



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem

# GPU kompatibilis áramlástan megoldó fejlesztése Taichi keretrendszerben

OTDK DOLGOZAT

*Készítette*

Damenija Vitéz

*Konzulens*

Sándor Levente, PhD Hallgató

2025. január 15.

# Tartalomjegyzék

|                                                 |           |
|-------------------------------------------------|-----------|
| <b>Kivonat</b>                                  | <b>3</b>  |
| <b>Abstract</b>                                 | <b>4</b>  |
| <b>Köszönetnyilvánítás</b>                      | <b>5</b>  |
| <b>1. Bevezetés</b>                             | <b>6</b>  |
| 1.1. Aneurizmák és kezelésük . . . . .          | 6         |
| 1.1.1. Sztentek . . . . .                       | 6         |
| 1.2. Hemodinamika és a CFD . . . . .            | 8         |
| 1.2.1. Sebészeti döntéshozatal . . . . .        | 8         |
| 1.2.2. Kihívások . . . . .                      | 9         |
| 1.2.3. GPU klinikai használata . . . . .        | 9         |
| 1.3. Motiváció . . . . .                        | 10        |
| <b>2. Lattice Boltzmann Módszer</b>             | <b>11</b> |
| 2.1. Boltzmann egyenlet . . . . .               | 12        |
| 2.2. Ütközési operátor . . . . .                | 13        |
| 2.2.1. BGK ütközési operátor . . . . .          | 14        |
| 2.3. LBM és az áramlástan . . . . .             | 14        |
| 2.4. LBM diszkretizálás . . . . .               | 15        |
| 2.4.1. Sebesség diszkretizálása . . . . .       | 16        |
| 2.4.2. Tér és idő diszkretizálása . . . . .     | 19        |
| 2.4.3. Algoritmus megvalósítás . . . . .        | 19        |
| 2.4.4. Stabilitási megfontolások . . . . .      | 20        |
| 2.5. Peremfeltételek . . . . .                  | 21        |
| <b>3. Taichi Keretrendszer</b>                  | <b>24</b> |
| 3.1. Használat . . . . .                        | 25        |
| 3.2. SNode rendszer . . . . .                   | 26        |
| <b>4. Implementáció</b>                         | <b>28</b> |
| 4.1. Használat . . . . .                        | 28        |
| 4.1.1. Lid-driven Cavity 2D példakód . . . . .  | 29        |
| 4.1.2. Poiseuille Gravity 3D példakód . . . . . | 31        |

|           |                                                   |           |
|-----------|---------------------------------------------------|-----------|
| 4.2.      | A szoftver felépítése . . . . .                   | 32        |
| 4.2.1.    | Adatosztályok . . . . .                           | 34        |
| 4.2.2.    | Logika . . . . .                                  | 35        |
| 4.3.      | Algoritmikus helyesség ellenőrzése . . . . .      | 39        |
| <b>5.</b> | <b>Eredmények</b>                                 | <b>40</b> |
| 5.1.      | Áramlástanai esetek validálása . . . . .          | 40        |
| 5.1.1.    | Lid-driven cavity . . . . .                       | 40        |
| 5.1.2.    | Poiseuille Gravity 2D . . . . .                   | 42        |
| 5.1.3.    | Porózus zónájú csőbeli áramlás . . . . .          | 43        |
| 5.2.      | Teljesítmény összehasonlítás OpenLB-vel . . . . . | 45        |
| <b>6.</b> | <b>Összefoglaló</b>                               | <b>47</b> |
|           | <b>Bibliography</b>                               | <b>51</b> |

# Kivonat

Az aneurizma egy igen gyakori kardiovaszkuláris megbetegedés, amely az érfal elváltozásával létrejövő zsákfali kiboltosulás. Kiszakadása esetén belső vérzést okoz, ami agyi artéria esetén sztrókhhoz, kómához vezethet. Kezelési módszerei, mint például a stent esetén a paraméterek kiválasztása jelenleg orvosonként eltérő gyakorlaton, intuíción alapszik, és ettől függően különböző klinikai kimenetelhez vezethet. A világszerte előregedő társadalmakban egyre nő az előfordulása, és ezzel az ideális kezelési paraméterek kérdésének körüljárásának fontossága.

Az optimális kezelést szolgáló paraméter választását segítheti szimulációs kísérletsorozatokon alapuló irányelv létrehozása, vagy egy páciens-specifikus módszer kidolgozása. Az utóbbi esetben a minél gyorsabban történő paraméter választást segíthetik a Lattice Boltzmann Method (LBM) alapú véráramlástani szimulációk, amelyek számítása jelentősen párhuzamosítható. A jelenleg széles körben elérhető LBM keretrendszerek átfogóan használják a C++ programozási nyelv Template eszközt a jó teljesítményt biztosító szimulációs környezet felállítására, viszont ez gyakran nehézkes, hosszas nyelvezettel és esetleges kompilációs hibákkal jár, ami a kód érthetőségét, változtathatóságát, és fejlesztését nehezíti akár egy mérnök, vagy egy tapasztalt programozó számára. Az elmúlt 3 évben elterjedt, és számos kutatócsoport által használt Taichi Keretrendszer lehetővé teszi, hogy Python nyelven lehessen írni nagy teljesítményű, akár videokártyán vagy HPC környezetben futó programkód megírását úgy, hogy közel azonos lehetőségeket nyújt, anélkül, hogy a programkódot írónak mély tapasztalata legyen a GPGPU programozásról.

A projekt célja a Taichi keretrendszeren épülő, egyszerre gyors teljesítményt nyújtó, könnyen megérthető és bővíthető keretrendszer létrehozása az aneurizmák kezelését szolgáló módszerek hemodinamikai elemzésére.

# Abstract

An aneurysm is a frequently occurring cardiovascular disease presenting as an outward bulging of a blood vessel wall caused by its pathological alteration, which carries the risk of rupturing, thus causing hemorrhage, which in the case of an intracranial aneurysm can lead to stroke or a coma. The selection of parameters for its treatment methods such as stents differs between physicians and is based on their practice and intuition, which can lead to varying clinical outcomes. With the worldwide trend of population ageing, its occurrence has been increasing, and so has the need for exploring the problem of choosing ideal treatment parameters.

To serve this purpose, computational fluid dynamic (CFD) simulations can help create guidelines or patient-specific methods that help improve long-term treatment results. In the latter case, adequately fast parameter selection could be achieved with the help of Lattice Boltzmann Method (LBM) based hemodynamic simulations which are highly parallelizable in their computations. The currently available widely used LBM frameworks extensively use the Template feature of the C++ programming language in order to provide a high performance simulation environment, however, this often hurts code readability and development with its verbose syntax and lengthy compilation errors, often to the detriment of engineers and even experienced programmers when it comes to writing simulation code. The Taichi framework which has become popular in the last 3 years and has been used by various research groups allows writing high performance code in Python that can be run on either GPUs or HPC environments while still providing the same performance benefits without the need for a substantial amount of GPGPU programming experience.

The goal of the project is to develop a high performance, easily usable and extensible fluid dynamics solver utilizing the Taichi framework in order to analyze the hemodynamics of aneurysm treatment methods.

# Köszönetnyilvánítás

Ezt a tanulmányt az NVIDIA Corporation támogatta az Academic Hardware Grants Programon keresztül.

# 1. fejezet

## Bevezetés

### 1.1. Aneurizmák és kezelésük

Az aneurizma egy ér (elsősorban artéria) falának helyi gyengülése következtében kialakuló kifelé domborodó tágulat, ami végül repedéshez vezethet, így vérzést okozva. Az orvosi szakirodalomban általában akkor beszélünk aneurizmáról, ha az ér átmérője legalább 50%-kal nagyobb, mint az adott szakasz normál szegmensének átmérője.

Az aneurizmák leggyakrabban az aorta különböző részein (hasi, mellkasi) és az agyi artériákban (pl. Willis-kör) fordulnak elő, és kezelés nélkül életveszélyesek lehetnek. Az aorta aneurizma megrepedése sürgős sebészeti beavatkozást igényel, mivel a jelentős vérvesztés következtében hypovolaemiás sokkhoz vezethet, míg az agyi aneurizmák megrepedése subarachnoidális vérzést okozhat, amely maradandó agykárosodást eredményezhet, és 30 napon belüli 45%-os halálozási kockázattal jár [3].

Bár az aneurizmák kialakulásának pontos mechanizmusa még nem tisztázott, kockázati tényezők közé tartozik az előrehaladott életkor, dohányzás, alkoholizmus, elhízás, cukorbetegség, magas vérnyomás, kiegyensúlyozatlan étrend, családi halmozódás (pl. genetika), artériákat érintő fertőzések (mycotikus aneurizmák), valamint az is elmondható, hogy agyi aneurizmák gyakrabban fordulnak elő nőknél [22, 35, 4, 37, 15].

Az agyi aneurizma elsősorban az idősebbeknél fordul elő [40], és jelenleg a stroke egyik vezető oka [48]. Ennek megfelelően, és a világszerte öregedő társadalmat figyelembe véve ez növeli az optimális kezelési módszerek és paraméterek kiválasztásának szükségességét (pl. a sztent és annak átmérője) [20].

A kezelés célja a véráramlás csökkentése az aneurizmában, mivel megfigyelték, hogy a helyi nyomás (áramlás) növekedése tovább tágíthatja az aneurizma falát, míg a véráramlás kritikus csökkentése az aneurizma zsugorodását, végül eltűnését idézheti elő [33].

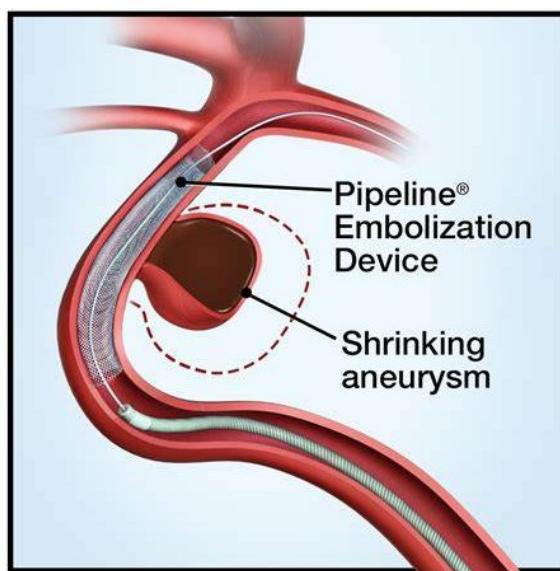
#### 1.1.1. Sztentek

Az aneurizmák kezelésének egy régóta használt módja, hogy fém spirálvezeték, vagyis endovaszkuláris coilt vezetünk be az aneurizma testjébe, így vérrögképződést és azt követő áramlástól való elzáródást, aneurizma zsugorodást okozva.

A sztenteket gyakran használják szűkület kezelésére, az ér falának kitágítására, hogy

megfelelő véráramlást biztosítsanak a helyreállított lumenben. Ilyen célú sztentek esetén katétert használnak, amelynek külső rétege maga a sztent, amelyet egy felfújható ballon segítségével helyeznek el az ér falában [46].

A sztenteket aneurizmáknál kezdetben azért alkalmazták, hogy az endosvaszkuláris coilt stabilizálják, vagyis hogy az ne tudjon az aneurizmán kívül terjeszkedni, így fölsértve az eret és akadályozni a normális áramlást. Később kiderült, hogy megfelelő hálósűrűség esetén a sztent használható önállóan áramlásterelő eszközként, mely az aneurizmán kívülre irányítja az áramlást, elkerülve ezzel a túlzott nyomást az aneurizma falán (ami további tágulást okozhat, és növeli a repedés kockázatát), valamint hosszabb távon az aneurizma zsugorodásához vezet [44, 12] (lásd: 1.1. ábra). Ezeket flow diverter sztenteknek hívják, és elhelyezésük esetén különbség, hogy maguktól kitágulnak amiután eltávolítottuk a körülrő hüvelyet, így nincs szükség ballonra [32].



**1.1. ábra.** Áramlásterelő sztent az aneurizmák kezelésére. A csökkent véráramlás és a falra ható nyírófeszültség az aneurizma hosszú távú zsugorodását eredményezi [1]

A sztentek trombogén hatásúak lehetnek, ami vérrögök felhalmozódását okozhatja a hálójukon, ez pedig részleges vagy akár teljes véráramlás-elzáródást is eredményezhet az ér lumenében, ischémiát okozva az ellátott szövetekben (pl. ischémiás stroke az agyban). Ez a komplikáció általában aszpirin és más vérhígító gyógyszerek szedésével kerülhető el, melyeket a sztent beültetése után akár egy évig vagy még hosszabb ideig is szedni kell [29, 36].

A sztentek másik kockázata az in-sztent restenosis (ISR), amely a lumen újbóli beszűkülése neointimális hiperplázia (hegképződés) miatt. Ez simaizomsejtek és makrofágok túlzott mértékű felszaporodásából áll [17]. Ezt bizonyos anyagokból készült sztentek alkalmazásával lehet csökkenteni, vagy bevont sztentekkel, melyek a szöveti falhoz hasonló bevonattal rendelkeznek [48].

## Kezelés kiválasztása

Egy kisebb átmérőjű, sűrűbb hálójú sztent segíthet megfelelő közegellenállást biztosítani az aneurizma gyorsabb zsugorodásához, viszont egy túl kicsi sztent mechanikailag instabil lehet, ami sztent elmozdulást (sztent embóliát) okozhat, és ez életveszélyes szövődmény [6]. Másrészt, a méret növelése elősegítheti a stabilitást, viszont hátrányosan befolyásolhatja az áramlási ellenállást, ami a kezelés sikerességét csökkentheti. Ez nehézséget jelent az optimális egyensúly megtalálásában egy adott ér esetén.

A hálósűrűség kiválasztása szintén nehéz feladat, mivel annak növelése az aneurizma felszíni nyomásokat növelheti vagy csökkentheti az aneurizma és a környező erek geometriájától függően [25].

Más paraméterek, mint például a sztent hossza, elhelyezési helye, vastagsága és hálómintája hasonló kihívásokkal járnak, és a műtét előtt az optimális paraméterek kiválasztása javíthatja a klinikai eredményeket.

## 1.2. Hemodinamika és a CFD

A Computational Fluid Dynamics (CFD) olyan makroszkópikus áramlástani változókat vizsgál, mint például a folyadék sebességet, a nyomást és a nyírófeszültséget egy adott áramlási problémában többnyire a Navier–Stokes egyenletek alkalmazásával, melyek leírják, hogy hogyan viselkednek a viszkózus folyadékok.

A hemodinamikai szimulációk segíthetnek megérteni, hogy az egyes beavatkozások milyen mechanizmusok révén működnek, és lehetőséget nyújtanak az adott kezelés (pl. sztent) optimális paramétereinek megtalálására, melyek javíthatják a klinikai eredményeket.

Ennek megfelelően potenciálisan alkalmazhatók klinikai irányelvek kidolgozására, melyek statisztikailag reprezentatív mintákon alapulnak, vagy elegendően gyorsan lefutó szimulációk esetén betegspecifikus kezelésekhöz is használhatók.

A véráramlás vizsgálatakor, az időbeli megvalósíthatóság érdekében gyakran egyszerűsítésként feltételezzük, hogy a vér newtoni folyadék, azaz a deformáció sebessége lineárisan arányos a  $\tau$  nyírófeszültséggel. A valóságban ez csak a vérplazmára igaz, mivel a vér különböző összetevői (pl. vörsejtek) nem-newtoni tulajdonságokat kölcsönöznek neki.

### 1.2.1. Sebészeti döntéshozatal

Mivel az aneurizmák ritkák és nincs rájuk rutinszerű szűrés, ezért többnyire véletlenül fedezik fel őket más vizsgálatok során. Az aneurizmák helyének és geometriájának azonosítása angiográfia során történik, amikor kontrasztanyagokat fecskendeznek a véráramba, hogy az ereket elkülönítsék a környező struktúráktól.

A számítógépes tomográfia (CT) röntgensugarakkal dolgozik, és a kontrasztanyagok, mint az jódtól megnövelt fotoelektromos elnyelést biztosítanak, így kiemelve a vérrel telt lument.

A mágneses rezonancia képalkotás (MR) az atom spin polarizációjának változását előidézve annak relaxációját (visszatérését a nyugalmi helyzetbe) méri, és leggyakrabban

gadolinium(III)-t használ, amely kimutatható relaxációs idő rövidülést idéz elő, így szintén kiemelve a vérrel telt lument [5, 45].

A CT-vizsgálatok gyakran alkalmazzák a digitális szubtrakciós angiográfia (DSA) technikát. Ez azt jelenti, hogy az ér kontrasztanyag befecskendezése előtti képet kivonják abból a képből, amelyet a kontrasztanyag áramlása során készítettek, így kiemelve az aneurizmát és a környező lumen szerkezetét (melyeken belül folyik a vér).

Aneurizma megrepedése esetén elengedhetetlen, hogy a kezelést segítő eszközök néhány órán belül eredményt hozzanak, ezért egy hemodinamikai szimulációnak a kritikus időkereten belül kéne lefutnia [16].

### 1.2.2. Kihívások

A 3D CFD szimulációk során a térbeli diszkrét felbontás, amely befolyásolja az eredmények pontosságát,  $O(N^3)$  arányban növekszik, ahol  $N$  az adott irány minimális hálórács felbontása<sup>1</sup>. Ez különösen megterhelő lehet számítási szempontból, ha a geometria különböző hosszskálái erősen eltérnek (például egy viszonylag kis aneurizma egy nagyobb érben).

Egy klinikailag hasznos hemodinamikai szimuláció az aneurizmák vészhelyzeti kezelése során olyan számítások elvégzését igényelné, melyek lehetővé teszik a különböző kezelési paraméterek (pl. sztent paraméterek) pontos összehasonlítását, hogy a lehető legjobb megoldást választhassuk, miközben viszonylag elegendő idő alatt futnak le a szimulációk (2-3 órán belül). Ez elengedhetetlen ahhoz, hogy kritikus időkereten belül el lehessen végezni a beavatkozást<sup>2</sup> [30, 38].

