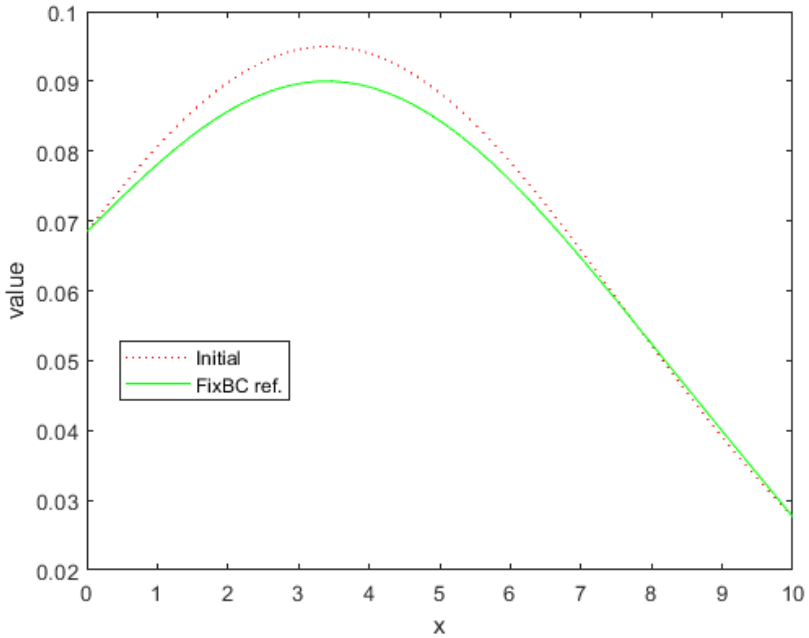
7. ábra
Szinusz kezdőérték-probléma és analitikus megoldása 50 adatponttal

6.2 analitikus megoldással nem bíró 1-dimenziós probléma

Ebben az esetben egy Gauss-függvényt vizsgálok:

$$f(x;t=0) = \frac{1}{\sqrt{2\pi}\cdot 4{,}2}\cdot\exp\left\{-\frac{1}{2}\cdot\left(\frac{x-3{,}4}{4{,}2}\right)^2\right\}. \qquad (17)$$

Ábrázolva:



8. ábra

Gauss kezdőérték-probléma és numerikus referenciamegoldása 50 adatponttal

6.3 A 2-dimenziós probléma analitikus megoldással

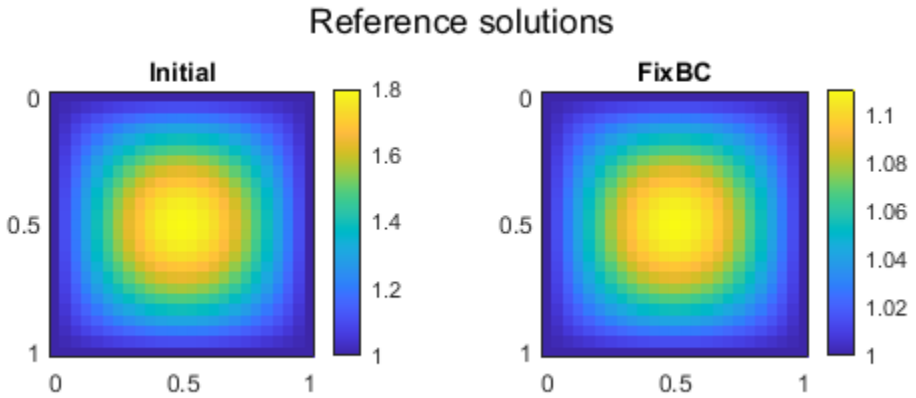
Ebben az esetben egy x-függő és egy y-függő szinusz függvény szorzatát vizsgálom, melyek nyugvópontjai épp a vizsgált tartomány határára esnek:

$$f(x;y;t=0) = 0{,}8 \cdot \sin\left(\pi \frac{x}{X_{\text{fin}}}\right) \cdot \sin\left(\pi \frac{y}{Y_{\text{fin}}}\right) + 1. \tag{18}$$

Ennek analitikus megoldása:

$$f(x;y;t) = 0{,}8 \cdot \exp\left\{-\pi^2 Dt \cdot \left(\frac{1}{{X_{\text{fin}}}^2} + \frac{1}{{Y_{\text{fin}}}^2}\right)\right\} \cdot \sin\left(\pi \frac{x}{X_{\text{fin}}}\right) \cdot \sin\left(\pi \frac{y}{Y_{\text{fin}}}\right) + 1. \tag{19}$$

Ábrázolva:



