M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

AFM-es felületi töltéssűrűség mérésének szimulációja

TDK DOLGOZAT

Készítette
Bakró Nagy István

Konzulens
Reichardt András

2014. október 20.

Tartalomjegyzék

Kivonat

Atomerő mikroszkóppal való fémezett felületű minta felületi töltéssûrûségének mérése során kritikus a tû és a minta közötti kapacitás ismerete. A kapacitás értékét közelítések mellett lehetséges analitikusan kifejezni. Numerikus szimulációval pontosabban ismerhetjük az értékét, ezáltal a felbontás nõ és a mérési zaj csökken.

A szimuláció során a lehetséges párhuzamosításokat felhasználva a számítási idõt elfogadhatóra csökkentettük, ami akár online feldolgozást is lehetõvé teszi.

Atomerő mikroszkóppal való fémezett felületű minta felületi töltéssűrűségének mérése során kritikus a tű és a minta közötti kapacitás ismerete. A kapacitás értékét közelítések mellett lehetséges analtikusan kifejezni. Numerikus szimulációval pontosabban ismerhetjük az értékét, ezáltal nagyobb felbontás is érhető el. A szimuláció során a lehetséges párhuzamosításokat felhasználva a számítási időt elfogadhatóra csökkentettük.

Abstract

The measurement of the surface charge density can be achived by Atomic Force Microscope (AFM). The common method is the two-pass technic, which main goal is to seperate the net force acting upon the tip into components. To do so we have to express the tip-sample capacitance, which is critical respectivly to the measurements accuracy.

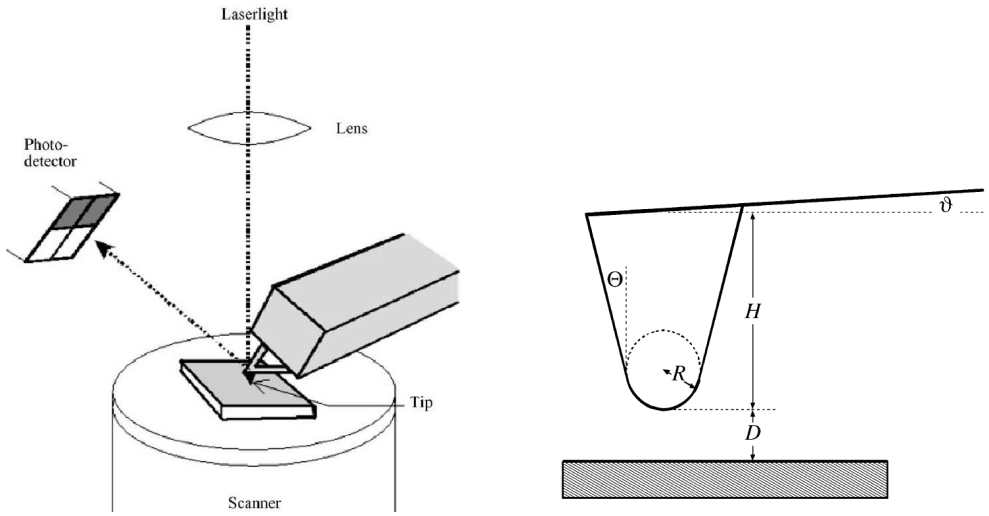
The value of the capacitance can be expressed analitically, but with some strong neglecting.

We are presenting the accelerated simulation of this capacitance using GPU's programmed in OpenCL's parallel framework.

1. fejezet

Bevezetés

1986-ban Binning demonstrálta az atomerő mikroszkóp (AFM) ötletét [1], ami mára a nanotechnológia egyik legfontosabb eszköze képalkotásra használható képalkotásra, nanolitográfiára és adott anyag alakítására [2]. Az AFM apparátusa az adott minta felületé és pásztázó kantilever végére erősített tű kölcsönhatásának vizsgálatát végzi. (lásd 1.1b. ábra) A felület és a tű közötti domináns kölcsönhatás határozza meg, hogy az anyag melyik fizikai mennyiségét kaphatjuk meg.