### 1.2.3. GPU klinikai használata

A videokártyák (GPU) képességeit általános számításra használó General Purpose GPU (GPGPU) módszerek lehetővé teszik a nagyszabású párhuzamos számításokat még egyetlen számítógépen is. Hasznossága a számítások gyors futtatásában egyre nyilvánvalóbbá vált a mesterséges intelligencia (MI) térnyerésével, amely szintén nagymértékben támaszkodik a párhuzamos számítási kapacitásra [24]. Ezért a párhuzamosításból származó teljesítményt hasznosítani lehet gyors válaszidőre tervezett megoldások fejlesztésében, amikor a képképző adatok megszerzésétől a szimulációk futtatásán és a kezelési paraméterek kiválasztásán keresztül a legjobb klinikai eredmény elérése a cél.

Azonban mivel a videokártyákra történő programozás speciális ismereteket igényel a párhuzamos programozás terén, mint például a globális és megosztott memória hatékony használata, valamint cache barát kódolási gyakorlatok, csak nemrégiben kezdték el támogatni egyes CFD keretrendszerek. A széles körben használt keretrendszerek, mint például az OpenFOAM, még mindig nem rendelkeznek hivatalos GPU-támogatással, mivel ennek megvalósítása és integrálása a meglévő kódbázisba egy összetett [34] feladat.

A meglévő keretrendszerek egyik nagy problémája, hogy az objektumorientált programozás (OOP) mint absztrakció nem támogatott, és nem használható megfelelően a GPU-k

<sup>1</sup>Adaptív háló finomítást nem feltételezve

<sup>2</sup>A kritikus időkeret az az időtartam, amely után a szövődmények vagy a halálozás valószínűsége jelentősen növekszik.

hatékony kihasználására. Emiatt az adatok kezelését újra kell tervezni, ami nem egyszerű feladat egy olyan keretrendszerrel, mint az OpenFOAM, amely több mint egymillió sorból álló kódbázissal rendelkezik [23].

Egy lehetséges megoldás a *data-oriented programming*, amely az adatok elrendezésének absztrakciójának elkerülésére összpontosít, hogy a lehető legrövidebb futásidőt biztosítsa a célhardveren (pl. GPU). Ez a logika és az adatok elkülönítését ösztönzi annak érdekében, hogy a teljesítmény-kritikus adatfolyam átláthatóbb, érthetőbb legyen [8]. Bár ez nem zárja ki osztályok és metódusok használatát, óvatosságra int az absztrakciók használatánál, mivel a cél a futási idő javítása cache-barát módon.

Magas szintű keretrendszerek, mint például a C++-hoz készült SYCL, a Pythonhoz írt Numba, vagy a Taichi lehetőséget biztosítanak arra, hogy rövid időn belül írjunk működőképes kódot, amely akár processzoron, vagy processzoron futhat, ezáltal különböző számítógépeken és nagy teljesítményű rendszereken is alkalmazható. Ez különösen hasznos lehet például nagy teljesítményű grafikus kártyákkal rendelkező laptopokon vagy HPC rendszerek használatakor.

### 1.3. Motiváció

A videokártyák folyamatosan (Moore-törvény szerint) növekvő számítási teljesítménye és a Taichi nevű, magas szintű párhuzamos számítást támogató új keretrendszer megjelenése lehetővé teszi egy olyan CFD keretrendszer létrehozását, amely gyors szimulációkat kínálhat klinikailag releváns pontossággal, miközben jól ismert és széles körben használt Python szintaxist alkalmaz, amely megkönnyíti az új módszerek folyamatos fejlesztését és bevezetését anélkül, hogy mély GPGPU programozási ismeretekre lenne szükség. Ez felhasználható betegspecifikus sztent paraméterek meghatározásában, vagy a paraméterekre vonatkozó irányelvek kidolgozásában.

A jelenleg újra teret nyerő CFD megoldó, a Lattice Boltzmann Módszer (LBM) különösen alkalmas a szimulációk párhuzamosítására, így különféle fizikai problémánál, mint például többfázisú áramlások, porózus anyagok és biokémiai reakciókkal kapcsolt áramlások esetén hatékonyan alkalmazható [18, 28].

Ez lehetővé teszi egy CFD keretrendszer fejlesztésének célkitűzését a Lattice Boltzmann Módszer használatával a Taichi keretrendszeren, amely lehetővé teszi a sztentkonfigurációk számítógépes szimuláción (in silico) alapuló értékelését betegspecifikus aneurizma geometriákon, melyeket orvosi képalkotással lehet kinyerni. Ez segít az intervenciós radiológusoknak meghatározni a legmegfelelőbb sztentet olyan betegek számára, akik akár sürgősségi vagy tervezett műtétre szorulnak.

A dolgozat az LBM-en (Lattice Boltzmann Módszeren) alapuló keretrendszer megvalósítását célozza a Python Taichi keretrendszer felhasználásával aneurizmák elemzésére, és ellenőrzi annak áramlástani helyességét, valamint összehasonlítja a teljesítményét más LBM alapú keretrendszerekkel.

## 2. fejezet

# Lattice Boltzmann Módszer

A Lattice Boltzmann Módszer a folyadék áramlást mezoszkópikus skálán közelíti meg, vagyis nem szimulálja a diszkrét folyadék részecskéket, és nem fogalmaz meg differenciálegyenletet a folyadék kontinuumra, mint makroszkópikus jelenségre, hanem a részecskék eloszlásának statisztikáját írja le, melyből majd adott feltételezésekkel levezethetőek a Navier-Stokes egyenletek.

Akkor alkalmazható, amikor már elhanyagolhatóak a mikroszkópikus részletek, vagyis egy adott molekula két egymást követő ütközése közt megtett átlagos úthossz  $l_{mfp}$  (mean free path) jóval kisebb, mint a rendszerünk vizsgálati skálájának hossza  $l$ , valamint a rendszer térbeli diszkretizálásakor kapott cellák hossza  $l_{av}$  jóval nagyobb az előzőnél, viszont jóval kisebb az utóbbinál [21]:

$$l_{mfp} \ll l_{av} \ll l \quad (2.1)$$

Jelöljük  $\vec{\xi}$ -vel adott részecske sebességét,  $\vec{x}$ -el egy térbeli pontot, valamint  $t$ -vel az időt. Ekkor igaz, hogy:

$$\vec{\xi} = \frac{d\vec{x}}{dt} \text{ (Molekuláris sebesség)} \quad (2.2)$$

Ekkor az adott időben, adott térben lévő és adott sebességű részecskékre vonatkozó  $f(\vec{x}, \vec{\xi}, t) \left[ \frac{1}{m^3 \cdot (m/s)^3} \right]$  eloszlásfüggvényre vonatkozóan többek közt az alábbi elvárásokat / tulajdonságokat fogalmazzuk meg:

$$\int_{\vec{\xi}} \int_{\vec{x}} f(\vec{x}, \vec{\xi}, t) d^3x d^3\xi = M(t) \text{ (Összes részecske száma adott időben)} \quad (2.3)$$

$$\int_{\vec{\xi}} f(\vec{x}, \vec{\xi}, t) d^3\xi = \rho(x, t) \text{ (Részecske sűrűség adott időben, helyen)} \quad (2.4)$$

$$\int_{\vec{\xi}} \vec{\xi} f(\vec{x}, \vec{\xi}, t) d^3\xi = \rho \vec{u}(\vec{x}, t) \text{ (Lendületsűrűség adott időben)} \quad (2.5)$$

Ennek megfelelően  $f(\vec{x}, \vec{\xi}, t)$  magába foglalja a helyi folyadék tulajdonságokat, valamint

megjelennek a makroszkópikus tulajdonságok, mint annak momentumjai<sup>1</sup>.

A kiindulási egyenlet a **Boltzmann egyenlet**, mely az  $f$  függvény időbeli változását írja le annak teljes deriváltjával [21] (A levezetésért lásd: 2.1):

$$\frac{Df(\vec{x}, \vec{\xi}, t)}{Dt} = \left( \frac{\partial}{\partial t} + \vec{\xi} \cdot \frac{\partial}{\partial \vec{x}} + \frac{d\vec{\xi}}{dt} \cdot \frac{\partial}{\partial \vec{\xi}} \right) f(\vec{x}, \vec{\xi}, t) = \Omega(f) \quad (2.6)$$

Ahol  $\Omega(f)$  az ütközési operátor, ami a részecskék ütközését és ebből eredendően az eloszlás időbeli megváltozását foglalja magába, nem megsértve a tömeg-és impulzusmegmaradás törvényét.

A Boltzmann Egyenletből kiinduló Lattice Boltzmann (LB) módszerből Chapman–Enskog elemzése alapján levezethető a Navier-Stokes egyenlet ami alapján már a vizsgálandó változók, mint a sebesség, nyomás, nyírófeszültség már kiszámolhatóak [31].

Numerikus szimulációknál mind a sebességirány, idő, és tér diszkrétizálva van, a leggyakrabban használt BGK approximáció esetén az ütközési függvény pedig:  $\Omega(f) = \frac{1}{\tau}(f^{eq} - f)$ , ahol  $\tau$  paraméter meghatározza a folyadék viszkozitását, az  $f^{eq}(\vec{\xi}, \vec{u})$  egyensúlyi eloszlás pedig függ  $\vec{\xi}$  mikroszkópikus sebesség és  $\vec{u}$  makroszkópikus sebességtől, számításakor pedig gyakran a Maxwell-Boltzmann eloszlásból levezetett függvény másodrendű Taylor-sor közelítése használt. Az algoritmus lokális cellabeli ütközési (collision), majd pedig a cellák közti terjedési (streaming) szakaszokra bontható egy adott időlépésben.

Az algoritmus erős lokalitása miatt kifejezetten alkalmas többszálúsításra, így például szuperszámítógép processzorain vagy videokártyán való futtatáshoz.

## 2.1. Boltzmann egyenlet

A Boltzmann egyenlet leírja a infinitezimálisan kicsi helyen elhelyezkedő, adott sebességű részecskéknek a statisztikáját egy időben az  $f(\vec{x}, \vec{\xi}, t)$  eloszlásfüggvény segítségével.

Az eloszlásfüggvény  $X$  térben elhelyezkedő,  $\Xi$  (mikroszkópikus) részecske sebesség térben lévő részecskék száma, mely 3D tér esetén egy 6-szoros integrál [21]:

$$N_{X,\Xi} = \int_{\vec{x} \in X} \int_{\vec{\xi} \in \Xi} f(\vec{x}, \vec{\xi}, t) d^3x d^3\xi \quad (2.7)$$

Az egyenlet során feltételezzük, hogy a részecskék tömege egységes (több komponensű rendszerek esetén külön-külön alkalmazzuk a Boltzmann egyenletet).

Ennek megfelelően a Boltzmann transzport egyenlet, ahol  $F$  a külső erőket,  $D$  a hővezetés okozta diffúziót, és  $C$  (collision) az ütközést jelenti:

$$\frac{df}{dt} = \left( \frac{\partial f}{\partial t} \right)_F + \left( \frac{\partial f}{\partial t} \right)_D + \left( \frac{\partial f}{\partial t} \right)_C \quad (2.8)$$

Izotermális (diffúziós tag nélküli) rendszerben adott pontban, adott sebességgel rendelkező részecskékre ható  $F$  erő esetén a  $t + dt$  időpillanatban a  $\mathbf{x} + \xi dt$  helyre fognak kerülni,

<sup>1</sup>A függvény megszorozva az egyik változójának valahányadik hatványával, majd integrálva a változó szerint

és új mikroszkópikus sebességük  $\vec{\xi} + \mathbf{F}dt$  lesz.

Ha feltételezzük hogy a  $dt$  idő alatt nem ütköznek egymással, akkor a számuk nem változik az új helyre érkezve:

$$f(\vec{x} + \vec{\xi}dt, \vec{\xi} + \mathbf{F}dt, t + dt) d^3x d^3\xi = f(\vec{x}, \vec{\xi}, t) d^3x d^3\xi \quad (2.9)$$

Viszont, mivel ütközés történhet, ezért a kettő különbsége adja a részecske sűrűség változását:

$$\begin{aligned} dN_C &= \left( \frac{\partial f}{\partial t} \right)_C dt d^3x d^3\xi \\ &= f(\vec{x} + \vec{\xi}dt, \vec{\xi} + \mathbf{F}dt, t + dt) d^3x d^3\xi - f(\vec{x}, \vec{\xi}, t) d^3x d^3\xi \\ &= df d^3x d^3\xi \end{aligned} \quad (2.10)$$

Ekkor  $(dt d^3x d^3\xi)$ -vel osztva és fölhasználva a teljes deriváltját  $f$ -nek kapjuk a **Boltzmann egyenlet LBM esetén használt alakját**:

$$\begin{aligned} \left( \frac{\partial f}{\partial t} \right)_C &= \frac{df}{dt} \\ &= \frac{\partial f}{\partial t} + \vec{\xi} \cdot \nabla f + \frac{d\vec{\xi}}{dt} \cdot \frac{\partial f}{\partial \vec{\xi}} \\ &= \frac{\partial f}{\partial t} + \vec{\xi} \cdot \nabla f + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \vec{\xi}} \end{aligned} \quad (2.11)$$

## 2.2. Ütközési operátor

Hivatkozunk  $\Omega$ -val a  $\left( \frac{\partial f}{\partial t} \right)_C$  parciális deriváltra, amit nevezzünk az ütközési operátornak. Ezt előbb definiálni kell, hogy használható legyen a Boltzmann egyenlet. Ehhez azzal a feltételezéssel szoktak élni, hogy csakis két test közötti ütközések történnek, valamint hogy ezek az ütköző részecskék függetlenek egymástól, vagyis sebességük közt nincs korreláció (Boltzmann erre a két feltételezésre a Stofzahlansatz néven hivatkozott) [21].

Ekkor az ütközés operátorhoz adott  $\vec{x}$  helyen  $\vec{\xi}_1$  sebességű részecskék esetén minden más  $\vec{\xi}_2$  sebességgárral figyelembe kell vennünk az ütközésük lehetőségét ami  $\vec{\xi}_{1C}$  és  $\vec{\xi}_{2C}$  egyértelműen meghatározható ütközés utáni sebességekhez vezet.

Ha a vonatkoztatási rendszerünket úgy határozzuk meg hogy  $\vec{\xi}_1$  és  $\vec{\xi}_2$  átlagával együtt mozog, akkor a részecskék sebessége ebben a vonatkoztatási rendszerben ellentétes irányú és egyenlő nagyságú, hiszen összegük 0<sup>2</sup> (Ez szintén elmondható az ütközés utáni sebességekre) [41].

Jelöljük  $\sigma$ -val az egységi hosszú, az előbb definiált vonatkoztatási rendszerben  $\vec{\xi}_1$  felé

---


$$^2(\vec{\xi}_1 - \frac{\vec{\xi}_1 + \vec{\xi}_2}{2}) + (\vec{\xi}_2 - \frac{\vec{\xi}_1 + \vec{\xi}_2}{2}) = (\vec{\xi}_1 - \vec{\xi}_1) + (\vec{\xi}_2 - \vec{\xi}_2) = 0$$

mutató vektort, hogy

$$\begin{aligned}\vec{\xi}'_1 &= \frac{\vec{\xi}_1 + \vec{\xi}_2}{2} + \frac{|\vec{\xi}_1 - \vec{\xi}_2|}{2} \boldsymbol{\sigma} \\ \vec{\xi}'_2 &= \frac{\vec{\xi}_1 + \vec{\xi}_2}{2} - \frac{|\vec{\xi}_1 - \vec{\xi}_2|}{2} \boldsymbol{\sigma}\end{aligned}\tag{2.12}$$

Ezt felhasználva az ütközési integrál definíciója [21]:

$$\Omega = \left( \frac{\partial f}{\partial t}(\vec{x}, \vec{\xi}_1) \right)_C = \int_{\boldsymbol{\sigma}} \int_{\vec{\xi}_2} B(|\vec{\xi}_1 - \vec{\xi}_2|, \boldsymbol{\sigma}) \cdot \left[ \overbrace{f(\vec{x}, \vec{\xi}_{1C}) f(\vec{x}, \vec{\xi}_{2C})}^{\vec{\xi}_1 \text{ ütközés után}} - \overbrace{f(\vec{x}, \vec{\xi}_1) f(\vec{x}, \vec{\xi}_2)}^{\vec{\xi}_1 \text{ ütközés előtt}} \right] d^3 \xi_2 d^3 \sigma\tag{2.13}$$

Ahol  $B$  az ütközési kernel mely csak a relatív sebesség nagyságának és az irányának a függvénye. Figyelembe veszi, hogy az ütközések „fordított irányban” is történhetnek, vagyis  $\vec{\xi}_{1C}, \vec{\xi}_{2C}$  bejövő sebességek ütközésének eredménye  $\vec{\xi}_1, \vec{\xi}_2$  sebességeket eredményez.

B-t úgy kell definiálni hogy nem sérti a tömeg, lendület, és energia megmaradás törvényeit.

### 2.2.1. BGK ütközési operátor

A gyakorlatban használt feltételezés mely számításképpessé teszi az ütközés operátort a Bhatnagar, Gross és Krook által definiált **BGK operátor**, mely esetén az ütközést úgy modellezzük, hogy annak hatása az, hogy visszamoszditson az adott pontban egy Maxwell-féle egyensúlyi  $f^{eq}$  eloszlásba az ütközési frekvenciával arányosan [13].

Ennek megfelelően a Boltzmann egyenlet [13]:

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + \vec{\xi} \cdot \nabla f + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \vec{\xi}} = -\frac{1}{\tau} (f - f^{eq})\tag{2.14}$$

$$f_{eq} = \frac{\rho}{(2\pi \frac{k_B}{m} T)^{\frac{3}{2}}} e^{-\frac{(\vec{\xi} - \vec{u})^2 m}{2k_B T}}\tag{2.15}$$

Ahol  $T[K]$  a hőmérséklet,  $k_B [\frac{J}{K}]$  a Boltzmann-állandó, valamint  $\tau[s]$  a relaxációs paraméter, mely meghatározza a folyadék  $\nu[\frac{m^2}{s}]$  kinematikus viszkozitását az alábbi összefüggés szerint:

$$\nu = c_s^2 \left( \tau - \frac{1}{2} \right)\tag{2.16}$$

ahol  $c_s[m/s]$  a hangsebesség.

### 2.3. LBM és az áramlástan

Ahhoz, hogy áramlástanai eseteknél használható legyen a Boltzmann egyenlet ahhoz le kell tudni vezetni belőle a Navier-Stokes egyenleteket.