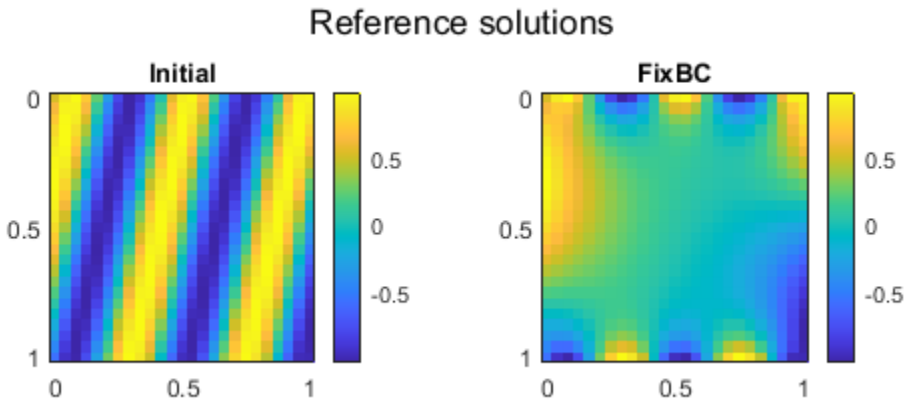
9. ábra
Szinuszok szorzatából álló kezdőérték-probléma és analitikus megoldása 25x25-ös adatrács mellett

6.4 analitikus megoldással nem bíró 2-dimenziós probléma

Ebben az esetben egy ferde irányban terjedő síkhullámot vizsgálok:

$$f(x;y;t=0)=\cos\left(2\pi\cdot[2{,}2 \quad 0{,}5]\cdot\begin{bmatrix}x\\y\end{bmatrix}-1\right). \tag{20}$$

Ábrázolva:



10. ábra
Ferde síkhullám kezdőérték-probléma és numerikus referenciamegoldása 25x25-ös adatrács mellett

6.5 A használt számítógép műszaki paraméterei

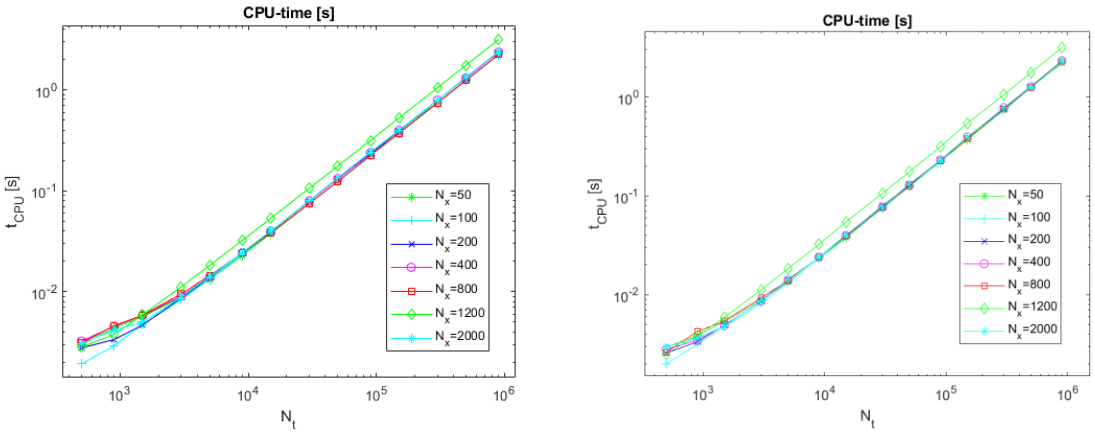
A vizsgálathoz használt számítógép műszaki paraméterei a korábbiakhoz képest változatlanok:

– CPU: 11. gen. *Intel Core i7-11700F*, 2,5 GHz
– RAM: 64 GB
– OS: *Windows10 Professional (22H2)*, x64
– GPU: *NVIDIA*
– *Open CL*: *Intel v2020.3.494*