(a) Az AFM apparátusa **(b)** A tű és minta modellje

1.1. ábra. *Az AFM apparátusa látható az (a) ábrán. A lázernyaláb a tű felületéről tükröződve egy fotódetektorra irányul. A tű pozícióját ez alapján nagy pontossággal ismerjük. A (b) ábrán a minta és a felette lévő tű modellje látható a kapacitás analitikus számításához. A tű R sugarú H magasságú és D távolságra van a mintától. (Forrás: [3])*

Az AFM felhasználása kontakt illetve kopogtató üzemmódú lehet. A kontakt mód során a felületen végighúzzuk a tűt és mérjük a z irányú elmozdulását. Így képesek vagyunk a minta felületén lévő atomok elrendezéséről magasságtérképet adni. Kopogtató mód [4] során a tűt elemeljük a mintától és f frekvenciával rezegtetjük. A letapogatás során az átlagos minta-tű távolságot a kontakt módú magasságtérkép

felhasználásával konstans értéken tartjuk. A kantilever dinamikáját ismerve a rezegtetés frekvenciájának eltéréséből számítható a tűre ható erő. Ezen erő nagyságát befolyásoló tényezők:

a) a minta és a tű kapacitására kapcsolt feszültség,

b) a minta felületi töltéssűrűség eloszlása,

c) Van der Waals erő.

A dolgozatban a felületi töltéssűrűség mérését tekintem célnak.

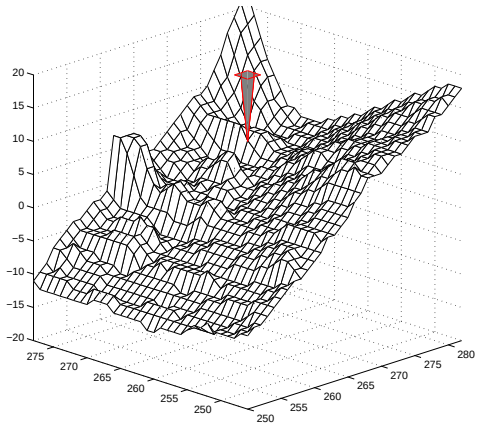
A [5, 3] szerint az erő a komponensét a minta és a tű közötti kapacitásból az (1.1) szerint származtathatjuk.

$$F_s = -\frac{\mathrm{d}E}{\mathrm{d}D} = -\frac{\mathrm{d}(CV^2/2)}{\mathrm{d}D} = -\frac{1}{2}\frac{\mathrm{d}C}{\mathrm{d}D}V^2 \tag{1.1}$$

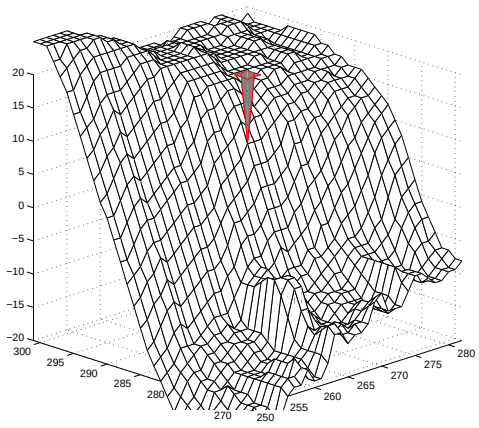
Ha a minta pásztázása során ezen a erőkomponens konstansnak mondható, tehát a felületi érdesség és a távolság pontatlansága elhanyagolhatóan kicsi, akkor a töltéssűrűség mérésében ez állandó hibát okoz. ami eliminálható. Az (1.1) számításában a kritikus elem a kapacitás értéke, amit numerikus számítás mellőzése esetén a [6] szerinti analitikus eredményt használhatjuk fel. A tű formáját a 1.1b ábra szerintinek veszi és a mintát sík felületnek feltételezi. A tűre vonatkozó feltételezés legtöbb esetben helyénvaló, viszont a minta nagyfokú érdessége és változatossága végett érvényét veszti. Szemléltetését a 1.2 ábrán lehet látni. Ilyen esetben a kapacitás értéke mintáról mintára változik és állandó hiba helyett, a mérést zajként terheli.

Ezen zaj kiküszöbölése a kapacitás numerikus szimulációjával lehetséges.

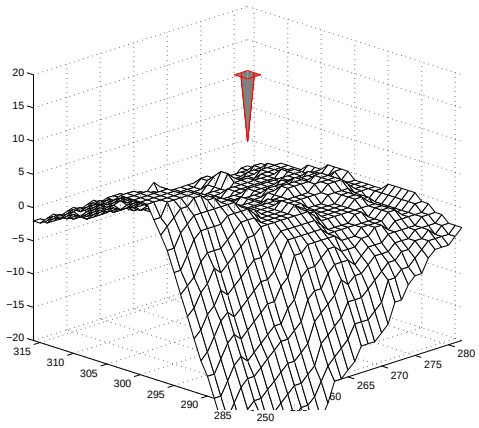
A probléma ezzel, hogy ezen szimulációt minden egyes mérési pontban el kell végezni, aminek a kivitelezése csak multiprocesszoros környezetben lehetséges elfogadható idő alatt.