A **Knudsen szám** egy dimenziótlán érték mely a rendszerünk skáláját jellemzi, és

$K_n = \frac{\lambda}{L}$ -ként van definiálva, ahol  $\lambda[m]$  egy mozgó részecske két ütközése közt eltelt átlagos hossz (mean free path), és  $L$  a rendszer jellemző hosszskálája (pl. hézag hossz melyen keresztül tömegáramlás történik).

A **Mach szám**  $M_a = \frac{U}{c_s}$  a rendszerünket jellemző  $U$  sebességet viszonyítja az adott közegbeli hangsebességhez:

Jelöljük  $\tau[s]$ -val két egymást követő ütközés között eltelt átlagos időt, valamint  $c[m/s]$ -vel az átlagos részecske sebesség nagyságát, melyre igaz, hogy  $c \approx c_s$  ( $c_s$ : Hangsebesség az adott közegben).

Ekkor:

$$\lambda = c \cdot \tau \quad (2.17)$$

$$\tau = \frac{\lambda}{c} \approx \frac{\lambda}{c_s} \quad (2.18)$$

$$\tau \approx \frac{K_n L}{U/M_a} \quad (2.19)$$

$$\tau \approx M_a K_n \frac{L}{U} \quad (2.20)$$

A kontinuitás egyenlet közvetlenül következik a BGK egyenletből (integrálunk  $\vec{\xi}$  szerint), valamint bizonyítható, hogy amennyiben elhanyagoljuk a Knudsen számot ( $K_n \rightarrow 0$ ) akkor a BGK egyenletben szereplő  $\tau$ -n keresztül le tudjuk vezetni az Euler egyenletet, valamint ha csak a Knudsen szám négyzetét hanyagoljuk el ( $K_n^2 \rightarrow 0$ ), akkor pedig a Navier-Stokes egyenletet tudjuk levezetni. Mindkét esetben úgy kezdünk hogy a Boltzmann egyenletet  $\xi_x$ -vel szorozzuk és  $\vec{\xi}$  szerinti integrálunk, majd egy folyamatos rekurzió problémába ütközve (lezárási probléma) lezárjuk az egyenletet azáltal, hogy az Euler egyenlet esetében a  $\tau$ -val szorzott tagokat, míg a Navier-Stokes esetében a  $\tau^2$ -el szorzott tagokat elhanyagoljuk [9]. Ezt követően számos lépésen keresztül történő átrendezéssel megkapjuk a kívánt egyenletet.

## 2.4. LBM diszkretizálás

Hogy gyakorlatban is használható legyen a Boltzmann egyenlet, ahhoz előbb diszkretizálni kell mind tér, sebesség, és időre nézve olyan módon, hogy a Navier-Stokes egyenletet ekkor is vissza tudjuk kapni.

A Navier-Stokes egyenlet levezetésénél abból indultunk ki, hogy a BGK egyenletet beszoroztuk  $\xi_x$ -vel és integráltuk  $\vec{\xi}$  szerint, valamint ezt követően a számolható  $M_{abc} = \int_{\vec{\xi}} f(\vec{x}, \vec{\xi}, t) \xi^a v^b \zeta^c d^3 \xi$  momentumok szerint fejeztük ki, és minden  $O(\tau^2)$  rendű tagot elhagytunk [9]. Így legfeljebb harmadrendű  $M_{300}, M_{120}, \dots$  momentumok jelennek meg, és azoknak is csak az egyensúlyi  $M_{abc}^{eq}$  változata amire van analitikus megoldásunk. Emiatt a NSE megoldásához elég a Boltzmann transzport egyenlet olyan közelítése mely megtartja a harmadrendű momentumokat. Következő lépésként pedig elvégezhetjük a diszkretizálást.

A diszkretizálás lényege hogy véges számú  $\vec{x}$  rácspontokra osztjuk a terünket, és a mikroszkópikus sebesség  $\vec{\xi}$  irányát úgy korlátozzuk, hogy szomszédos rácspont felé mutathat, legyen ez akár diagonális irányban vagy nem, és minden irányú sebesség nagyságát úgy ha-

tárazzuk meg, hogy 1 diszkrét  $\Delta t$  idő lépés alatt a következő rácsponthoz jusson ( $c = \frac{\Delta x}{\Delta t}$  rácssebesség). Ilyen sebesség nagyságok mellett elkerüljük az interpoláció szükségességét és az abból eredő számítási pontatlanságot, valamint lehetővé válik a hatékony implementálása az LBM algoritmus idő szerint léptető *terjedési* része [21].

A külön diszkrétizációs konfigurációkra  $DdQq$  módon hivatkozunk, ahol  $d$  a dimenziószám (2D/3D), és  $q$  a diszkrét mikroszkópikus irányok száma (pl. 2D esetben, ahol átlós irányt is engedünk D2Q9 konfigurációnk van, ahol figyelembe vesszük a  $\mathbf{0}$  irányt is.) [13]

A rácspontoknál külön-külön értelmezendő ütközéseket egy ún. ütközési (collision), valamint az azokat követő részecske áramlást megvalósító terjedési (streaming) lépés alatt hajtjuk végre.

### 2.4.1. Sebesség diszkrétizálása

Megmutatták, hogy az  $f^{eq}$  egyensúlyi eloszlás többdimenziós Hermite-polinom szerinti sorfejtése  $\vec{\xi}$  szerint szintén harmadrendig<sup>3</sup>, és az azt követő diszkrétizációja a Gauss–Hermite kvadratúra módszerével elegendő a megmaradási törvények betartásához és a Navier-Stokes egyenlet helyreállításához [21, 26].

$$\mathbf{H}^{(n)}(\vec{x}) = (-1)^n \frac{1}{\omega(\vec{x})} \nabla^{(n)} \omega(\vec{x}), \text{ Hermite-polinomokat tartalmazó } n\text{-ed rendű tenzor} \quad (2.21)$$

$$\omega(\vec{x}) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{x^2}{2}}, \text{ súlyfüggvény} \quad (2.22)$$

Ahol  $\nabla_{\alpha}^{(n)} = \frac{\partial}{\partial \alpha_1} \dots \frac{\partial}{\partial \alpha_n}$ , vagyis azonos  $\alpha$  bármely permutációjára (a deriválási sorrendet fölcserélhetjük)

Bizonyítható hogy a Hermite-polinomok ortogonálisak egymásra, vagyis

$$\int_{\vec{x}} \omega(\vec{x}) H_{\alpha}^{(n)}(\vec{x}) H_{\beta}^{(m)}(\vec{x}) d\vec{x} = \begin{cases} \prod_{i=1}^d n_i!, & \text{ha } n = m \text{ és } \alpha \text{ a } \beta \text{ permutációjára} \\ 0, & \text{egyébként} \end{cases} \quad (2.23)$$

, ahol  $n_i$  jelöli hogy  $x_i$  hányszor szerepel  $\alpha$ -ban.

Bármilyen  $f \in \mathbb{R}$ ,  $\int_{\vec{x}} |\mathbf{f}(\vec{x})|^2 \omega(\vec{x}) d\vec{x} < \infty$  függvény esetén kifejezhetjük azt a Hermite-polinomok segítségével:

$$f(\vec{\xi}) = \omega(\vec{x}) \sum_{n=0}^{\infty} \frac{1}{n!} \mathbf{a}^{(n)} \cdot \mathbf{H}^{(n)}(\vec{x}) \quad (2.24)$$

$$\mathbf{a}^{(n)} = \int f(\vec{x}) \mathbf{H}^{(n)}(\vec{x}) d\vec{x} \quad (2.25)$$

Végezzük el a Hermite sorfejtést  $f^{eq}(\rho, \vec{u}, T, \vec{\xi})$ -n  $\vec{\xi}$  szerint:

---

<sup>3</sup>Egyes cikkek szerint javíthatja a numerikus stabilitást és pontosságot egy magasabb rendű sorfejtés használata.

$$f^{eq}(\rho, \vec{u}, T, \vec{\xi}) = \omega(\vec{\xi}) \sum_{n=0}^{\infty} \frac{1}{n!} \mathbf{a}^{(n),eq}(\rho, \vec{u}, T) * \mathbf{H}^{(n)}(\vec{\xi}) \quad (2.26)$$

$$\mathbf{a}^{(n),eq}(\rho, \vec{u}, T) = \int_{\vec{\xi}} f^{eq}(\rho, \vec{u}, T, \vec{\xi}) \mathbf{H}^{(n)}(\vec{\xi}) d\vec{\xi} \quad (2.27)$$

$f^{eq} = \frac{\rho}{(2\pi \frac{k_B T}{m})^{\frac{3}{2}}} e^{-\frac{(\vec{\xi}-\vec{u})^2 m}{2k_B T}}$  esetén látható hogy fölírhatjuk  $\omega$  segítségével tetszőleges  $d$  dimenzió esetén:

$$f^{eq}(\rho, \vec{u}, T, \vec{\xi}) = \frac{\rho}{(2\pi \frac{k_B T}{m})^{\frac{d}{2}}} e^{-\frac{(\vec{\xi}-\vec{u})^2 m}{2k_B T}} = \frac{\rho}{(\frac{k_B T}{m})^{d/2}} \omega\left(\frac{\vec{\xi}-\vec{u}}{\sqrt{\frac{k_B T}{m}}}\right) \quad (2.28)$$

$$\mathbf{a}^{(n),eq} = \frac{\rho}{(\frac{k_B T}{m})^{d/2} \int_{\xi} \omega\left(\frac{\vec{\xi}-\vec{u}}{\sqrt{\frac{k_B T}{m}}}\right)} \mathbf{H}^{(n)}(\vec{\xi}) d\vec{\xi} \quad (2.29)$$

$$\vec{\eta} := \frac{\vec{\xi}-\vec{u}}{\sqrt{\frac{k_B T}{m}}} \quad (2.30)$$

$$\mathbf{a}^{(n),eq} = \rho \int_{\vec{\xi}} \omega(\vec{\eta}) \mathbf{H}^{(n)}\left(\sqrt{\frac{k_B T}{m}} \vec{\eta} + \vec{u}\right) d\vec{\eta} \quad (2.31)$$

, melynek az analitikus megoldásai harmadrendig:

$$a^{(0),eq} = \rho \quad (2.32)$$

$$\mathbf{a}^{(1),eq} = \rho \vec{u} \quad (2.33)$$

$$a_{\alpha\beta}^{(2),eq} = \rho \left( u_{\alpha} u_{\beta} + \left( \frac{k_B T}{m} - 1 \right) \delta_{\alpha\beta} \right) \quad (2.34)$$

$$a_{\alpha\beta\gamma}^{(3),eq} = \rho \left[ u_{\alpha} u_{\beta} u_{\gamma} + \left( \frac{k_B T}{m} - 1 \right) (\delta_{\alpha\beta} u_{\gamma} + \delta_{\beta\gamma} u_{\alpha} + \delta_{\gamma\alpha} u_{\beta}) \right] \quad (2.35)$$

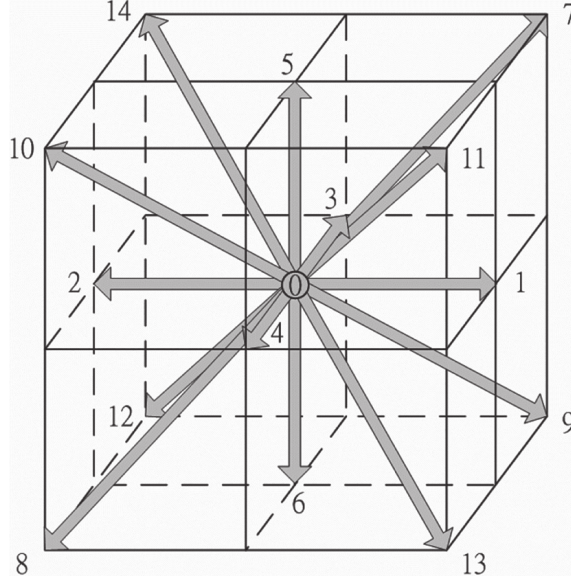
Ha harmadrendig közelítjük  $f^{eq}$ -et, akkor  $\vec{\xi}$  szerint másodrendig:

$$f^{eq}(\rho, \vec{u}, T, \vec{\xi}) \approx \omega(\vec{\xi}) \sum_{n=0}^N \frac{1}{n!} \mathbf{a}^{(n),eq} \cdot \mathbf{H}^{(n)} \quad (2.36)$$

$$f^{eq}(\rho, \vec{u}, T, \vec{\xi}) \approx \omega(\vec{\xi}) \rho \left[ 1 + \vec{\xi} \cdot \vec{u} + \left( \vec{\xi} \cdot \vec{u} \right)^2 - \vec{u} \cdot \vec{u} + \left( \frac{k_B T}{m} - 1 \right) (\vec{\xi} \cdot \vec{\xi} - d) \right] \quad (2.37)$$

$$f^{eq}(\rho, \vec{u}, T, \vec{\xi}) \approx \omega(\vec{\xi}) \rho Q(\vec{u}, T, \vec{\xi}) \quad (2.38)$$

A Gauss–Hermite kvadratura alapján közelíthetjük  $f^{eq}$  harmadrendű Hermite sorfejtését diszkrét  $n$  darab mintavételi  $\xi_i$  pontokkal, ami lehetővé teszi a sebesség diszkretizálását:



**2.1. ábra.** A  $D3Q15$  diszkrét sebesség konfiguráció, ahol vízszintes  $(y-1, y, y+1)$  szeletenként 5 sebesség van [41].

$$\int_{\vec{x}} \omega(x) g(\vec{x}) d\vec{x} = \sum_{i=1}^n w_i f(x_i) \quad (2.39)$$

Ehhez az kell, hogy a pontok száma  $n$ -re igaz legyen hogy  $n \geq (N+1)/2$  ahol  $N$  a Hermite-Polinom maximálisan vett rendje.

Megtéve ezt  $\vec{\xi}$  szerint:

$$\mathbf{a}^{(n),eq} = \int_{\vec{\xi}} f^{eq}(\vec{\xi}) \mathbf{H}^{(n)}(\vec{\xi}) d\vec{\xi} = \rho \int_{\vec{\xi}} \omega(\vec{\xi}) Q(\vec{\xi}) \mathbf{H}^{(n)} d\vec{\xi} \quad (2.40)$$

$$= \rho \sum_{i=1}^n w_i Q(\vec{\xi}_i) \mathbf{H}^{(n)}(\vec{\xi}_i) \quad (2.41)$$

Megfelelő  $\{\vec{\xi}_i\}_i$  diszkrét sebesség halmaz kiválasztása után és izotermális feltételezés után, valamint fölvéve az új  $\vec{c}_i = \frac{\vec{\xi}_i}{\sqrt{3}}$  sebességeket levezethető a diszkrét-sebesség Boltzmann egyenlet, valamint BGK esetben a végső egyenlet a diszkrét  $f^{eq}$ -ra [27]:

Diszkrét-sebesség Boltzmann egyenlet

$$\frac{\partial f_i}{\partial t} + \vec{c}_i \cdot \nabla f_i = \Omega(f_i), \forall i : 0 \leq i < q \quad (2.42)$$

### LBM BGK Egyensúlyi Eloszlásfüggvény

$$f_i^{eq} = w_i \rho \left( 1 + \frac{\vec{c}_i \cdot \vec{u}}{c_s^2} + \frac{(\vec{c}_i \cdot \vec{u})^2}{2c_s^4} - \frac{\vec{u} \cdot \vec{u}}{2c_s^2} \right) \quad (2.43)$$

Ahol  $c_s \left[ \frac{\Delta x}{\Delta t} \right]$  a fénysebesség és  $w_i$  az adott irányra meghatározott súly<sup>a</sup>.

<sup>a</sup>Ezekre többek közt olyan igények fogalmazódnak meg, mint például hogy ne sértsék a megmaradási törvényeket, biztosítsák adott rendig a rotációs izotrópiát

Az egyensúlyi eloszlásfüggvényhez a sebesség és nyomás, mint makroszkopikus változók kiszámítása szükséges [27]:

### LBM Makroszkopikus Változók

$$\rho [-] = \sum_{i=1}^q f_i \quad (2.44)$$

$$\vec{u} \left[ \frac{\Delta x}{\Delta t} \right] = \frac{1}{\rho} \sum_{i=1}^q f_i \vec{c}_i \quad (2.45)$$

ahol  $q$  a diszkrét sebesség irányok száma.

## 2.4.2. Tér és idő diszkretizálása

A (2.42) egyenletnél tudjuk, hogy  $\vec{\xi} = \frac{d\vec{x}}{dt} = \vec{c}_i$ , és baloldalt  $f_i$ -nek a teljes deriváltja  $\frac{Df_i}{Dt}$  szerepel.

Egy másik meglátás, hogy  $\vec{x}$  az idő szerint függ oly módon, hogy a sebesség a konstans  $c_i$ , vagyis  $x = x_0 + c_i(t - t_0) = x_0 + c_i \Delta t$ , valamint hogy  $\Omega_i(f_i) = \Omega_i(\vec{x}, t)$ , mivel  $f_i$  csak  $\vec{x}, t$ -től függ (a sebesség konstans) [21].

Ennek megfelelően integráljuk (2.42)-t  $t_0$ -tól  $t_0 + \Delta t$ -ig:

$$f_i(\vec{x}_0 + \vec{c}_i \Delta t, t_0 + \Delta t) - f_i(\vec{x}_0, t_0) = \int_{\zeta=0}^{\Delta t} \Omega_i(\vec{x}_0 + \vec{c}_i \zeta, t_0 + \zeta) d\zeta \quad (2.46)$$

A  $t_0$  és  $x_0$  ekkor vehető tetszőleges diszkrét időpontnak és térbeli pontnak.

A jobb oldalnál el kell dönteni, hogy milyen módon lesz az integrál közelítve. A leggyakoribb megoldás, hogy elsőrendben közelítünk  $\Delta t \cdot \Omega_i(\vec{x}, t)$ -vel (BGK ütközési operátor esetén a (2.47) egyenlethez jutunk) [21].

## 2.4.3. Algoritmus megvalósítás

Az  $f$  eloszlásfüggvény idő szerinti frissítését/léptetését két részre bontva valósítjuk meg.

Az első az ütközési (collision) szakasz, amikor (DdQq konfigurációt esetben) az ütközés során minden diszkrét  $\Delta x^d$  térrészt elfoglaló rácsponton belül a valamennyi megmaradási törvénynek megfelelően<sup>4</sup> újrendeződik az  $f$  részecske eloszlás az ütközések hatásaként.