7. EREDMÉNYEK

7.1. Futásidő összevetése az Explicit Euler-módszerel

7.1.1. Az 1-dimenziós eset

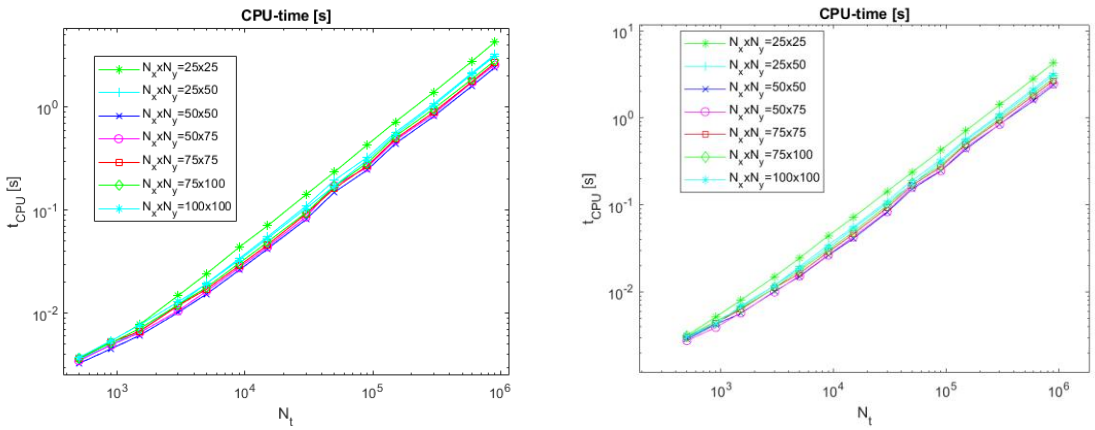


11. ábra

Euler- (balra) és *CNe* (jobbra) módszer futásideje 1 dimenzióban

Látható, hogy *Euler* módszere és a *CNe* módszer között nincs számottevő különbség a futásidő tekintetében. Ez nem is csoda, hiszen ugyanaz a kernelkód futott mindkét esetben, csupán a parametrizálást elvégző – gazda oldali – kódban van némi különbség.

7.1.2. A 2-dimenziós eset

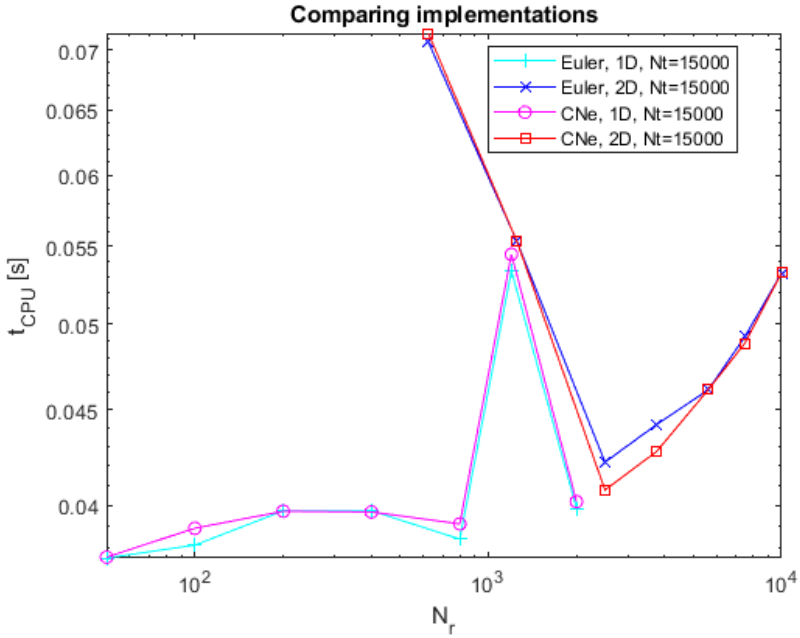


12. ábra

Euler- (balra) és *CNe* (jobbra) módszer futásideje 2 dimenzióban

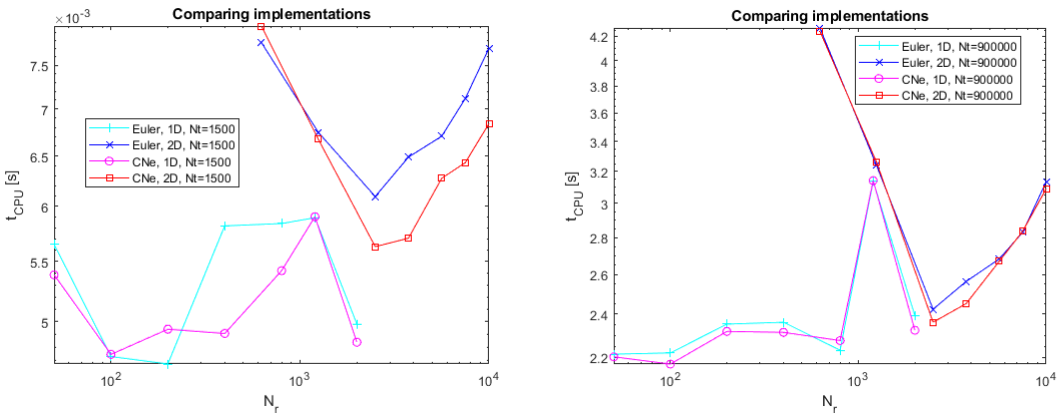
2 dimenzióban is ugyanazt a megállapítást tehetjük, mint 1 dimenzióban: Nincs számottevő különbség a két módszer futásidő igénye között.