(a) 1. pillanat



(b) 2. pillanat



(c) 3. pillanat

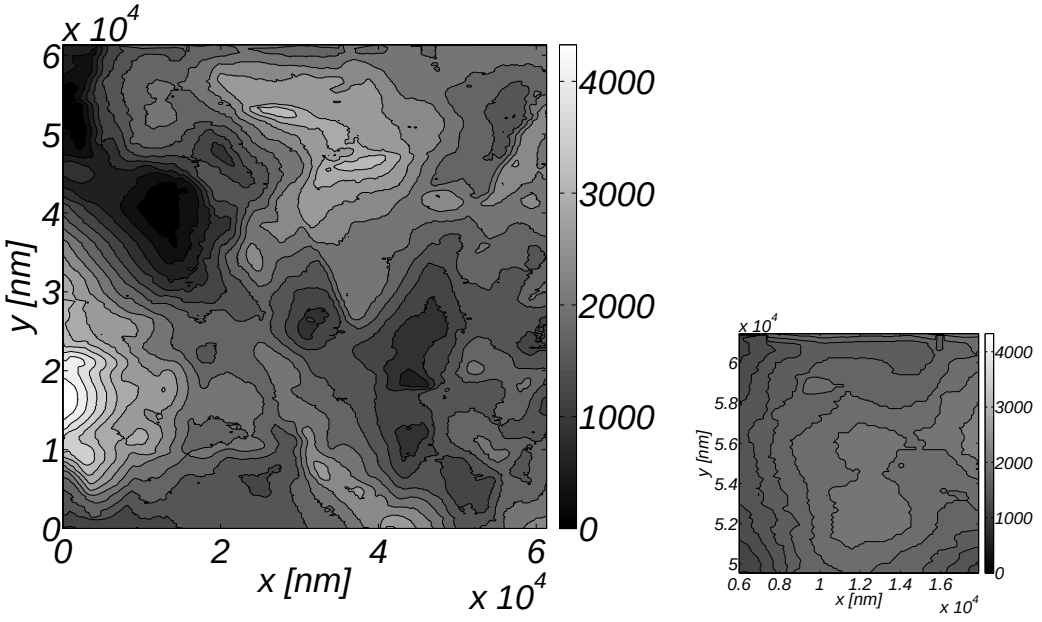
1.2. ábra. *A magasságtérkép változása a második mérés során fix pozíciójú tű esetén.*

2. fejezet

A feladat

A minta egy fémezett felület, aminek magasságtérképét mérések eredményeként adott pontossággal ismerjük. A magasságmérések egy négyzetes háló felett történtek, amelynek mindkét irányában $\delta_x = \delta_y \simeq 120nm$ azonos a felbontása A magasságmérés függőleges pontossága $\delta_z \simeq 20nm$ volt. A töltéssűrűség méréséhez szükséges második pásztázás során a tűt a mintához képest $V_{tu} = 500mV$ potenciálra kapcsoljuk. Illetve a fémezett felülettől közel azonos távolságra rezegtetjük és a (hosszú hegyes) tűre ható erőt mérjük.

A dolgozatban felhasznált magasságtérkép-mérési eredmény (2.1. ábra) egy 512×512 méretű szürkeárnyalatos *.tiff állomány, amely értéke $0 - 255$-ig terjed. A mérőberendezés adatlapja alapján és a mérés végén megjelenített konstansokkal lehetséges a kép skálázása.



(a) Mérési eredmény **(b)** Mérés részlete

2.1. ábra. *Méréssel kapott magasságtérkép. Felbontás* $d_x = d_y = 120nm$ $d_z = 18.03nm$

A dolgozat céljául egy olyan szimulátor építését tűztem ki, amely segítségével közel valós időben lehetséges a mérés alapján a felületi töltéseloszlásról a mérőeszköz pontosságánál finomabb felbontást elérni.

2.1. Fizikai probléma matematikai formalizálása

A megoldandó feladat egy elektrosztatikus feladat. A minta és a tű közötti térben nincsenek töltések, így itt a Poisson egyenlet helyett a Laplace-egyenlet 2.1 érvényes.

$$\Delta V(x, y, z) = 0 \tag{2.1}$$

Az egyenletet a következő részben (2.2) ismertetett megfontolások végett egy redukált 3D-s térfogatban oldom meg. Ezen 3D-s térfogatra egy inhomogén ponthálót illesztünk, amelynek vízszintesen $d_{ax} = d_{ay}$, függőlegesen d_z a felbontása, ami a használt AFM apparátus felbontásával $d_z = \delta_z$ egyezik meg. Az így kapott térbeli háló minden pontjához hozzárendeljük az $V_{i,j,k} \simeq V(id_x, jd_y, kd_z)$ potenciált. Dirichlet határfeltételek a felület fémezése, amely zérus potenciálú és az adott ($V_{tu} = 500mV$) potenciálú tű fémes felülete. A térnek a minta és a tű felületétől különböző határfelületén homogén Neumann feltételt alkalmazunk az elhanyagolások (végtelen tér) és a töltésmentesség miatt.