<sup>4</sup>A megmaradási törvények megfelelő momentumok nem változnak

A BGK egyenlet értelmében ez az újrendeződés az egyensúlyi  $f^{eq}$  eloszlás felé történik [21].

#### LBM BGK Ütközési Szakasz

$$f_i^* = f_i - \frac{\Delta t}{\tau} (f_i - f_i^{eq}), \forall i : 0 \leq i < q \quad (2.47)$$

ahol  $f_i^*$  az  $i$ -edik diszkrét irány ütközés utáni eloszlása adott helyen adott időben és  $q$  a diszkrét irányok száma.

Diszkrét esetben  $\Delta t$  és  $\tau$  együttesen határozzák meg a  $\nu$  kinematikus viszkozitást. Ennek megfelelően gyakorlatban a dimenziótlan relaxációs paramétert  $\tau^* = \frac{\tau}{\Delta t}$ -t használjuk. A kinematikus viszkozitás és  $\tau^*$  közötti összefüggés [21]:

$$\nu = c_s^{*2} (\tau^* - 1/2) \frac{\Delta x^2}{\Delta t} \quad (2.48)$$

ahol  $c_s^* \left[ \frac{\Delta x}{\Delta t} \right]$  a rács mértékegységben vett hangsebesség, mely értéke az általánosan használt D2Q9, D3Q15, D3Q19, D3Q27 konfigurációk esetén  $\frac{1}{\sqrt{3}}$ .

Az ütközés utáni (post-collision) eloszlás ezt követően minden egyes  $\vec{\xi}_i$  diszkrét sebességre annak irányában halad tovább a következő rácspontra. Ezt hívjuk a terjedési (streaming) szakasznak [27].

#### LBM Terjedési Szakasz

$$f_i(\vec{x} + \vec{c}_i, t + \Delta t) = f_i^*(\vec{x}, t), \forall i : 0 \leq i < q \quad (2.49)$$

Ahol  $q$  a diszkrét irányok száma.

Bizonyítható, hogy ez a két szakasz egymást követően megvalósítja a (2.14) BGK egyenlet diszkrétizált változatát, ahhoz konvergál  $\Delta t \rightarrow 0$  esetben [13].

### 2.4.4. Stabilitási megfontolások

#### $\tau^*$ megválasztása

Hogy megvizsgáljuk  $\tau^*$  stabilitását, feltételezzük hogy a tér bármely diszkrét pontjában jelenleg azonos az eloszlásunk. Ennek megfelelően nem lesz hatása a terjedési szakasznak, mivel minden egyes diszkrét csomóponton azonos változás lesz propagálva  $\Delta t$  időlépés alatt.

Ennek megfelelően jelölhetjük  $f_i^*$ -t  $f(t + \Delta t)$ -ként:

$$f_i(t + \Delta t) = f_i(t) - \frac{1}{\tau^*} (f_i(t) - f_i^{eq}) \quad (2.50)$$

$$f_i(t + \Delta t) = \left( 1 - \frac{1}{\tau^*} \right) f_i(t) + \frac{1}{\tau^*} f_i^{eq} \quad (2.51)$$

Ez egy diszkrét idejű relaxációs egyenlet (az exponenciális bomlás diszkrét analógja),

melynél ha  $\tau < 1/2$  akkor egyre növekvő nagysággal oszcillálna  $f_i$  az egyensúlyi  $f_i^{eq}$  körül, vagyis a stabilitás szükséges feltétele, hogy  $\tau^* \geq \frac{1}{2}$  (a gyakorlatban valamennyivel nagyobbak kell lennie ennél a számítási pontosság miatt) [21].

### Maximális sebesség

Intuitívan érezhető, hogy amennyiben egy makroszkópikus folyadék sebesség  $\vec{u}$  nagyobb  $\frac{\Delta x}{\Delta t}$ -nél, akkor a szomszédok közötti információ áramlást megvalósító terjedési szakasz helytelenné válik, mivel képtelen a 2 rácstávolságra lévő csomópontokhoz információt juttatni  $\Delta t$  időlépés alatt [21].

Ezt formálisan a Courant–Friedrichs–Lewy feltétel fogalmazza meg a konvergenciára vonatkozóan, és a fönti jelenséget a  $C = |\vec{u}| \frac{\Delta t}{\Delta x}$  Courant számmal jellemzi, mely összehasonlítja a folyadék  $\vec{u}$  advekción sebességét az információ  $\frac{\Delta x}{\Delta t}$  propagációs sebességével [21].

$C \leq 1$  szükséges feltétel, valamint egyes módszereknél kiszámolható egy  $C_{max} < 1$  érték, mely szigorít ezen, vagyis  $C \leq C_{max}$ -nak is teljesülnie kell. Az LBM esetén erősen peremfeltétel és paraméter függő, hogy a stabilitás mely  $C$  értéknél sérül, viszont irányelveként használható, hogy  $|\vec{u}|_{max} < \frac{1}{\sqrt{3}} \frac{\Delta x}{\Delta t} \approx 0.577 \frac{\Delta x}{\Delta t}$  [21]. Ennek átlépését lehet figyelmeztetésként jelezni egy keretrendszerben.

## 2.5. Peremfeltételek

Valós folyadékáramlási problémáknál a folyadékon kívül szilárd (fix vagy mozgó) objektumok, előre ismert (meghatározott) sebességek, sebességgradiensek, nyomások, nyomásgradiensek, és egyéb azonos időben zajló folyamatok jelenhetnek meg, és egy CFD módszernél elvárás hogy ezeket kezelni tudja.

Az LBM mezoszkópikus volta miatt sokszor azonos hidrodinamikai viselkedést lehet elérni különböző egyenletek (algoritmusok) megvalósításával, ennek megfelelően a peremfeltétel címszó alatt máig jelennek meg új algoritmusok melyek jobb teljesítményt és/vagy magasabb rendű pontosságot kínálnak [47, 7].

A peremfeltétel algoritmusoknál az egyenes peremfeltételeket a szakirodalom kétféle csoportra bontja, a link-wise csoportra, ahol a peremfeltétel két diszkrét pont között helyezkedik el, valamint a wet-node csoportra, ahol pedig magán a diszkrét ponton értelmezzük (és ennek megfelelően módosítja pl. az ütközési szakaszt). Az olyan módszerek mint az immersed boundary method esetén a rácson kívüli struktúra erőnkön keresztül fejt ki a hatását a folyadékon (és fordítva), és ez nem tartozik semelyik csoportba a kettő közül [21].

### Periódikus feltételek

Ismétlődő áramlási struktúrát tartalmazó rendszerben (pl. egyenes csőben történő lamináris folyás) úgy egyszerűsíthetjük a problémát, hogy adott dimenzióban engedjük a periódicitást, vagyis hogy az egyik oldalon kimenő folyadék az ellentétes oldal bemenete.

Az LBM esetén könnyen implementálható ez, csupán a kimenő  $f_i$  populációt az ellentétes bemenetéhez streameljük [27]:

$$f_i(\vec{x} + \vec{L}, t + \Delta t) = f_i^*(\vec{x}, t) \quad (2.52)$$

ahol  $L$  a periodicitás iránya és nagysága az átvitt hossz.

Gyakorlatban a szimulációs határok mentén implementáljuk a periodicitást  $x$ ,  $y$ , vagy  $z$  irányban a  $\%$  maradék operátor segítségével.

### Visszapattanási séma

A visszapattanási (Bounce-back) módszer a súrlódásmentes (no-slip) felületeket valósítja meg, és gyakorta használt szimulációk fölállításakor [13]. Azon az elven alapszik, hogy a szilárd felülethez ütköző részecske populációk visszaverődnek az ütközéssel ellentétes irányba. A visszaverődés miatt a felületre merőleges sebesség 0, és mivel visszaverődéskor a felülettel párhuzamos sebesség iránya ellentétesé válik, vagyis nincsen spekuláris (tükröszerű) visszaverődés. Ezért nem „csúszik” a folyadék, tehát nem súrlódásmentes a felület.

Kétféleképpen lehet megvalósítani a visszapattanási sémát. Az első lehetőség, hogy a következő  $t + \Delta t$  lépésben már megtörténik a visszaverődés, vagyis a felülettel ütközés a  $t + \frac{1}{2}\Delta t$  időlépésben történik meg. Ezt hívják félúti visszapattanásnak (halfway bounce-back) [21]. Mivel ekkor nincs szükség az ütközés irányába lévő szilárd csomópontra, ezért az ütközési szakasz változtatása nélkül, közvetlenül a terjedési (streaming) szakasznál implementálhatjuk.

A második, hogy a  $t + \Delta t$  lépésben eljut a felülethez, és  $t + 2\Delta t$  lépésnél jut vissza az eredeti csomóponthoz. Ezt hívják teljes úti visszapattanásnak (fullway bounce-back) [21]. Ekkor a szilárd csomópontra értelmezett ütközési szakaszt kell módosítani.

Mindkét esetben nagyjából a két csomópont közt valósul meg az ütközési felület, valamint míg a teljes úti visszapattanás elsőrendű  $O(\Delta t)$  hibát, addig a félúti visszapattanás másodrendű  $O(\Delta t^2)$  hibát eredményez<sup>5</sup>.

A félúti visszapattanás implementálása ennek megfelelően:

$$f_{\bar{i}}(\vec{x}_b, t + \Delta t) = f_i^*(\vec{x}_b, t) \quad (2.53)$$

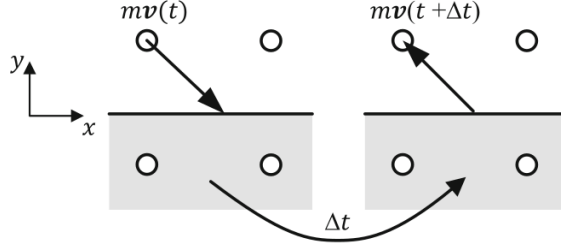
ahol  $\bar{i}$  az  $i$  iránnyal ellentétes irány indexe.

A fenti logika szerint a felülettel párhuzamos mozgást engedélyező free-slip peremfeltételt úgy lehet megvalósítani, hogy csakis a felületre merőleges irányt fordítjuk meg.

### Dirichlet peremfeltételek

A Dirichlet peremfeltétel fix értéket határoz meg adott makroszkópikus változónak egy adott helyen. Egy fix sebességű mozgó fal (vagy bemenet/kimenet) esetén a bounce-back sémát annak megfelelően módosítjuk, hogy a részecske perdülete a megszabott  $\vec{u}_w$  fal

<sup>5</sup>A másodrendű hiba jobb skálázású, hiszen  $\Delta x$  csökkentésével azzal négyzetesen arányosan csökken a hiba.



**2.2. ábra.** *Visszapattanási algoritmus, mely során a felülettel párhuzamos és merőleges komponensek mind megfordulnak irányukban [21]. Az ábrán  $v$  nem a mikroszkópikus folyadékssebességet, hanem a részecske-sebességet jelenti.*

sebesség szerint változik [21]:

$$f_i(\vec{x}_b, t + \Delta t) = f_i^*(\vec{x}_b, t) - 2w_i\rho_w \frac{\vec{c}_i \cdot \vec{u}_w}{c_s^2} \quad (2.54)$$

ahol a  $w$  index a  $\vec{x}_w = \frac{1}{2}\vec{c}_i\Delta t$  fal helynél jelölt jellemzőket érti. Mivel  $\rho_w$  értéke ismeretlen ekkor, ezért közelíteni lehet pl.  $\rho(\vec{x}_b)$ -val.

Egy bemenet/kimeneten meghatározott fix  $p$  nyomás  $\rho_w(p)$ -n keresztül az ún. anti-bounce-back módszer segítségével lehet megvalósítani:

$$f_i(\vec{x}_b, t + \Delta t) = f_i^*(\vec{x}_b, t) + 2w_i\rho_w \left[ 1 + \frac{(\vec{c}_i \cdot \vec{u}_w)^2}{2c_s^4} - \frac{\vec{u}_w^2}{2c_s^2} \right] \quad (2.55)$$

Mivel  $\vec{u}_w$  értéke ismeretlen, ezért becsülnünk kell. Ezt lehet becsülni például a peremfeltételi  $\vec{x}_b$  csomópont és a felület befelé mutató normálisa irányában lévő következő  $\vec{x}_{b+1}$  csomópont sebességeinek fölhasználásával:  $\vec{u}_w \approx \vec{u}(\vec{x}_b) + \frac{1}{2} [\vec{u}(\vec{x}_b) - \vec{u}(\vec{x}_{b+1})]$  [21]

## 3. fejezet

# Taichi Keretrendszer

A Taichi egy 2019 óta létező Python-ba ágyazott keretrendszer, mely segít könnyen írható, hordozható (platform független), és magas teljesítménnyel rendelkező parallelizált programok írásában, melyek futtathatóak akár videokártyán vagy processzoron. Az MIT, ETH Zürich, és a Stanford Egyetem oktatási célokra használja szimulációs kurzusok keretei között, valamint az OPPO telefon cég interaktív háttérképek létrehozására veszi igénybe [43].

Azáltal, hogy a Python szintaxisát használja és a GPGPU programozás módszereit (pl. grid-stride loop használatát) absztrahálja, ezáltal könnyíti a fizikai szimulációk és számításigényes algoritmusok írását, olvashatóságát, és ezért nemcsak programozók, hanem a Python nyelvezetét ismerő mérnökök is hozzájárulhatnak az algoritmus implementációhoz. Videokártyás futtatás esetén támogatja mind a CUDA, Vulkan, és Metal keretrendszeren való futtatást, és ez lehetővé teszi hogy széles körű számítógép konfigurációkon lehessen futtatni a programunkat. Erre egy példa, hogy a dolgozat témáját képző áramlástan megoldó futtatható akár egy 2015-ben megjelent AMD R9 390 videokártyán, vagy egy M1 SoC-vel rendelkező MacBook-on.

Szélesebb körű hardveren teszi lehetővé a futtatást olyan hasonló megoldásokhoz képest, mint a Numba keretrendszer (mely csak CUDA-val támogat videokártyán futtatást), vagy az Nvidia 2021-ben megjelent Warp keretrendszere (melyet szerkezetében és működésében ihletett a Taichi), mivel ezek csak CUDA-s videokártyás futtatást támogatnak.

A Taichi egy másik hasznos része az, hogy támogatja a Template-ek lehetőségeit anélkül, hogy ez külön szintaxist igényelne, vagyis a Python alapértelmezett típusai hivatkozásai (pl. adott lebegőpontos vagy egész értéknél) a kompilálásnál konstansnak értelmeződnek, ennek megfelelően nem jár teljesítmény eséssel (viszont futás közben változó értékekre nem használható).

Mivel a Taichi függvényeket osztálybeli metódusokként is lehet definiálni, melyek ezen a módon elérhetik az adott objektum adatait (mint konstant értékeket), ebben az értelemben jóval átláthatóbb kód készíthető egy keretrendszer fejlesztésekor a Numba-hoz képest, mivel ott csak statikus metódusként lehet algoritmust definiálni<sup>1</sup>.

---

<sup>1</sup>Kísérleti módon ott is van lehetőség arra, hogy osztályon belüli metódusokat használjunk (jitclass), de akkor nincs lehetőség változó osztály mezőkre, szimulációs konfigurációtól függően változó dimenziójú tömbök használatára (pl. 2D/3D)

A keretrendszer hátrányai közé tartozik a teljesítmény profilozás nehézsége, melyet csak minimálisan támogat (csak a kernelben eltöltött időt, és olvasási, írási műveletek idejét jegyzi le) amennyiben nem CUDA-n futtatunk. Ezt javíthatja, ha a `func` dekorátor helyett `real_func` dekorátort használjuk a kerneleken belül használt függvények definíciójánál, viszont ekkor nem használhatjuk ki a Template szerű viselkedést, mivel pontosan definiálni kell a paraméterek típusát (egészen a lebegő pontos értékek típusáig, a vektor és mátrix dimenziókhöz).

Egy másik hátránya az, hogy nincs a lehetőségünk a GPGPU programozási tapasztalatokból eredő optimalizálásra, mivel jelentősen absztrahál a nyelv CUDA és OpenCL-hez képest.

### 3.1. Használat

Ahhoz, hogy egy függvényt Taichi segítségével futtassuk Pythonból, ahhoz jelölni kell a `@kernel` dekorátor segítségével, az ezen belül használt függvényeket pedig a `@func` dekorátorral. Míg a kernel esetén explicit jelölni kell a paraméterek adattípusát (a self pointeren kívül) és az esetleges visszatérési érték típusát, addig a `func` függvényeknél ez nem szükséges.

A `func` függvények valójában inline módon értelmeződnek, vagyis nem valódi függvényhívást jelentenek az adott eszközön, hanem mint ha a függvény kódjának tartalma be lenne másolva a meghívott helyre (valódi függvény definíálásához a `@ti.real_func` dekorátort kell használni, viszont ekkor a paraméterek típusát explicit definiálni kell).

A legfőbb osztály a Taichinál a `Field`, mely tömbként teszi elérhetővé az adott típusú értékeket, legyen ez skalár vagy Vektor vagy Mátrix (pl. minden diszkrét pontnál egy sebesség vektor). Ennek a memória elrendezése tetszőlegesen változtatható anélkül, hogy az adatok elérési módja változna (Lásd: SNode rendszer alfejezet)

Mivel az elért skalár értékek automatikusan konstansként értelmeződnek, ezért ha egy változó skalár értéket akarunk, ahhoz egy „0 dimenziós” Taichi `Field`-et kell definiálni, melyet a kódban `[None]`-al kell indexelni, akár el akarjuk érni a változót vagy írni akarjuk.

Gyakran hasznos a `ti.static()` kulcsszó (C++ megfelelője a `constexpr if`), amely támogatja, hogy az algoritmusok egyes részei másképpen fussanak le a szimuláció során anélkül, hogy ez a teljesítmény rovására menjen, vagyis előírja, hogy a statikus `if` elágazás melyik része fusson le már kompilálás közben. Ilyen például az, hogy térfogati erő nélküli szimulációban ne kelljen lefutnia feleslegesen az azt megvalósító kódnak, vagy hogy a vizualizálás a megfelelő módon történjen mind 2D, 3D esetben.