7.1.3. Rögzített időlépésszám esete



13. ábra
Euler- és *CNe* módszer összehasonlítása 15000 időlépés esetén

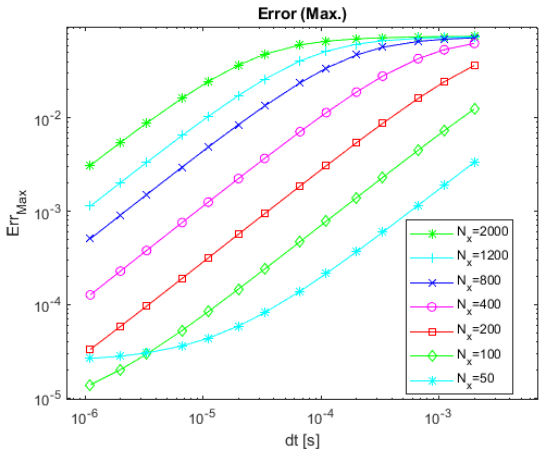
Azonban rögzített időlépésszám mellett azt láthatjuk, hogy a futásidő hektikus módon függ a rendszer méretétől. Hatalmas csúcs látható az $25 \times 50 = 12500$-as rendszerméretnél. Ennek oka az *Open CL* automatikus optimalizáló algoritmusában keresendő, melynek segítségével a feladatokat csoportosítja és kiosztja az egyes GPU-magok között. Érdemes megfigyelni, hogy kicsi időlépésszám esetén a fluktuációk erősebbek, míg nagyobb időlépésszám esetén a görbe alakja már nem változik, csupán a függőleges tengelyen látható futásidő értékek nőnek:



14. ábra
Euler- és *CNe* módszer összehasonlítása 1500 és 900000 időlépés esetén

7.2. Az 1-dimenziós problémák vizsgálata analitikus referenciamegoldással

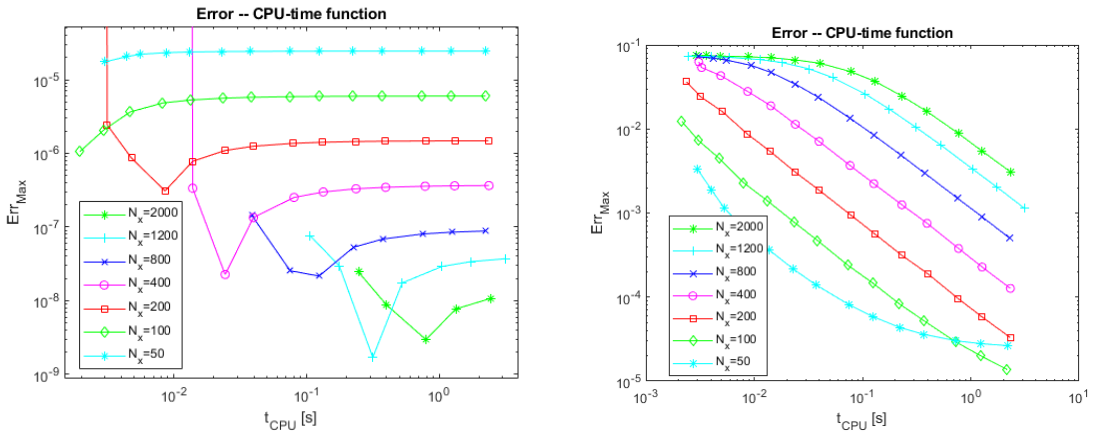
7.2.1. verifikáció hibastatisztika alapján



15. ábra
CNe módszer hibastatisztikája 1D szinusz kezdőérték-probléma és analitikus referenciamegoldás mellett

Látható, hogy a módszer nagy időlépések esetén meglehetősen pontatlan, bár a hiba nagysága az időlépések növelésével aszimptotikusan tart egy felső korláthoz. Ennek oka, hogy a módszer konvex kombináción alapszik, emiatt az értékek sose kerülnek ki a kezdőértékek által megadott tartományból.

7.2.2. Futásidő-hiba karakterisztika összevetése az explicit Euler-módszerrel



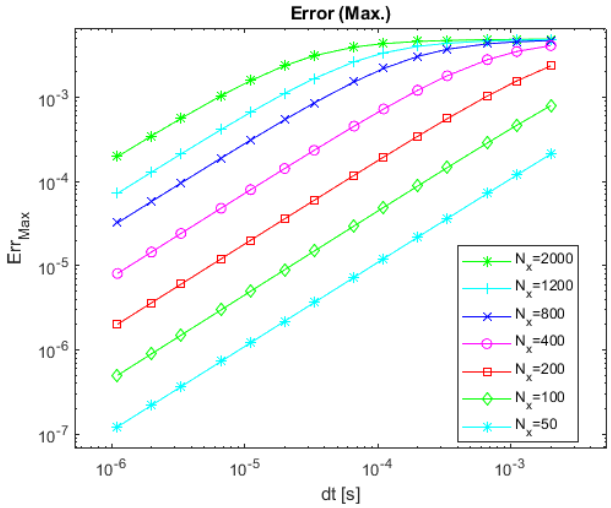
16. ábra
Euler- (balra) és *CNe* (jobbra) módszer futásidő-hiba karakterisztikája 1D szinusz kezdőérték-probléma és analitikus referenciamegoldás mellett

Azt mondhatjuk, hogy a stabilitás szempontjából a *CNe* módszer jobb, hisz feltétel nélkül stabil, viszont pontosságában nem tudja felülmúlni az *Euler*-módszert annak stabilitási tartományán belül. Érdemes ezenkívül megfigyelni, hogy – az 50 adatpontos esetben látható módon – az időlépés csökkentésével a hiba az *Euler*-módszernél is megszokott térbeli diszkretizációból eredő hibához tart, jóllehet *Euler*

módszerével ellentétben *CNe* módszernél nem jelennek meg a jellegzetes pipa alakú görbék. A jelenség további vizsgálatához kisebb időlépésméreteket is számba kéne venni.

7.3. analitikus megoldással nem rendelkező 1-dimenziós problémák vizsgálata

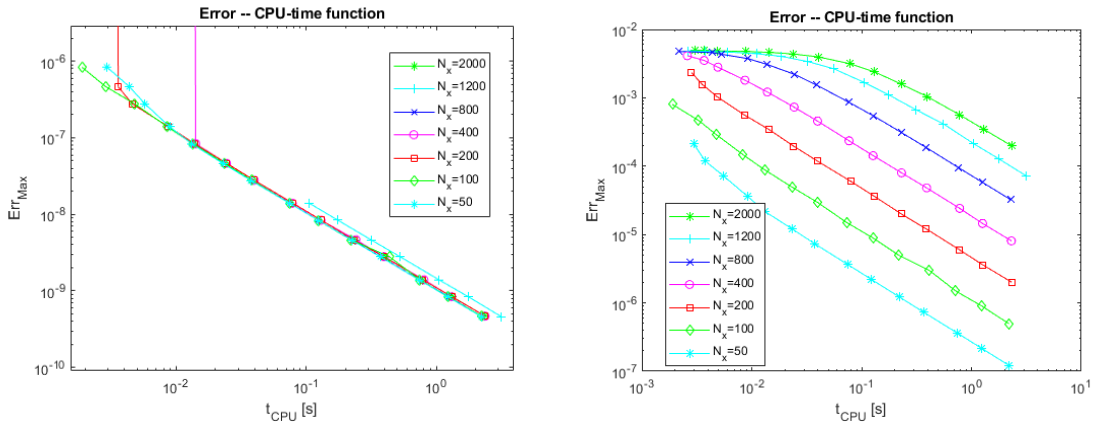
7.3.1. Verifikáció hibastatisztika alapján



17. ábra
CNe módszer hibastatisztikája 1D Gauss kezdőérték-probléma és numerikus referenciamegoldás mellett

Itt is nagy, de fentről korlátos hibával dolgozik az algoritmus nagy időlépések esetén.

7.3.2. Futásidő-hiba karakterisztika összevetése az explicit Euler-módszerrel



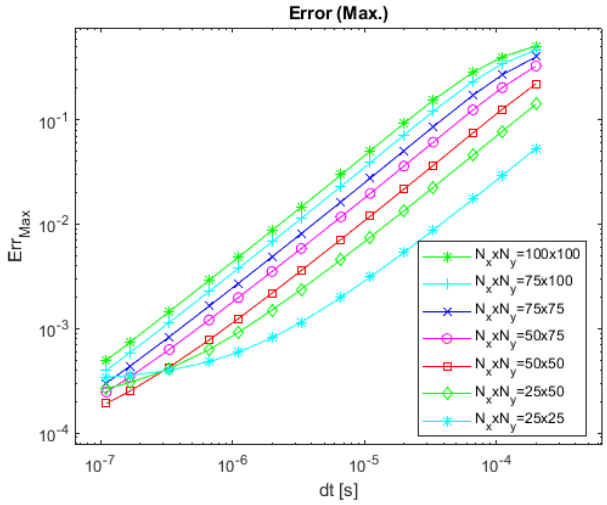
18. ábra

Euler- (balra) és *CNe* (jobbra) módszer futásidő-hiba karakterisztikája 1D *Gauss* kezdőérték-probléma és numerikus referenciamegoldás mellett

Itt is látható, hogy míg stabilitás szempontjából a *CNe*, addig pontosság szempontjából továbbra is az *Euler*-módszer az erősebb. Érdekes megfigyelés továbbá, hogy míg *Euler* módszerénél a hiba nagysága lényegében érzéketlen a térbeli osztásközre, addig a *CNe* módszernél nagyságrendi különbségeket tapasztalunk a különböző rendszerméretek esetében produkált hibák között.