Egy másik funkciója a `ti.static()`-nek az úgynevezett loop unrolling, vagyis ha alkalmazuk a `ti.static(range(X))` függvényt, ahelyett hogy egy növekvő számlálóval minden iteráció végén történjen egy logikai elágazás<sup>2</sup>, ahelyett X-szer egymás után bemásolódik ugyanaz a kód. Ekkor a számlálóra történő hivatkozások be lesznek égetve iterációnként más-más értékre.

---

<sup>2</sup>Loop esetén az elágazás azt vizsgálja, hogy a számláló elérte-e az X-1 végértékét. Ha igen, akkor megnövelve a számlálót visszamegy az iteráció kód elejére, különben átlép az iteráció utáni kódra

Egy másik lehetőség, hogy a Taichi CPU és CUDA runtime esetén lehetővé teszi külső kód futtatását. Ez történhet úgy, hogy közvetlenül átadjuk magát a forráskódot (.c, .cpp, .cu), vagy a könyvtárat (.so, .dylib, .dll, .ll). Ez hasznos lehet, amikor meglévő lineáris algebra könyvtárakat kívánunk használni egyenletek megoldásakor, vagy jelen esetben akkor, amikor vektorprocesszor utasításokat akarunk használni CPU-n, mivel jelenleg a Taichi nem támogatja azokat.

A vizualizációt rendkívül könnyű megvalósítani, mivel csak egy ablakot kell létrehozni a `ti.GUI` osztály példányosításával, majd a méretének megfelelően egy azonos dimenziójú Field-et kell átadni a `set_image()` metódus segítségével (lásd: 3.1. ábra).

### 3.1. kódrészlet. *Taichi vizualizációs kódrészlet*

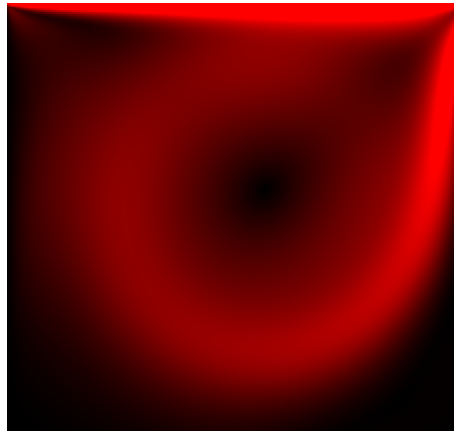
---

```

1 @ti.kernel
2 def draw(self, pixel: ti.template()):
3     for i, j in self.lbm.data.vars.population_array:
4         vel = self.lbm.velocity_at_LBM_taichi(ti.Vector([i,j]))
5         vel_mag = vel.norm()
6
7     for k_1 in ti.static(range(self.rescale_factor)):
8         for k_2 in ti.static(range(self.rescale_factor)):
9             i_draw, j_draw = (
10                 i * self.rescale_factor + k_1,
11                 j * self.rescale_factor + k_2,
12             )
13             pixel[i_draw, j_draw][0] = vel_mag * 13

```

---



**3.1. ábra.** *Példa a Taichin keresztül elérhető vizualizációra Lid Driven Cavity áramlástan benchmark esetén*

### 3.2. SNode rendszer

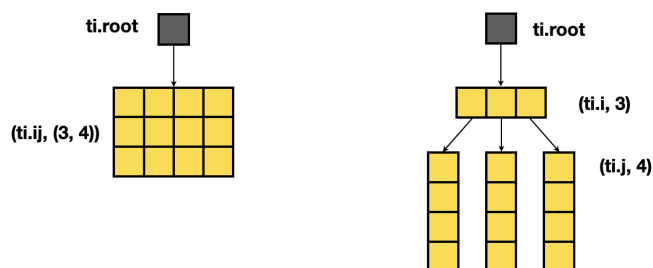
Taichinál az adatszerkezetek alapját az SNode rendszer képezi, amely hierarchikus formában szervezi a különféle adattípusokat. Az SNode fák csomópontokat tartalmaznak, amelyek lehetnek sűrű adattömbök vagy mutatók. Minden csomópont különböző típusú adatokat, például tömböket vagy skalárokat tárolhat. Ennek megfelelően kialakítható sor szerinti (row-major) vagy oszlop szerinti (column-major) memória elrendezés.

Egy másik lehetőség a blokkosítás, amely a 2D/3D teret blokkokra osztva azokon belül

sor szerinti vagy oszlop szerinti elrendezést biztosít, mely javíthatja a teljesítményt azáltal, hogy megfelelő blokk méret esetén a szomszédos elem elérésekor az már nagyobb eséllyel benne lesz a gyorsítótárban. Gondoljunk arra, hogy például a sor szerinti elrendezés esetén egy oszlop irányban szomszédos adat elérésekor a tömb sorhosszával ugrik az elhelyezkedés a memóriában. Ekkor viszont az pedig még nincs (előre) betöltve a gyorsítótárban, ezért újonnan memória hozzáférés szükséges<sup>3</sup>.

A `Field`-ek esetén ha meghatározzuk az alakját (`shape` paraméter), akkor az alapértelmezetten sor szerinti memória elrendezést kap, ellenkező esetben pedig a 3.2 ábra szerinti szintaxissal (ahol `x` az alak nélküli `Field` változó) lehet egy tetszőleges memória elrendezést biztosítani.

```
ti.root.dense(ti.ij, (3, 4)).place(x)   ti.root.dense(ti.i, 3).dense(ti.j, 4).place(x)
```



**3.2. ábra.** Az *SNode* rendszernél a jobb oldalt látható sűrűn pakolt 1 dimenziós tömbök egy memóriában sor szerinti (row-major) sorrendben helyezkednek el, megegyezve a bal oldali 2 dimenziós tömb definiálásával [42].

<sup>3</sup>Memória eléréskor nem csak a kívánt adat, hanem egy fix méret egyszerre töltődik be, és a gyorsítótárban tárolódik. Ennek az implementációnak megfelelően összességében kevesebb számú memória hozzáférésre lehet szükség

## 4. fejezet

# Implementáció

A kód Python 3.11-ban és Taichi 1.7.2 verzió alatt íródott, és a struktúrája úgy lett kialakítva (lásd: 4.1. ábra), hogy mind a statikus és dinamikus (időlépésenként változó) adatot az `LBMDData` nevű osztály foglalja össze, míg az azokat felhasználó algoritmusok osztályai referenciát kapnak hozzá.

### 4.1. Használat

A Lattice Boltzmann Módszert összefoglaló LBM osztály példányosításával lehet a szimulációkat fölállítani. A konstruktorában egy `parameters` szótár (dictionary) objektumot adunk át, mely a szimuláció egészére vonatkozó (nem csomópont függő) beállításokat tartalmazza, majd azt követően a `SetupIndexer2D/SetupIndexer3D` osztálybeli objektummal kell felállítani a peremfeltételeket, a térfogati erőket pedig az LBM osztály `add_force` metódusával kell hozzáadni.

Az LBM osztály példányosításakor megadandó konstruktor paraméterek:

- **Kötelező bemenetek:**

- `dimensions: List[int]`: A csomópontok felbontásai minden dimenzióra (pl. `[256, 256]`)
- `Q: int`: A diszkrét makroszkopikus sebességek száma
- `lengths_SI: List[float]`: A rendszer fizikai hossza méterben (pl. `[1.5, 1.3]`)
- `kinematic_viscosity_SI: float`: A szimulált folyadék kinematikai viszkozitása  $[m^2/s]$ -ban
- `collider: Class[Collider]`: Az alkalmazott ütközési algoritmus osztálya (pl. `ColliderBGK`, `ColliderTRT`)
- `boundary_conditions: BoundaryConditions`: A különböző típusú csomópontokhoz használt peremfeltételek (pl. `BCDefault()`)

- **Opcionális bemenetek:**

- `has_F: bool = False`: `True`-ra kell állítani külső térfogati erő használata esetén (hozzáadva az `LBM::add_force(self, f: Force)` függvénynel)

- `forcing_scheme: Class[ForcingScheme] = ForcingSchemeEmpty`: Az erőrend-szer osztálya, ha erők használatára van szükség
- `pressure_0_SI: float = 101325`: A légköri nyomás [Pa]-ban
- `unit_density_SI: float = 1.0`: A  $\rho = 1.0$  egységnyi sűrűség értéke [ $kg/m^3$ ]-ben
- `lattice_border_types: List[BorderType] = [BorderType.NON_PERIODIC_BORDER]*D`: Megadja, hogy egy adott dimenzió periodikus vagy nem.

• **Választható bemenetek, amelyek közül csak az egyik választható egyszer-re:**

- (a) `dt: float`: A használandó diszkrét időlépés. Csökkentése alacsonyabb  $\tau^*$ -ot eredményez.
- (b) `tau_dimensionless: float`: A használni kívánt  $\tau^*$  dimenziótlan relaxációs ráta (pl. 0.55). Nagyobbnak kell lennie, mint 0,5. Csökkentése alacsonyabb `dt`-t eredményez.

#### 4.1.1. Lid-driven Cavity 2D példakód

A használatához egy példa kód egy Lid-driven Cavity 2D áramlás megvalósítására (5.1.1. alfejezet):

##### 4.1. kódrészlet. *Lid-driven cavity kódrészlet*

---

```

1 U = lid_velocity # [m/s]
2 L_characteristic = lid_dimension # [m]
3 kinematic_viscosity_SI = U * L_characteristic / Re # [m^2/s]
4 resolution = 256
5
6 parameters = {
7     "kinematic_viscosity_SI": kinematic_viscosity_SI, # [m^2/s]
8     "dimensions": (resolution, resolution),
9     "Q": 9,
10    "collider": ColliderBGK if not use_trt else ColliderTRT,
11    "tau_dimensionless": tau_dimensionless, # [s]
12    "lengths_SI": [lid_dimension, lid_dimension], # [m]
13    "boundary_conditions": BCDefault(),
14 }
15
16 lbm = LBM(parameters)
17
18 # boundary conditions
19 dx, dt = lbm.data.gp.dx, lbm.data.gp.dt
20 vel_top_SI = [lid_velocity, 0.0]
21 setup_indexer = SetupIndexer2D(lbm)
22
23 for WALL in [
24     setup_indexer.WALL_LEFT,
25     setup_indexer.WALL_RIGHT,
26     setup_indexer.WALL_BOTTOM,
27 ]:
28     setup_indexer.apply_solid(WALL)
29 vel_top_LBM = setup_indexer.apply_dirichlet_velocity(

```

```

30     setup_indexer.WALL_TOP, vel_top_SI
31 )
32
33 lbm.init()
34
35 t = 2 #s
36 num_steps = t // lbm.data.gp.dt
37 for _ in range(num_steps):
38     lbm.collide_stream()
39
40 lbm.write_to_vtk('output/run.vti')

```

Az első 3 sorban meghatároztunk egy felső fedél pozitív  $x$  irányban történő mozgásának sebességét, majd a fedél hosszát a karakterisztikus hosszunk véve kiszámoltuk a  $\nu$  kinematikus viszkozitást az előre meghatározott Reynolds szám segítségével.

Utána a térbeli felbontást a resolution változóval határoztuk meg, mely a lid-driven cavity esetén azonos mindkét irányban (mivel a fizikai hosszok mindkét irányban azonosak). Egy 256-os felbontás azt jelenti ebben az esetben, hogy mind  $x$  és  $y$  irányban 256-256 csomópont, vagyis 255-255 cella van. A számítások (ütközés, terjedés) csomópontonként történnek, melyek a cellák sarkainak felelnek meg, mint más keretrendszerekénél is (pl. Palabos, OpenLB).

Ezt követően a **parameters** szótáron keresztül átadtuk a kinematikus viszkozitást, a 2 irány felbontását (a diszkrét csomópontok számát irányonként), és a  $Q$  diszkrét sebességek számát 9-nek határoztuk meg, mely az átlós szomszédok közti információ áramlást is lehetővé teszi 2D-ben, így csökkentve a sebesség diszkretizáció által okozott pontatlanságot.

Az ütközési szakasznál használt algoritmushoz a **ColliderBGK** osztály adtuk meg (nem példányosítjuk, csak átadjuk az osztályt), egy másik lehetőség a **ColliderTRT**, mely az  $f$  eloszlást szimmetrikus és asszimmetrikus részre bontja<sup>1</sup>, majd miután két különböző relaxációs időt alkalmazott rájuk összeadja őket, ezzel egyes esetekben növelve a stabilitást kisebb  $dt$  értékek esetén (ahol a BGK algoritmus elszállt).

A **tau\_dimensionless** dimenziótlan relaxációs ráta meghatározza a diszkrét  $dt$  idő nagyságát és közvetlenül befolyásolja a stabilitást. Nagyobb értéke egy nagyobb  $dt$  szimulációs időlépéshez vezet, viszont adott felbontás és kinematikus viszkozitás mellett akár instabilitást is okozhat (ekkor pedig csökkenteni kell).

A **BCDefault()** azt biztosítja jelen esetben, hogy a szilárd falaknál a teljes úti visszapatlanás (full-way bounceback), és a mozgó felső fedélnél pedig a teljes úti visszapatlanáson alapuló sebesség Dirichlet peremfeltételi algoritmus legyen alkalmazva.

A fizikai hosszakat a **lengths\_SI** listán keresztül adjuk át, melynek meghatározására azért van szükség, mert míg a szimuláció algoritmusai a  $\Delta x$  rácsra vonatkoztatott mértékegységgel dolgoznak, addig a peremértékeket SI mértékegységben adjuk meg, és a szimulációs eredmények is SI mértékegységben jelennek meg, melyeknél az átváltásokhoz szükség van a fizikai hosszakra.

A 16. sorban miután fölállítottuk a szimulációt a (minden csomópontnál azonos) globális beállításokkal, meg kell határoznunk a peremfeltételeinket. Ehhez szükséges van egy

---

<sup>1</sup> $f = f^+ + f^- = \frac{f+\bar{f}}{2} + \frac{f-\bar{f}}{2}$ , ahol  $\bar{f}$ -nél föl vannak cserélve az ellentétes irányok

`SetupIndexer` osztálybeli objektumra, melyet 2D esetben a származtatott `SetupIndexer2D` osztály segítségével hozhatunk létre. Az egyes falak kijelöléséhez a `setup_indexer` objektum már előre rendelkezik `WALL_LEFT`, stb... indexekkel, melyeket fölhasználunk hogy sűrűdásmentes falakat hozzunk létre az összes határon (`apply_solid`) kivéve a felsőt, melynél a `vel_top_SI` fedél mozgási sebesség szerinti Dirichlet sebesség peremfeltételt alkalmaztunk (`apply_dirichlet_velocity`).

A peremfeltételek felállítása után szükséges az `init()` metódus futtatása, különben kivételt dob a szimuláció amikor futtatni próbáljuk a `collide_stream()` metódust.

Amiután lefuttattuk 2 mp-ig a szimulációt, a `write_to_vtk(file_path)` metódussal tudjuk kiírni az eredményeket.

#### 4.1.2. Poiseuille Gravity 3D példakód

A használatához egy másik példa kód egy Poiseuille gravity 3D Áramlás megvalósítására (lásd: 5.1.2. alfejezet):

---

#### 4.2. kódrészlet. *Poiseuille gravity 3D*

---

```

1 D,Q = 3,19
2 lengths_SI = [length_SI, height_SI, height_SI]
3 resolution = [ round(resolution_vertical * length_SI / height_SI),
4               resolution_vertical, resolution_vertical]
5
6 dx = height_SI / (resolution[1] - 1)
7 tau_dimensionless = forced_tau_dimensionless
8 # [N/m^3]
9 gravity_SI = [gravity_SI, 0.0, 0.0]
10 lattice_border_types = [ BorderType.PERIODIC_BORDER, BorderType.NON_PERIODIC_BORDER
11                          , BorderType.NON_PERIODIC_BORDER ]
12
13 parameters = {
14     "kinematic_viscosity_SI": kinematic_viscosity_SI, # [m^2/s]
15     "dimensions": resolution,
16     "Q": Q,
17     "collider": ColliderBGK,
18     "unit_density_SI": density_SI,
19     "tau_dimensionless": tau_dimensionless,
20     "lengths_SI": lengths_SI if D == 3 else lengths_SI[:-1], # [m]
21     "boundary_conditions": BCDefault(),
22     "lattice_border_type_by_dim": lattice_border_types,
23     "forcing_scheme": ForcingSchemeGuo,
24     "lambda_TRT": 3/16,
25     "has_F": True
26 }
27
28 lbm = LBM(parameters)
29 F_gravity = AccelerationConstantGlobal(lbm.data, gravity_SI)
30 lbm.add_force(F_gravity)
31
32 setup_indexer = SetupIndexer3D(lbm)
33 setup_indexer.apply_solid(setup_indexer.WALL_TOP)
34 setup_indexer.apply_solid(setup_indexer.WALL_BOTTOM)
35 setup_indexer.apply_solid(setup_indexer.WALL_NEAR)

```

```

36 setup_indexer.apply_solid(setup_indexer.WALL_FAR)
37
38 dimensions = lbm.data.gp.dimensions
39 r_sqr = ((resolution_vertical-1)/2)**2
40 epsilon = 2.0
41 epsilon_sqr = epsilon ** 2
42
43 from tqdm import tqdm
44 j_middle, k_middle = (dimensions[1]-1)/2, (dimensions[2]-1)/2
45 @ti.kernel
46 def setup_pipe():
47     for j in range(dimensions[1]):
48         for k in range(dimensions[2]):
49             dist_sqr = (j-j_middle)**2 + (k-k_middle)**2 - r_sqr
50             if dist_sqr >= 0:
51                 for i in range(dimensions[0]):
52                     lbm.lp_init.cell_type[i,j,k] = CellType.SOLID
53             else:
54                 for i in range(dimensions[0]):
55                     lbm.lp_init.cell_type[i,j,k] = CellType.FLUID
56
57 setup_pipe()
58 lbm.init()
59
60 t = 2 #s
61 num_steps = t // lbm.data.gp.dt
62 for _ in range(num_steps):
63     lbm.collide_stream()
64
65 lbm.write_to_vtk('output/run.vti')