7.4. A 2-dimenziós problémák vizsgálata analitikus referenciamegoldással

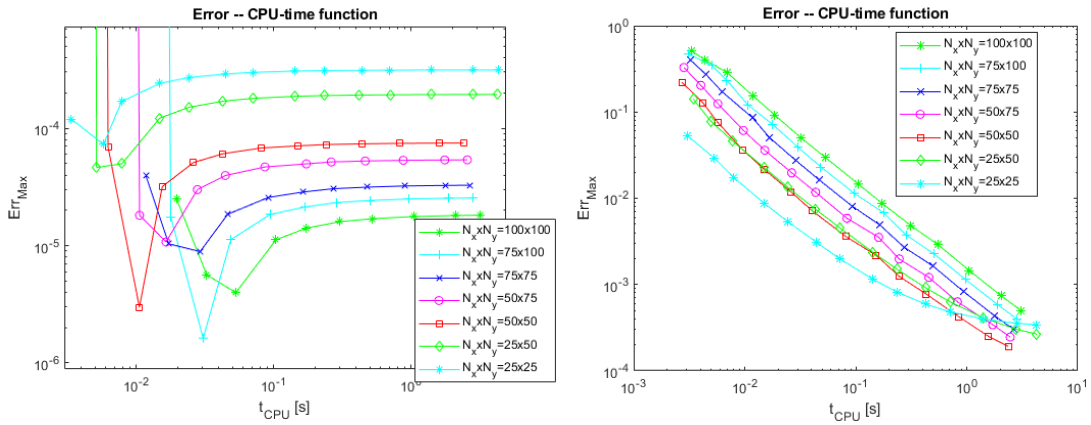
7.4.1. verifikáció hibastatisztika alapján



19. ábra
CNe módszer hibastatisztikája szinuszok szorzatából álló 2D kezdőérték-probléma és analitikus referencia-megoldás mellett

Az 1-dimenziós esethez hasonlóan itt is a pontatlanság írható fel a módszer rovására, továbbá a 25×25-ös esetben jól látszik, hogy kis időlépésekre a hiba kevésbé csökken, valószínűleg az adott térbeli osztásközre jellemző diszkretizációs hibához tart. A felső korlátra utaló jelek csupán a legnagyobb (75×100 és 100×100) adatpontszámok esetében látszanak, valószínűleg ki kéne terjeszteni a vizsgálatot nagyobb időlépésméretre is.

7.4.2. Futásidő-hiba karakterisztika összevetése az explicit Euler-módszerrel

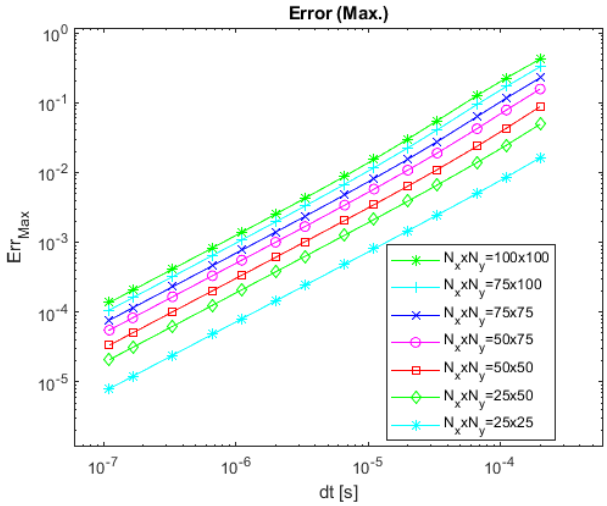


20. ábra
Euler- (balra) és *CNe* (jobbra) módszer futásidő-hiba karakterisztikája szinuszok szorzatából álló 2D kezdőérték-probléma és analitikus referenciamegoldás mellett

Itt is látható, hogy a 25×25-ös eset az *Euler* módszernél is tapasztalható diszkretizációs hibához tart.

7.5. analitikus megoldással nem rendelkező 2-dimenziós problémák vizsgálata

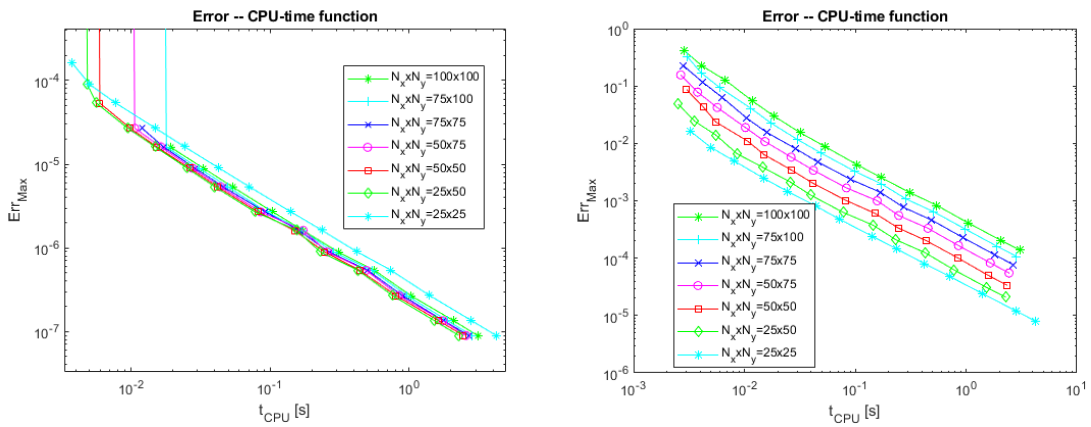
7.5.1. Verifikáció hibastatisztika alapján



21. ábra
CNe módszer hibastatisztikája 2D síkhullám kezdőérték-probléma és numerikus referenciamegoldás mellett

Az 1-dimenziós esethez hasonlóan itt is feltűnnek a viszonylag nagy hibaértékek, főleg nagy időlépések esetén, ráadásul a korábbi ábrákra jellemző felső korlát sem tapasztalható, az összefüggés teljesen lineárisnak tűnik. Remélhetőleg az időlépéseket tovább növelve itt is megváltozna a helyzet, de erre nem került sor jelen mérésben.

7.5.2. Futásidő-hiba karakterisztika összevetése az explicit Euler-módszerrel



22. ábra
Euler- (balra) és *CNe* (jobbra) módszer futásidő-hiba karakterisztikája 2D síkhullám kezdőérték-probléma és numerikus referenciamegoldás mellett

Az eredmény most is hasonlít az 1 dimenzióban tapasztaltakhoz. A *CNe* módszer nemcsak nagyságrenddel nagyobb hibájával tér el az *Euler*-módszertől, hanem abban is, hogy míg *Euler* módszerénél a hiba alig függ a térbeli osztásközök méretétől, addig a *CNe* módszer esetében már jól elkülöníthetők az egyes görbék, noha akkor különbségek nincsenek, mint 1 dimenzióban.

8. KONKLÚZIÓ

Alapvetően elmondható, hogy az explicit *Euler*-módszer – a saját stabilitási tartományán belül – pontosabb, mint a *CNe* módszer, utóbbi viszont feltétlenül stabil, hiszen az eredmény értékei sosem eshetnek a kezdő adatrács konvex burkán kívülre. Emiatt a *CNe* egy hasznos algoritmus alapjául szolgálhat, ha a pontosság nem annyira fontos, viszont feltétlenül olyan körülmények között kell számítást végezni, amikor az *Euler*-módszer már nem használható. A két módszer közötti eltérés továbbá, hogy numerikus referenciamegoldásokkal dolgozva míg az *Euler*-módszernél látszólag nem függ a hiba a térbeli osztásközök méretétől, addig a *CNe* módszernél már tapasztalhatók efféle különbségek. Megjegyzendő persze, hogy a numerikus megoldás önmagában rendelkezik olyan hibataggal, amivel egy analitikus megoldás nem. Analitikus referencia esetén teljesen másképp függ a két módszernél tapasztalható hiba az időlépések méretétől, viszont vannak arra utaló jelek, hogy kicsi időlépésekre mindkét módszer ugyanahhoz a hibataghoz – a térbali diszkretizáció okozta hibához – konvergál (ennek pontosabb megfigyeléséhez célszerű lenne kisebb időlépéseket is bevonni a vizsgálatba).

Az időlépés méretek kiterjesztése mellett hasznos volna még a *CNe* módszer szekvenciális változatán is elvégezni a mérést, és összevetni a párhuzamos változattal, valamint a *CpC* módszer implementációjában lévő hiba kijavítása után azon is elvégezni a vizsgálatot. Mindez a félév hátralévő részében várható.

IRODALOMJEGYZÉK

[1] KOICS D.: **Diffúziós egyenlet végesdifferencia algoritmusának profilozása.** Diplomamunka, Miskolci Egyetem, 2021.

[2] KOICS D.: **Kutatási beszámoló**, Miskolci Egyetem, 2022. január

[3] D. KOICS, K. NEHÉZ, E. KOVÁCS: **Examining the performance of MATLAB's matrix capabilities, testing on Euler's method applied on the diffusion equation**, in Production Systems and Information Engineering. vol. 10 (3) pp. 92-114, 2023. https://doi.org/10.32968/psaie.2022.3.9

[4] KOICS D.: **Kutatási beszámoló**, Miskolci Egyetem, 2023. augusztus

[5] https://bitbucket.org/koicsd/diffusion2/

[6] https://bitbucket.org/koicsd/test-tool/

[7] E. KOVÁCS, J. MAJÁR: **On some analogous transient phenomena**, 32th MicroCAD International Multidisciplinary Scientific Conference, University of Miskolc, 5-6th Sept. 2018. ISBN 978-963-358-162-9

[8] E. KOVÁCS: **A class of new stable, explicit methods to solve the non-stationary heat equation**, Numerical Methods Partial Differential Equations, vol. 37, no. 3, pp. 2469–2489, 2020. https://doi.org/10.1002/num.22730