```

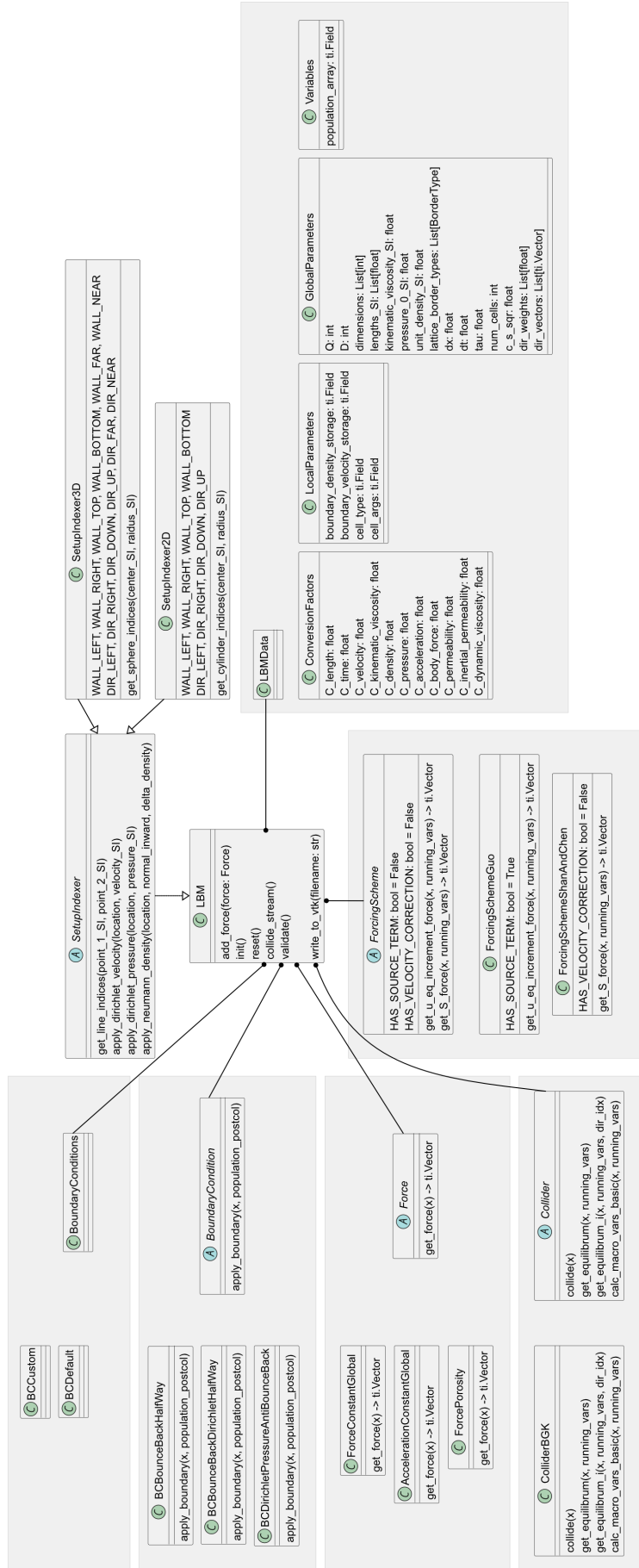
Itt már 3 dimenziós a szimuláció, és egy ennek megfelelő  $Q$  (diszkrét irányok száma) értéket kell választani. Egy gyakori választás a 19, vagyis a szimuláció a D3Q19 konfigurációban fog lefutni.

A felbontásnál itt már megjelenik az, hogy a hosszok arányának megfelelően kell alakulnia, vagyis hogy a diszkrét celláknál meg kell hogy egyezzen a hossza, a magassága, és a szélessége (3D esetben kockának kell lenniük).

Egy másik különbség a lid-driven cavity példakódhoz képest, hogy  $x$  irányban periódikus feltételt alkalmazunk a végetlen hosszú csőbeli áramlás megvalósításához (9. sor), valamint hogy a gravitációt erő formájában kell hozzáadni, ami a 28-29. sorokban történik meg.

Ezen kívül még ki kell „faragni” a cső geometriát, vagyis a 46. sorban kezdődő `setup_pipe()` taichi kernelen keresztül egy  $x$  tengelyű, a szimulációs keret peremeit súroló hengeren kívül eső pontokat szilárdnak állítjuk be.

## 4.2. A szoftver felépítése



4.1. ábra. Vázlatos Osztálydiagram.

### 4.2.1. Adatosztályok

#### LocalParameters

A különböző peremfeltételt megvalósító, valamint a szimulációban részt nem vevő térbeli pontok azonosítása érdekében minden csomóponthoz tartozik a típusát jelölő azonosító (`data.parameters.LocalParameters::cell_type`).

Az egyes típusokhoz, mint például a Dirichlet sebesség peremfeltételt meghatározó csomópontokhoz tartoznak mellékes információk (pl. fix sebesség egy bemenetnél).

Tekintettel arra, hogy nem minden csomópont peremfeltétel, és adott peremfeltételi értékek gyakran fordulhatnak elő többször, ezért ahelyett hogy minden csomóponthoz tartozzon peremfeltételi érték miatt allokált memória, inkább külön fix számú argumentum index tartozik minden csomóponthoz (`data.parameters.LocalParameters::cell_args`). Jelenleg 2 index van, és ez már 2D esetben is megtérül, mivel ha azt feltételezzük hogy legfeljebb 65536 egyedi érték lehet kategóriánként (Dirichlet sebesség, Dirichlet nyomás), akkor már 16 biten tudunk tárolni egy indexet, így 2 index egy 32 bites értékkel megegyező méretű.

#### Variables

A diszkrét csomópontonként tárolt, a futtatás során változó információk tárolásáért a `data.variables.Variables` osztály felelős, mely jelenleg a diszkrét  $f$  sűrűségfüggvény értékeket tartalmazó `population_array` és adott csomópontra értelmezett (és minden időlépés kezdetén újra kiszámolt) `local_F` tagokat tartalmazza.

Az, hogy milyen idővel változó információt tárolunk csomópontoként keretrendszerrel függő, pl. egyes keretrendszerek (pl. FluidX3D) csomópontoként tárolják a kiszámolt sebességet és sűrűséget a helyi erővel ellentétben. Ez azért van, mert külső erő megléte esetén a sebességet korrigálni kell, és itt döntenünk kell, hogy vagy a korrigált sebességet tároljuk, vagy magát a külső erőt. Az esetünkben az utóbbi lett implementálva végül.

#### GlobalParameters

A szimulációs környezet egészére vonatkozó, a futtatás közben bármikor olvasható (de nem írandó) információkat tartalmazza a `GlobalParameters`. Ilyen például  $D$  dimenziószám, a  $Q$  diszkrét mikroszkópikus sebességek száma,  $\tau$ ,  $\tau^*$  és annak reciproka mivel gyakran úgy fordul elő a számolandó egyenletekben<sup>2</sup>, valamint a  $\nu$  kinematikus viszkozitás mint rács (LBM) és SI mértékegységben.

#### ConversionFactors

A lebegőpontos számábrázolás adta kerekítési hibák miatt a szimulációs változókat ( $f$  populáció értékek, sebesség, sűrűség) nem SI, hanem rács mértékegységekben tároljuk annak érdekében hogy a lehető legkevesebb információ vesszen el a kerekítés következtében (a lebegőpontos számábrázolás diszkrét, véges jellege miatt).

---

<sup>2</sup>Ekkor nem kell folyamatosan számolni feleslegesen

Az adatok mentésekor viszont fontos, hogy értelmezhetőek legyenek az értékek, és ennek érdekében szükséges az SI mértékegységre váltás. Hogy ne kelljen többször leírni ugyanazt a mértékegységváltási kódot, ezért az átváltási faktorok egységesítve vannak a `ConversionFactors` osztályban, mely a  $C_a$  nevű mezőivel az a változó  $LBM > SI$  irányba történő váltást végzi szorzáskor, vagyis  $a[LBM] \cdot C_a = a[SI]$  (a visszafelé történő váltáshoz osztani kell).

#### 4.2.2. Logika

A logikát körülölelő kódstruktúra, és a periodikus peremfeltételt kivéve az algoritmusok a *The Lattice Boltzmann Method Principles and Practice* (Krüger et al.) [21] könyvből lettek átvéve.

#### Peremfeltételek

A peremfeltételek (a periódikus feltételt kivéve) a `src.boundary.BoundaryCondition` osztályból származnak, ahol felülírják az `apply_boundary()` absztrakt metódust. Az ütközési algoritmushoz és az összes adathoz a `collider` és `data` mezőn keresztül férnek hozzá.

A szimuláció szerint alkalmazandó peremfeltételek a csomópont típus (`CellType`) szerint vannak csoportosítva.

Ezt a szimuláció során vagy kézzel kell megadni (`src.boundary_conditions.BCCustom`), vagy egy másik lehetőség a `BCDefault` osztály példányosítása, ahol a full-way bounce-back, az azon alapuló Dirichlet feltétel, és a pressure anti-bounce-back [21] van meghatározva. Az egyes peremfeltételek alkalmazásához a `src.utils.setup_indexer.SetupIndexer2D` (vagy 3D amikor 3D-ben vagyunk) osztályt kell példányosítani, melynek külön elérhető `WALL_X`,  $X$  irányban lévő falat kijelölő mezői, valamint peremfeltételt alkalmazó metódusai (pl. `apply_solid`) vannak.

Kivételt képez a periódikus feltétel, ami nem osztályban, hanem egy külső függvényben van implementálva (`src.utils.streaming.try_get_streaming_index`). Videókártyák esetén drága (teljesítményt csökkentő) feltételes elágazások elkerülése végett a maradék operátor ki lett használva, ahol a rács dimenziók szerint osztunk. Ha adott irányban az  $x/y/z$  érték változott a maradék operátor elvégzése után, akkor átléptünk az adott határon, vagyis ha nem nulla a különbségük, akkor alapesetben nem engedjük, hogy a terjedési szakasz megtörténjen. Mivel periódikus  $x/y/z$  dimenziónál a `BorderType` 0 értéknek lett meghatározva, ezért a maradék operátor előtti és utáni értékek különbségét a határ típusokkal való szorzás lenullázza, vagyis ekkor mindegy ha átléptünk a rács másik oldalára<sup>3</sup>.

Ennek megfelelően 3 sorban implementálni lehet a periodikus feltételt:

#### 4.3. kódrészlet. Periodikus peremfeltétel implementálása

```

1 @ti.func
2 def try_get_streaming_index(global_params, x, c_i):
3     """for non-boundary nodes"""
4     x_next = x + c_i
5     x_next_mod = (x_next) % (global_params.dimensions)

```

<sup>3</sup>ha átlósan lépünk át több irány határán, akkor az összes átlépett határnak periódikusnak kell lennie

```

6     # need_break: next node crosses border at non-periodic edge, no streaming
7     need_break = ((x_next_mod - x_next) * global_params.lattice_border_types).any()
8
9     return x_next_mod, need_break

```

---

## Ütközési, terjedési szakasz

Az Ütközési, Terjedési szakasz felelős a csomópontok közti információ átadásáért és feldolgozásáért, és a felhasználó szempontjából időlépésenként az azokat megvalósító metódust (`LBM::collide_stream`) kell futtatni, hogy a szimuláció előrehaladjon.

Az ütközési szakasz az adott cellán belüli részecske populációkat az egyensúlyi állapot felé mozgatja a  $\Delta t$  diszkrét időlépés és  $\nu$  kinematikus viszkozitásnak megfelelő mértékben (modellezve a részecskék ütközését), míg a terjedési szakasz a részecskék vándorlását modellezi.

A teljesítmény szűk keresztmetszetét képi a terjedési szakasz algoritmusának megválasztása, melynél az irodalom gyakran használja a pull és push kifejezést. Szinte minden keretrendszerben, hogy a jobb teljesítmény érdekében egy csomópont iterációban hajtjuk végre egyszerre az ütközést és terjedést (`LBM::collide_stream`). Amennyiben nem így tennénk, akkor nagyjából kétszer több memória írásra lenne szükség, mivel az ütközés eredményét külön kellene tárolni (ahelyett hogy lokális változó legyen a végrehajtó egységnél).

A pull arra hivatkozik, hogy az ütközési szakasz után az ütközés utáni populációkat (diszkrét eloszlásokat) az ütközési helyi csomópontokra írjuk, majd a következő ütközés előtt a csomópont betölti a hozzá terjedő populációkat a szomszédos helyekről. Ezzel ellentétben a push esetén az ütközés után a memória írás közvetlen a szomszédos csomópontokra történik.

Amennyiben két populációs információt tartalmazó tömbünk van, akkor nem kell törődni azzal, hogy ne íródjon felül egy olyan populáció amelyet egy másik csomópont fog használni a jelenlegi időlépésben. Ekkor van egy `population_array` és egy `population_new` tömbünk, melyeket váltogatni tudunk páros és páratlan időlépés szerint.

Egy másik megközelítés, hogy csakis egy populációs tömbünk van, viszont ekkor csak oda írhatunk, ahova olvastunk. Amennyiben az időlépés páratlanságától függően egyszer pull-t, majd push-t alkalmazunk, akkor el lehet érni, hogy csakis egy populációs tömb használatával végrehajtsunk egy időlépést. Ilyen megközelítés például az esoteric pull (push), ami diszkrét iránytól függően pull-t vagy push-t alkalmaz, és ezt váltogatja az időlépés páratlanságától függetlenül.

Annak ellenére hogy implementáltuk az esoteric twist algoritmust, végül a két populációs tömböt használó pull algoritmusnál maradtunk. Ennek az az oka, hogy a Taichi esetén lassabban bizonyultak a push jellegű írások, és az esoteric twistnél gyorsabban teljesített a két tömbös, pointer cserélő módsze (30-50%-al, ami nem elvárta). Hogy ennek mi a pontos oka még nem tisztázott.

Ennek megfelelően az egyesített `collision_streaming` algoritmus:

### 4.4. kódrészlet. Ütközési algoritmus kódrészlet (`src.lbm._run_iteration_kernel`)

---

```

1  @ti.kernel
2  def _run_iteration_kernel(self):
3      ti.loop_config(block_dim=Config.block_dim_cell_loop)
4      for x in ti.grouped(self.data.vars.population_array):
5          running_vars = self.type_running_variables()
6
7          cell_type = self.data.lp.cell_type[x]
8          if cell_type == CellType.FLUID:
9              self.setup_running_vars(x, running_vars)
10             self.collider.collide(x, running_vars)
11         else:
12             self._apply_preproc_conditions_func(x, running_vars)
13
14         self.data.population_new[x] = running_vars.population_postcol

```

---

Ahol a `running_vars`-ba töltjük be egy pull séma alapján az érkező populációkat, és a `collide` algoritmus oda írja az ütközés eredményét (`running_vars.population_postcol`). A peremfeltételek a terjedésre szintén ide írják a terjedési szakasz közben közölt információjukat.

Ahelyett hogy feleslegesen visszaírjuk a `population_new` újonnanani populáció információkat a `vars.population_array` tömbbe, inkább cseréljük a két tömb referenciáját (azt, hogy a változók mire hivatkoznak). Mivel a Taichi nem támogatja sem a pointereket, sem a változók által mutatott referenciák cseréjét, ezért a `_run_iteration_kernel`-nek van egy redundáns, azonos kódú `_run_iteration_kernel_B` másolata, mely azután kompilálódik, hogy a referenciákat fölcseréltük.

## Külső erők

A külső erők megvalósításához a `forces.ForcingScheme` absztrakt osztályból leszármaztatva lehet implementálni a különböző algoritmusokat. Az absztrakt osztály annak megfelelően van kialakítva hogy támogassa mind az ütközéskor használt makroszkopikus sebesség korrigálása, valamint az ütközési algoritmus extra taggal való bővítés által az erő implementálását. Jelenleg a Guo [14] és ShanChen [39] alapú algoritmusok vannak implementálva. A Guo algoritmus implementációja látható az alábbi kódrészletnél:

### 4.5. kódrészlet. *Shan-Chen Erő Séma kódrészlet*

---

```

1  @ti.data_oriented
2  class ForcingSchemeGuo(ForcingScheme):
3      HAS_SOURCE_TERM: bool = True
4      # Guo 'u_eq' is the same as calculated 'u' calculated during postprocessing
5      HAS_VELOCITY_CORRECTION: bool = False
6
7      @ti.func
8      def get_u_eq_increment_force(self, x, running_vars: ti.template()):
9          # return ForcingSchemeGuo.get_u_eq_increment_accel(self, x, running_vars.F
10         / vars.population_array[x].sum())
11         pass
12
13     @ti.func
14     def get_u_eq_increment_accel(self, x, accel):
15         # return (0.5 * accel)
16         pass

```

---

```

16
17 @ti.func
18 def get_S_force(self, x, running_vars: ti.template()):
19     # The Lattice Boltzmann Method Principles and Practice
20     # Eq. (6.5)
21     u = running_vars.velocity
22
23     c_u = self.c @ u
24     c_F = self.c @ running_vars.F
25     # u_F = ti.Vector([0.0]* self.data.gp.Q)
26     u_F = dot(u, running_vars.F)
27     # print(u, u - u_F)
28     S = (
29         (1.0 - 0.5 * self.inv_tau_dimensionless)
30         * self.w
31         * self.inv_c_s_sqr
32         * (c_F - u_F + self.inv_c_s_sqr * c_u * c_F)
33     )
34     return S
35
36 @ti.func
37 def get_S_accel(self, x, accel):
38     rho = self.data.vars.population_array[x].sum()
39     return self.get_S_force(x, accel) * rho

```

Egy erő osztály implementálásánál még egy lehetőség az, hogy külön Field-eket definiálhatunk, melynek lefoglalódik a memóriája amennyiben használjuk őket, vagyis olyan többletinformációkat igényelhetnek, mint pl. a permeabilitás egy porozitást megvalósító algoritmusnál. A használatát mutatja a ForcePorosity osztály, mely a Forchheimer taggal bővített Darcy törvényt írja le:

#### 4.6. kódrészlet. *ForcePorosity osztály, Forchheimer taggal bővített Darcy törvényt megvalósító kódrészlet*

```

1 class ForcePorosity(Force):
2     # (Field Name, Field Dimension ('D'|1| etc..), Field DType, (Default Value)
3     REQUIRED_LOCAL_FIELDS: List[Tuple] = [
4         {
5             "name": "permeability",
6             "dimension": 1,
7             "dtype": "float",
8             "default_value": float("inf"),
9         },
10        {
11            "name": "inertial_permeability",
12            "dimension": 1,
13            "dtype": "float",
14            "default_value": float("inf"),
15        },
16    ]
17
18 @ti.func
19 def get_force(self, x):
20     rho = self.data.vars.population_array[x].sum()
21     mu = rho * self.data.gp.kinematic_viscosity_LBM
22     k, k_1 = self.data.lp.permeability[x], self.data.lp.inertial_permeability[x]
23
24     q = (
25         self.data.vars.population_array[x] @ self.data.gp.dir_vectors
26     ) / rho + 0.5 * self.data.vars.local_F[x]

```

Látszik a Field definiálás szintaxisa, valamint hogy azzal, hogy az alapértelmezett a permeabilitás és inerciális permeabilitás értéke végtelen, ezért nincs hatása az algoritmusnak a nem-porózus cellákra.

### 4.3. Algoritmikus helyesség ellenőrzése

A kód áramlástan helyességét ellenőrzi a különböző áramlástan esetek lefuttatása, és azok eredményeinek összehasonlítása egy referenciához, mely lehet analitikus megoldás, vagy más módon kiszámolt eredmény.

Ezek regressziós tesztként is szolgálnak, vagyis amikor változik, bővül a kód, a valamennyi tesztelés validálása biztosítja hogy ezeket követően is megfelelő áramlástan viselkedést kapunk.

Az algoritmikus helyesség eddig assert kijelentésekkel volt megtámogatva, vagyis amennyiben a Taichi keretrendszer debugging módban volt elindítva, a futás közben ezek a kijelentések mindig kiértékelődnek, és amennyiben az értékük hamis az adott időlépésben, akkor a program leállt, jelezve azt, hogy nem teljesül a kiszabott feltétel.

Ilyen például, hogy makroszkópikus változók kiértékelésénél nem lehet sem negatív sem NaN<sup>4</sup> a sűrűség, a sebesség, valamint hogy adott áramlástan esetek validálásánál a (rács-mértékegységtől SI-be, valamint fordítva történő) mértékegység átszámításnál a különböző módokon történő átváltások azonos eredményt adnak e.

Ezek segítségnek bizonyultak a fejlesztés korai fázisában, valamint a 3D LBM, a TRT Ütközési Operátor, és a különböző terjedési algoritmusok (AA, Esoteric Pull/Push) implementációjakor.

---

<sup>4</sup>Not a Number, vagyis Nem szám, ami az LBM elszállását, a stabilitás megsértését jelenti

## 5. fejezet

# Eredmények

### 5.1. Áramlástanai esetek validálása

#### 5.1.1. Lid-driven cavity

Az egyik teszt eset (áramlástanai eredményt ellenőrző kód) a lid-driven cavity probléma, melynél 2D esetben egy  $1 \times 1$   $m^2$ -es dimenziójú négyzet felső oldala mozog  $x$  irányban  $1$  nagyságú  $m/s$  sebességgel ( $v = (1, 0) [m/s]$ ), valamint 3D esetben egy  $1 \times 1 \times 1$   $m^3$ -es dimenziójú kocka felső oldal mozog átlósan (45 fokot bezárva)  $x, z$  irányban  $1$   $m/s$  nagyságú sebességgel ( $v = (\sqrt{\frac{1}{2}}, 0, \sqrt{\frac{1}{2}}) [m/s]$ ).

Mindkettő implementálva van a keretrendszerben, viszont csak az előzőre van teszt eset, melyben előre kiszámolt referencia adatokhoz hasonlítunk, melyek a középső vízszintes sávban lévő sebesség  $y$  komponensek, valamint a középső függőleges sávban lévő sebesség  $x$  komponenseket adja összehasonlításra [10, 2].

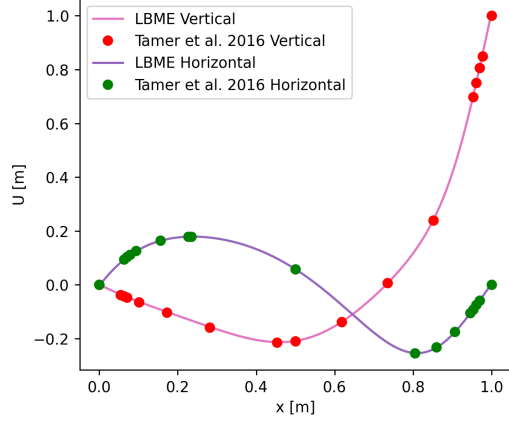
A teszt feltétele annak lett meghatározva, hogy külön a vízszintes és függőleges sávokban teljesülnie kell annak, hogy az adott sávban a százalékos sebesség különbség átlaga kisebb, mint 5%.

A százalékos különbség így lett meghatározva az  $i$ -edik diszkrét pontra a sávban:

$$\text{diff}_i = \frac{|v_i - v_{i_{ref}}|}{\min(|v_i|, |v_{i_{ref}}|)} \quad (5.1)$$

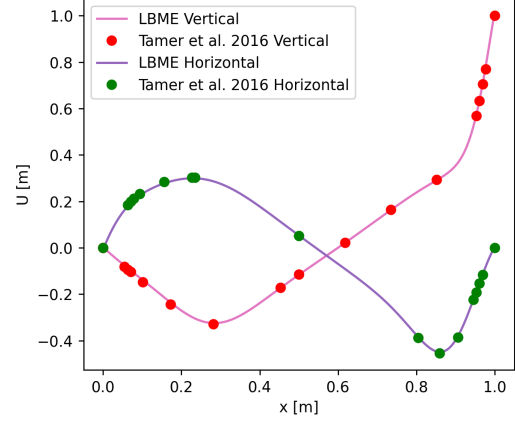
ahol a sebesség  $x$  komponensét vesszük a függőleges, és  $y$  komponensét a vízszintes sávban.

Lid-driven cavity flow, Re: 100,  $\tau^*$ : 0.800, Resolution: 256



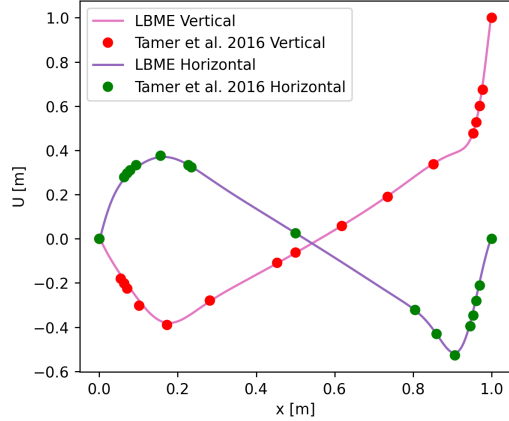
(a) Átlagos eltérés 128 ezer időlépés után: vízszintes 0,949%, függőleges 2,374%

Lid-driven cavity flow, Re: 400,  $\tau^*$ : 0.800, Resolution: 256



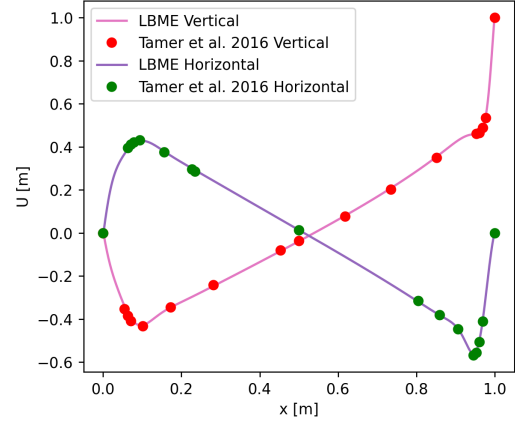
(b) Átlagos eltérés 128 ezer időlépés után: vízszintes 1,703%, függőleges 2,498%

Lid-driven cavity flow, Re: 1000,  $\tau^*$ : 0.800, Resolution: 256



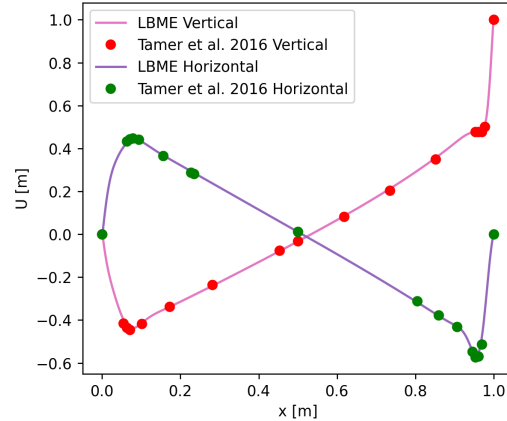
(c) Átlagos eltérés 248 ezer időlépés után: vízszintes 2,525%, függőleges 2,997%

Lid-driven cavity flow, Re: 3200,  $\tau^*$ : 0.700, Resolution: 512



(d) Átlagos eltérés 400 ezer időlépés után: vízszintes 1,162%, függőleges 1,747%

Lid-driven cavity flow, Re: 5000,  $\tau^*$ : 0.650, Resolution: 512



(e) Átlagos eltérés 400 ezer időlépés után: vízszintes 1,685%, függőleges 1,833%

5.1. ábra. Lid-Driven Cavity eredmények különböző Reynolds számok esetén

### 5.1.2. Poiseuille Gravity 2D

A Poiseuille áramlásnál 2D Esetben két végtelen hosszú párhuzamos síklemez közti, és 3D esetben egy végtelen hosszú egyenes csőbeli lamináris áramlást feltételezünk egy fix nyomás gradiens mellett úgy, hogy a felületeknél sűrűlódásmentesség áll fenn.

Ennek az eredménye az, hogy a cső (síklemezek közti tér) közepétől való távolság függvényében a felületekkel párhuzamos (az esetünkben  $y$  irányú) sebesség egy parabólikus profilt vesz fel.

A nyomás gradienst egy  $y$  irányú  $g \left[ \frac{m}{s^2} \right]$  gravitációs gyorsulás alkalmazásával valósítjuk meg, és az  $y$  irányban periodikusan ismétlődő áramlási mintát kihasználva periódikus feltételt alkalmazunk.

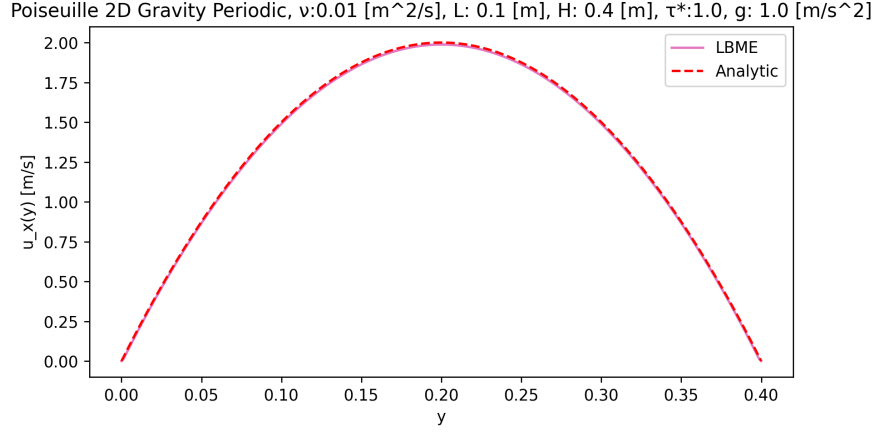
Ennek megfelelően az analitikus megoldás 2D lamináris esetben:

$$v_{2D}(x, x_{center}, \nu, g, H) = \frac{g}{2\nu} \left( \frac{H^2}{4} - (x - x_{center})^2 \right) \quad (5.2)$$

Ahol  $x[m]$  a jelenlegi távolság az alsó síklemeztől,  $x_{center}[m]$  a síklemezek közti tér közepének koordinátája,  $\nu[m^2/s]$  a kinematikus viszkozitás, és  $H[m]$  a síklemezek közti távolság.

Hasonlóan 3D esetben, ahol síklemezek közti távolság helyett belső cső átmérő van, és a jelenlegi  $x$  távolságot a cső szélektől mérjük:

$$v_{3D}(x, x_{center}, \nu, g, H) = \frac{g}{4\nu} \left( \frac{H^2}{4} - (x - x_{center})^2 \right) \quad (5.3)$$



**5.2. ábra.** *Poiseuille Gravity 2D Validálási Eredmények (alsó cső szélétől vett távolság szerinti sebesség  $x$  komponens). Az átlagos eltérés 0,843% volt*

A 5.2 ábrán leírt paraméterek mellett az analitikus megoldáshoz képest az átlagos eltérés 0,848% volt (Számolás módjáért lásd az 5.1 egyenletet).

Az  $x$  irányban periódikus ismétlődést alkalmaztunk, viszont a számítási hossz  $L$  0,1 méternek volt vége. A síklapok közti távolság 0,4 méter volt, a lamináris áramlást  $0,01 [m^2/s]$  kinematikus viszkozitás mellett biztosítottuk, valamint a nehézségi gyorsulás  $1,0[m/s^2]$ -nek volt vége.

A függőleges felbontás 315, a vízszintes felbontás 79 volt.

Hogy stabilan, de mégis gyorsan (nagy  $\Delta t$  időlépésekkel) fusson le a szimuláció, a  $\tau^*$  dimenziótlan relaxációs időt 1,0-nak vettük. A szimuláció összesen 200 ezer időlépésig futott.

### 5.1.3. Porózus zónájú csőbeli áramlás

Porozitás alatt az adott szilárd test területén belüli üres, vagyis nem szilárd tér hányadát értjük (pl. szivacos csontállományban a csontállomány nélküli tér hányada).

Makroszkópikus viselkedését egy áramláson belül modellezhetjük a Darcy-törvénnyel, mely szerint a  $q$  felületi sebességgel arányos nyomásgradiens figyelhető meg a  $\mu$  dinamikus viszkozitás és a  $k$  porózus anyagot jellemző permeabilitás függvényében. Izotróp esetben  $k$  egy skalár, míg anizotróp esetben egy mátrix (másodrendű tenzor) [19]:

$$\nabla p = -\frac{\mu}{k} \vec{q}, \text{ anizotróp} \quad (5.4)$$

$$\vec{q} = -\frac{1}{\mu} \underline{k} \nabla p \text{ izotróp} \quad (5.5)$$

Biológiai szöveteknél a tehetetlenségi hatásokat is modellezni szokták úgy, hogy egy úgynevezett Forchheimer taggal bővítik a Darcy egyenletet, mely a felületi sebesség négyzetével arányos [11]:

$$\nabla p = -\frac{\mu}{k} \vec{q} - \frac{\rho}{k_1} |\vec{q}| \vec{q} \quad (5.6)$$

Ahol  $k_1$  az úgynevezett inerciális permeabilitás.

A validálási eset 2D-ben azonos a Poiseuille Gravity 2D-vel, viszont az ismétlődő minta közepétől kezdve pozitív  $x$  irányban adott hosszú egy teljes magasságú porózus zóna található adott  $k$  permeabilitással,  $k_1$  inerciális permeabilitással, valamint 3D-ben ugyanez, csak síklemez helyett egy csőbeli lumenről van szó.

A referencia a módosított (izotróp, Forchheimer tagos) Darcy-törvény szerint kiszámolt, porózus zónán belül számolt nyomásesések összege, és ezt hasonlítjuk össze a porózus zóna kezdeti és vége között értelmezett nyomáskülönbséggel, és legfeljebb egy 5%-os különbséget engedünk.

A validálási eset 2D-ben az, hogy 2,  $x$  irányban  $L$  hosszú párhuzamos síklemezünk van, melyek közt  $H$  távolság van, valamint az  $x=0$ -nál a bemeneten adott sebességű a folyadékunk. A síklemezek felülete sűrűlódásmentes. Az  $x=L/2$ -től kezdve pozitív  $x$  irányban adott hosszú egy teljes magasságú porózus zóna van adott permeabilitással, inerciális permeabilitással, valamint 3D-ben ugyanez, csak síklemez helyett egy csőbeli lumenről van szó.

## Porous Zone 2D

Az  $x$  irányban periódikus ismétlődést alkalmaztunk, viszont a számítási hossz  $L$  6 milliméternek volt véve. A síklapok közti távolság 1 milliméter volt, a  $\nu$  kinematikus viszkozitást  $1,0 \cdot 10^{-6} [m^2/s]$ -ra állítottuk, valamint a nehézségi gyorsulás  $10,0 [m/s^2]$ -nek volt véve.

A függőleges felbontás 200, a vízszintes felbontás 1200 volt.

Hogy stabilan, de mégis gyorsan (nagy  $\Delta t$  időlépésekkel) fusson le a szimuláció, a  $\tau^*$  dimenziótlan relaxációs időt 0,8-nak vettük. A szimuláció összesen 200 ezer időlépésig futott.

A porózus zóna a síkszelet közepétől véve kezdődött és 0,5 milliméter hosszú tartott, és a  $k$  permeabilitás  $100 [\Delta x^2]$ , a  $k_1$  inerciális permeabilitás pedig  $100 [\Delta x]$  volt.

Az elvárt nyomásesés úgy lett kiszámolva, hogy össze lettek adva a porózus zónán a módosított Darcy-törvénnyel kapott nyomáskülönbségek.

Ennek megfelelően a nyomásesés különbség 2D esetben 3,324% volt.

A szimuláció szemléltető vizualizációja az 5.3. ábrán szerepel.

## Porous Zone 3D

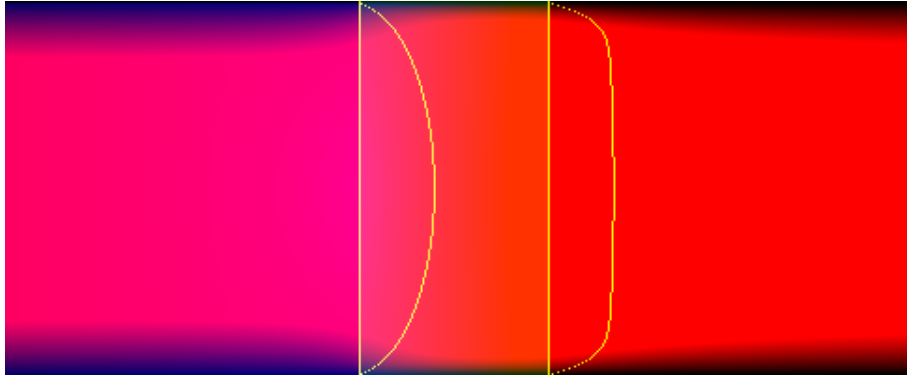
Az  $x$  irányban periódikus ismétlődést alkalmaztunk, viszont a számítási hossz  $L$  5 milliméternek volt véve. A belső cső átmérő 1 milliméter volt, a  $\nu$  kinematikus viszkozitást  $1,0 \cdot 10^{-6} [m^2/s]$ -ra állítottuk, valamint a nehézségi gyorsulás  $10,0 [m/s^2]$ -nek volt véve.