[9] E. KOVÁCS, Á. NAGY, M. SALEH: **A Set of New, Stable, Explicit, Second Order Schemes for the Non-Stationary Heat Conduction Equation**, Mathematics, vol. 9 (18), no. 2284, 2021. https://doi.org/10.3390/math9182284

[10] Á. NAGY, J. MAJÁR, E. KOVÁCS: **Consistency and Convergence Properties of 20 Recent and Old Numerical Schemes for the Diffusion Equation**, Algorithms, vol. 15 (11), no. 425, 2022. https://doi.org/10.3390/a15110425

[11] MARK H. HOLMES: **Introduction to Numerical Methods in Differential Equations**, chap. 3, Springer, 2007. ISBN 0-387-30891-1

[12] D. NAGY, L. PLAVECZ, F. HEGEDŰS: **The art of solving a large number of non-stiff, low-dimensional ordinary differential equation systems on GPUs and CPUs**, Communications in Numerical Science and Nonlinear Simulation, vol. 112, no. 106521, 2022. https://doi.org/10.1016/j.cnsns.2022.106521

[13] https://en.wikipedia.org/wiki/OpenCL

[14] https://www.khronos.org/opencl/

[15] https://www.intel.com/content/www/us/en/developer/tools/opencl-sdk/choose-download.html

[16] https://www.mathworks.com/help/matlab/ref/ode15s.html

[17] https://git-scm.com/

KÓDJEGYZÉK

Példa a 0 szakaszban található GPU oldali kódot buildelő, és a benne lévő kernelfüggvényt parametrizáló CPU oldali kódra (64 bites lebegőpontos számábrázolást feltételezve):

```
std::vector<const char*> c_lines = readFile("C:/path/to/kernel.cl");
cl_int err;
cl_program program;
program = clCreateProgramWithSource(context,
                    c_lines.size(), c_lines.data(), NULL, &err);
if (err != CL_SUCCESS) {
        printErr("creating Program object", err);
        return;
}
err = clBuildProgram(program, 1, &device, NULL, NULL, NULL);
if (err != CL_SUCCESS) {
        printErr("building kernel-program", err);
        return;
}
double D_times_dt = D * dt;
cl_double2 coeffs;
coeffs.x = D_times_dt / pow(scaling.dx(), 2);
coeffs.y = D_times_dt / pow(scaling.dy(), 2);
cl_uint2 cl_size {scaling.Nx, scaling.Ny};
cl_long4 neighbours {
        -1, 1,  // x neighbours diff
        -(cl_long)scaling.Nx, (cl_long)scaling.Nx  // y neighbours diff
};
odd_kernel = clCreateKernel(program, name, &err);
if (err != CL_SUCCESS) {
        printErr("selecting Kernel for name", err);
        return;
}
// before:
err = clSetKernelArg(odd_kernel, 0, sizeof(buf1), &buf1);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// after:
err = clSetKernelArg(odd_kernel, 1, sizeof(buf2), &buf2);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
```

```
        return;
}
// coeffs:
err = clSetKernelArg(odd_kernel, 2, sizeof(coeffs), &coeffs);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// size:
err = clSetKernelArg(odd_kernel, 3, sizeof(cl_size), &cl_size);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// neighbours:
err = clSetKernelArg(odd_kernel, 4, sizeof(neighbours), &neighbours);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
even_kernel = clCreateKernel(program, name, &err);
if (err != CL_SUCCESS) {
        printErr("selecting Kernel for name", err);
        return;
}
// before:
err = clSetKernelArg(even_kernel, 0, sizeof(buf2), &buf2);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// after:
err = clSetKernelArg(even_kernel, 1, sizeof(buf1), &buf1);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// coeffs:
err = clSetKernelArg(even_kernel, 2, sizeof(coeffs), &coeffs);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// size:
```

```
err = clSetKernelArg(even_kernel, 3, sizeof(cl_size), &cl_size);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
// neighbours:
err = clSetKernelArg(even_kernel, 4, sizeof(neighbours), &neighbours);
if (err != CL_SUCCESS) {
        printErr("setting Kernel-argument", err);
        return;
}
```

ahol cl_context context és cl_device_id device az *Open CL* inicializációja során létrehozott struktúrapointerek, std::vector<const char*> readFile(const char*) egy megfelelő, fájl beolvasására alkalmas függvény, cl_mem buf1, buf2 a video-RAM lefoglalásakor kapott struktúrapointerek, továbbá void printErr(const char*, cl_int) a statikus könyvtáramban [5] található, hibakódot felismerő és a hiba rövid nevét kiíró függvény, Scaling scaling pedig a diplomamunkám [1] során megalkotott, skálázási paramétereket tároló struktúra.

Megjegyzendő, hogy a fent ismertetett programkód nem idézet az [5] repozitóriumból, csupán annak megértését szolgáló, egyszerűsített változata, ennélfogva ebben a formájában nem is került tesztelésre.