A függőleges felbontás 99, a vízszintes felbontás 495 volt.

Hogy stabilan, de mégis gyorsan (nagy  $\Delta t$  időlépésekkel) fusson le a szimuláció, a  $\tau^*$  dimenziótlan relaxációs időt 0,53-nak vettük. A szimuláció összesen 10 ezer időlépésig futott.

A porózus zóna a csőszelet közepétől véve kezdődött és 0,25 milliméter hosszú tartott, és a  $k$  permeabilitás  $50 [\Delta x^2]$ , a  $k_1$  inerciális permeabilitás pedig  $20 [\Delta x]$  volt.

Az elvárt nyomásesés úgy lett kiszámolva, hogy csomópontoként össze lettek adva a porózus zónában a módosított Darcy-törvénnyel számolt nyomáskülönbségek.



**5.3. ábra.** *Porous Zone 2D Konfiguráció vizualizálása. A középén lévő zöld színű zóna a porózus zóna, valamint a kék szín emelkedett nyomást jelez. A vörös szín a sebesség nagyságával arányos. A sárga görbék az adott függőleges egyenesen vett sebesség profilt jelzik, és mutatják a porózus zóna bemenete és kimenete közti különbségét.*

Ennek megfelelően a nyomásesés különbség 2D esetben 1,930% volt.

## 5.2. Teljesítmény összehasonlítás OpenLB-vel

Az OpenLB egy objektum-orientált keretrendszere az LBM-nek, és mind processzoron és CUDA-t támogató videokártyán futtatható.

Mivel gyakori fejlesztés alatt van és jelenleg az egyik leghíresebb LBM keretrendszer (olyannyira, hogy évente tavasszal fizetős workshopokat rendeznek „Spring School” néven), ezért célszerű a saját keretrendszerünket hozzá hasonlítani, hogy valós képet kapjunk a Taichi nyelv teljesítményéről, használhatóságáról.

A teljesítmény fölmérésére a *Lid-Driven Cavity 3D*-t fogjuk lefuttatni  $128^3$  és  $256^3$  felbontáson. A metrika amit használni fogunk az úgynevezett MLUPS, vagyis Mega Lattice Updates Per Second, ami megadja hogy 1 másodperc alatt átlagosan hány cella frissítődik futáskor.

A futás előtt mind a keretrendszerünk és az OpenLB esetén 10 időlépés fut le előzetesen úgy, hogy az MLUPS számításba nem számít (Ez a benchmarking gyakorlat mind az OpenLB, és az ahhoz hasonló Palabos keretrendszer esetén megfigyelhető).

A processzorok esetén lejegyezzük a használt memória típusát és frekvenciáját, mivel az algoritmus legfőbbképpen memória sávszélesség limitált.

| Lid Driven 3D Cavity Benchmark       |               |                  |               |
|--------------------------------------|---------------|------------------|---------------|
| Eszköz                               | Keretrendszer | Felbontás        | Teljesítmény  |
| Intel 4690k 4,5Ghz DDR3 1866Mhz RAM  | Saját         | 128 <sup>3</sup> | 31,9 MLUPS    |
| Intel 4690k 4,5Ghz DDR3 1866Mhz RAM  | Palabos 2.3.0 | 128 <sup>3</sup> | 28,33 MLUPS   |
| Intel 4690k 4,5Ghz DDR3 1866Mhz RAM  | OpenLB 1.7    | 128 <sup>3</sup> | 57,72 MLUPS   |
| Intel 4700MQ 3,4Ghz DDR3 1866Mhz RAM | Saját         | 128 <sup>3</sup> | 19,95 MLUPS   |
| Intel 4700MQ 3,4Ghz DDR3 1866Mhz RAM | Palabos 2.3.0 | 128 <sup>3</sup> | 22,53 MLUPS   |
| Intel 4700MQ 3,4Ghz DDR3 1866Mhz RAM | OpenLB 1.7    | 128 <sup>3</sup> | 38,05 MLUPS   |
| Nvidia RTX A5000                     | Saját         | 128 <sup>3</sup> | 1179,96 MLUPS |
| Nvidia RTX A5000                     | OpenLB 1.7    | 128 <sup>3</sup> | 2145,97 MLUPS |
| Nvidia RTX A5000                     | Saját         | 256 <sup>3</sup> | 1376,2 MLUPS  |
| Nvidia RTX A5000                     | OpenLB 1.7    | 256 <sup>3</sup> | 2845,7 MLUPS  |
| Timi 64 Mag                          | Saját         | 128 <sup>3</sup> | 139,23 MLUPS  |
| Timi 64 Mag                          | OpenLB 1.7    | 128 <sup>3</sup> | 827,53 MLUPS  |
| Timi 64 Mag                          | Saját         | 256 <sup>3</sup> | 259,39 MLUPS  |
| Timi 64 Mag                          | OpenLB 1.7    | 256 <sup>3</sup> | 908,05 MLUPS  |

Érdemes megjegyezni, hogy a Taichi jelenleg nem támogatja a vektorprocesszor utasításokat CPU-n, vagyis az olyan műveletek, mint a skalárszorzat, mátrix szorzás jelentősen lassabb lesz. Ennek korrigálásának egy lehetősége külső C könyvtár importálása, mint például az OpenBLAS, és annak használata amennyiben nem videokártyán futtatunk. Az ettől várható teljesítmény javulás jelentősebbnek bizonyul 3D esetben mivel ekkor a diszkrét mikroszkópikus sebességek száma jelentősen megnő, és ekkor a sebesség számításakor a diszkrét irányokat tartozó mátrix amivel szorzunk nem 9x2 (D2Q9), hanem pl. 19x3 (D3Q19), 3x27-re (D3Q27) nő.

## 6. fejezet

# Összefoglaló

Az áramlások numerikus modellezésére használt módszerek közül az újabban népszerűségnek örvendő és fejlődő Lattice Boltzmann Módszer tulajdonságai közé tartozik a jelentős mértékű párhuzamosítás, mely kedvez a videokártyákon történő általános célú számítású (GPGPU) futtatásnak, melyeknél még mindig érvényes az idő elteltével az exponenciális teljesítmény növekedése a megjelenő termékeknek.

Ez egy lehetőséget nyújt a hemodinamikai kutatások részét képző szimulációk futtatásához, az azokhoz szükséges algoritmusok megvalósításához, viszont mivel a jelenlegi keretrendszerek C++ nyelven íródnak, ezért a maximális teljesítmény eléréshez a C++ nyújtotta Template-k széles körű használata szükséges, ami viszont nehezíti az olvashatóságot. Az erre épülő videokártyán futó kód (CUDA, OpenCL) megírásához szükség van jelentős GPGPU programozási ismeretre, ezért gyakran a szimuláció felállításának, és az ahhoz kigondolt algoritmusok megírásának idejét növeli.

A Python szintaxisát használó Taichi nyelv lehetővé teszi, hogy magas teljesítményű kódot írassunk mely akár processzoron vagy videokártyán fut, így lehetővé téve a rapid prototyping-et akár hemodinamikai szimulációt esetén.

A dolgozat annak a kérdését járta körül, hogy lehetséges-e a Taichi-t használó keretrendszer létrehozása, mely a nyilvánosan elérhető keretrendszerekhez hasonló teljesítményt nyújt.

Az eredmény egy relatíve kevés (3116) sor alatt megvalósított, teljesítményében processzoron futtatáskor a Palaboshoz hasonló teljesítményt nyújtó, viszont videokártyás futtatás esetén OpenLB-hez képest elmaradt teljesítményű, de könnyen bővíthető Lattice Boltzmann Módszeren alapuló áramlástani keretrendszer.

Továbbfejlesztési lehetőségek közé tartozik a vektorprocesszor utasításkészlet implementálása CPU-n külső könyvtáron keresztül a jobb teljesítmény érdekében, a terjedési szakasz moduláris kicserélhetősége az összehasonlítás és teljesítmény elemzése érdekében, egy mesh voxelizáció algoritmus implementálása vagy integrálása hogy könnyen be lehessen tölteni különböző ér geometriákat, valamint az interpolált visszapattnás és immersed boundary módszer (IBM) algoritmusok implementálása hogy komplex geometriákat lehessen használni javuló pontossággal alacsonyabb felbontásokon.

# Irodalomjegyzék

- [1] Medmovie pipeline embolisation device figures.
- [2] Tamer A AbdelMigid, Khalid M Saqr, Mohamed A Kotb, and Ahmed A Aboelfarag. Revisiting the lid-driven cavity flow problem: Review and new steady state benchmarking results using gpu accelerated code. *Alexandria engineering journal*, 56(1):123–135, 2017.
- [3] Michael K Abraham and Wan-Tsu Wendy Chang. Subarachnoid hemorrhage. *Emergency Medicine Clinics*, 34(4):901–916, 2016.
- [4] A Stijntje E Bor, Gabriel JE Rinkel, Jeroen van Norden, and Marieke JH Wermer. Long-term, serial screening for intracranial aneurysms in individuals with a family history of aneurysmal subarachnoid haemorrhage: a cohort study. *The Lancet Neurology*, 13(4):385–392, 2014.
- [5] Thorsten M Buzug. Computed tomography. In *Springer handbook of medical technology*, pages 311–342. Springer, 2011.
- [6] Franklyn Colón-Arias, Melanie Moronta-Franco, and Anthony Gutiérrez-Martínez. Coronary stent embolization during percutaneous coronary intervention: Successful retrieval using twisting guide wire technique by radial approach. *Journal of Cardiology Cases*, 23(6):271–273, 2021.
- [7] Zhi-Qiang Dong, Lian-Ping Wang, Cheng Peng, and Tao Chen. A systematic study of hidden errors in the bounce-back scheme and their various effects in the lattice boltzmann simulation of viscous flows. *Physics of Fluids*, 34(9), 2022.
- [8] Richard Fabian. Data-oriented design. *framework*, 21:1–7, 2018.
- [9] Martin Geier, Andrea Pasquali, and Martin Schönherr. Parametrization of the cumulant lattice boltzmann method for fourth order accurate diffusion part i: Derivation and validation. *Journal of Computational Physics*, 348:862–888, 2017.
- [10] U Ghia, K.N Ghia, and C.T Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387–411, 1982.

- [11] Alfio Grillo, Melania Carfagnay, and Salvatore Federico. The darcy-forchheimer law for modelling fluid flow in biological tissues. *Theoretical and Applied Mechanics*, 41(4):283–322, 2014.
- [12] Bradley A Gross and Kai U Frerichs. Stent usage in the treatment of intracranial aneurysms: past, present and future. *Journal of Neurology, Neurosurgery & Psychiatry*, 84(3):244–253, 2013.
- [13] Zhaoli Guo and Chang Shu. *Lattice Boltzmann method and its application in engineering*, volume 3. World Scientific, 2013.
- [14] Zhaoli Guo and TS Zhao. Lattice boltzmann model for incompressible flows through porous media. *Physical review E*, 66(3):036304, 2002.
- [15] Yazeed W Haddad, Ejona Korcari, Gina Natasha Polsinelli, and Ding Yuchuan. The effect of cocaine and methamphetamine on saccular aneurysm formation and rupture: A literature review. *Brain Hemorrhages*, 2(3):111–115, 2021.
- [16] O Heiskanen, A Poranen, T Kuurne, S Valtonen, and M Kaste. Acute surgery for intracerebral haematomas caused by rupture of an intracranial arterial aneurysm: a prospective randomized study. *Acta neurochirurgica*, 90:81–83, 1988.
- [17] Virginia J Hogg, Xiu Rong Dong, and Mark W Majesky. Neointima formation: a local affair, 2010.
- [18] Haibo Huang, Michael Sukop, and Xiyun Lu. Multiphase lattice boltzmann methods: Theory and application. 2015.
- [19] M King Hubbert. Darcy’s law and the field equations of the flow of underground fluids. *Transactions of the AIME*, 207(01):222–239, 1956.
- [20] Brianna M Krafcik, David H Stone, Ming Cai, Isabel A Jarmel, Mark Eid, Philip P Goodney, Jesse A Columbo, and Michael F Mayo Smith. Changes in global mortality from aortic aneurysm. *Journal of Vascular Surgery*, 2024.
- [21] Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. The lattice boltzmann method. *Springer International Publishing*, 10(978-3):4–15, 2017.
- [22] Roger M Krzyżewski, Kornelia M Kliś, Renata Kucala, Jarosław Polak, Borys M Kwinta, Anna Starowicz-Filip, Krzysztof Stachura, Karolina Piszczek, Marek Moskała, and Krzysztof A Tomaszewski. Intracranial aneurysm distribution and characteristics according to gender. *British journal of neurosurgery*, 32(5):541–543, 2018.
- [23] Noel Llopis and Snappy Touch. High-performance programming with data-oriented design. *Game Engine Gems*, 2:251–261, 2011.
- [24] Chien-Ping Lu. Ai, native supercomputing and the revival of moore’s law. *APSIPA Transactions on Signal and Information Processing*, 6:e9, 2017.

- [25] Shunsuke Masuda, Soichiro Fujimura, Hiroyuki Takao, Kohei Takeshita, Takashi Suzuki, Yuya Uchiyama, Kostadin Karagiozov, Toshihiro Ishibashi, Koji Fukudome, Makoto Yamamoto, et al. Effects of different stent wire mesh densities on hemodynamics in aneurysms of different sizes. *Plos one*, 17(6):e0269675, 2022.
- [26] Keijo K Mattila, Paulo C Philippi, and Luiz A Hegele. High-order regularization in lattice-boltzmann equations. *Physics of Fluids*, 29(4), 2017.
- [27] Daniel T. Thorne Michael C. Sukop. *Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers*. Springer, 2006.
- [28] Romain Noël, Fei Ge, Yue Zhang, Laurent Navarro, and Guy Courbebaisse. Lattice boltzmann method for modelling of biological phenomena. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2654–2658. IEEE, 2017.
- [29] Julio C Palmaz. Intravascular stents: tissue-stent interactions and design considerations. *AJR. American journal of roentgenology*, 160(3):613–618, 1993.
- [30] Jaechan Park, Hyunjin Woo, Dong-Hun Kang, Yong-Sun Kim, Min Young Kim, Sang Gyu Kwak, et al. Formal protocol for emergency treatment of ruptured intracranial aneurysms to reduce in-hospital rebleeding and improve clinical outcomes. *Journal of neurosurgery*, 122(2):383–391, 2015.
- [31] DV Patil. Chapman–enskog analysis for finite-volume formulation of lattice boltzmann equation. *Physica A: Statistical Mechanics and its Applications*, 392(12):2701–2712, 2013.
- [32] Laurent Pierot. Flow diverter stents in the treatment of intracranial aneurysms: Where are we? *Journal of neuroradiology*, 38(1):40–46, 2011.
- [33] Laurent Pierot and Ajay K Wakhloo. Endovascular treatment of intracranial aneurysms: current status. *Stroke*, 44(7):2046–2054, 2013.
- [34] S Posey, J Luitjens, O Hennigh, and S Oberlin. Gpu-based hpc and ai developments for cfd. In *Eleventh International Conference on Computational Fluid Dynamics*, 2022.
- [35] Stuart H Ralston, Ian D Penman, Mark WJ Strachan, and Richard Hobson. *Davidson’s Principles and Practice of Medicine: Davidson’s Principles and Practice of Medicine E-Book*. Elsevier Health Sciences, 2018.
- [36] Hans J Rupprecht, Harald Darius, Ulrike Borkowski, Thomas Voigtlander, Bernd Nowak, Sabine Genth, and Jurgen Meyer. Comparison of antiplatelet effects of aspirin, ticlopidine, or their combination after stent implantation. *Circulation*, 97(11):1046–1052, 1998.
- [37] Natzi Sakalihasan, Jean-Olivier Defraigne, Marie-Ange Kerstenne, Jean-Paul Cheramy-Bien, Diane T Smelser, Gerard Tromp, and Helena Kuivaniemi. Family

- members of patients with abdominal aortic aneurysms are at increased risk for aneurysms: analysis of 618 probands and their families from the liege aaa family study. *Annals of vascular surgery*, 28(4):787–797, 2014.
- [38] Mohamed Salhab, Jessica Farmer, and Isam Osman. Impact of delay on survival in patients with ruptured abdominal aortic aneurysm. *Vascular*, 14(1):38–42, 2006.
  - [39] Xiaowen Shan and Hudong Chen. Lattice boltzmann model for simulating flows with multiple phases and components. *Phys. Rev. E*, 47:1815–1819, Mar 1993.
  - [40] Carmelo L Sturiale, Waleed Brinjikji, Mohammad H Murad, and Giuseppe Lanzino. Endovascular treatment of intracranial aneurysms in elderly patients: a systematic review and meta-analysis. *Stroke*, 44(7):1897–1902, 2013.
  - [41] Sauro Succi and S Succi. *The lattice Boltzmann equation: for complex states of flowing matter*. Oxford university press, 2018.
  - [42] Inc. Taichi Graphics Technology. Fields (advanced) taichi docs.
  - [43] Inc. Taichi Graphics Technology. Taichi lang.
  - [44] Isabel Wanke and Michael Forsting. Stents for intracranial wide-necked aneurysms: more than mechanical protection. *Neuroradiology*, 50:991–998, 2008.
  - [45] Catherine Westbrook and John Talbot. *MRI in Practice*. John Wiley & Sons, 2018.
  - [46] Michael H Wholey, Mark Wholey, Patrice Bergeron, Edward B Diethrich, Michel Henry, Jean C Laborde, Klaus Mathias, Subbarao Myla, Gary S Roubin, Fayaz Shawl, et al. Current global status of carotid artery stent placement. *Catheterization and cardiovascular diagnosis*, 44(1):1–6, 1998.
  - [47] Xuewen Yin and Junfeng Zhang. An improved bounce-back scheme for complex boundary conditions in lattice boltzmann method. *Journal of Computational Physics*, 231(11):4295–4303, 2012.
  - [48] Junjie Zhao, Hao Lin, Richard Summers, Mingmin Yang, Brian G Cousins, and Janice Tsui. Current treatment strategies for intracranial aneurysms: an overview. *Angiology*, 69(1):17–30, 2018.