Az így adódó lineáris egyenletrendszer megoldására lehetséges direkt és iteratív megoldó algoritmusokat alkalmazni. A párhuzamosítási szándékok miatt az iteratív megoldást választottuk, mivel a multiprocesszoros környezetek tipikusan kevés fajlagos-memóriával [1] rendelkeznek. Ekkor nem teljesen pontos megoldást kapunk, azonban gyorsabban juthatunk el az elfogadható eredményhez. A számítási pontosság növelhető az iterációt leállító konvergencia követelmény keményebb megszabásával, ami persze több iterációt jelent.

Az iteratív megoldás során a megoldás aktuális értékének kiszámításához az előző megoldásból indulunk ki. A (2.1) egyenletben szereplő deriválást az elsőrendű Taylor közelítés alkalmazásával a (2.2) szerinti 6-pontos sémát kapjuk.

$$V_{ijk}^{n+1} = \Delta_1 \cdot \left(V_{i-1,j,k}^n + V_{i+1,j,k}^n + V_{i,j-1,k}^n + V_{i,j+1,k}^n\right) + \Delta_2 \cdot \left(V_{i,j,k-1}^n + V_{i,j,k+1}^n\right) \tag{2.2}$$

ahol $V_{i,j,k}^n$ az az n-dik iterációs lépésben az i, j, k indexű pontban mérhető potenciált jelöli, Δ_1 a vízszintes felbontásból, Δ_2 a függőleges felbontásból adódó állandó.

[1] Fajlagos alatt az egy szimulációra jutó memóriát értem. (Természetesen ezen szimulációk egyszerre futnak, így a fajlagos-memória akkumulálódik.)

2.2. Szimulálandó térfogat

A felületmérés során a függőleges pontossághoz képest a vízszintes felbontás jóval kissebb (ritkábbak) $\delta_x = \delta_y = 120nm \gg \delta_z = 10nm$ (lásd 2. rész). A Coulomb-kölcsönhatás a távolság négyzetével fordítottan arányos, így az előbb említettek értelmében egy mérési pont szomszédjait, pontosabban egy redukált környezetét szükséges csupán szimulálni. Hiszen azon kívül már elhanyagolható a villamos térerősség. (Másképpen megfogalmazva a vízszintes mérési pontok távolsága jóval nagyobb mint a Coulomb-kölcsönhatás effektív távolsága). Ezzel az elhanyagolással a feladat már numerikusan kezelhető méretűre csökken.

$1/r$-es közel tés esetén 3 szélére mek ra lesz

Ezen módon egy mért pont 3×3-as $(360 \times 360nm)$-es környezetét veszem figyelembe, a középső pont felett lévő elektródát (tűt) feltételezve. A szimulációs tér alsó felületét (minta felületét) az említett környzet adja.

Mivel ezen pontokra úgy is tekinthetünk, mint a minta magasság-függvényének a mintavételezésével kapott mintáira, így a közbülső pontokat interpolációval (bilineáris interpolációval, mozgó átlagolással) számítom. Az interpoláció N_{ip} faktorával [2] lehetséges a szimulációs tér vízszintes felbontását $d_{ax} = d_{ay} = \delta_x / N_{ip}$ megkapni.

Interpolációs szükségessé

2.3. Szimuláció felépítése

A mérési pontokhoz tartozó szimulációk során a felület magasságának mérési adatait már ismertnek feltételezzük. A teljes magasságtérkép pontjait külön-külön vizsgáljuk. Egyetlen pontban a mérési eredmény kiszámításának lépései az alábbiak :

1. A mérési pont körüli 3×3-as felület részének megállapítása,
2. Közbeeső (virtuális) mérési pontok interpolációval történő generálása a felbontás növelése végett,
3. Szimulálandó térfogat méretének számítása,
4. Direkt/iteratív megoldó algoritmussal a tér meghatározása, a tűre ható erő számítása illetve a tű alatti töltésmennyiség számítása,
5. Adatok mentése.

[2] N_{ip}-szeresére növeljük a pontok számát.

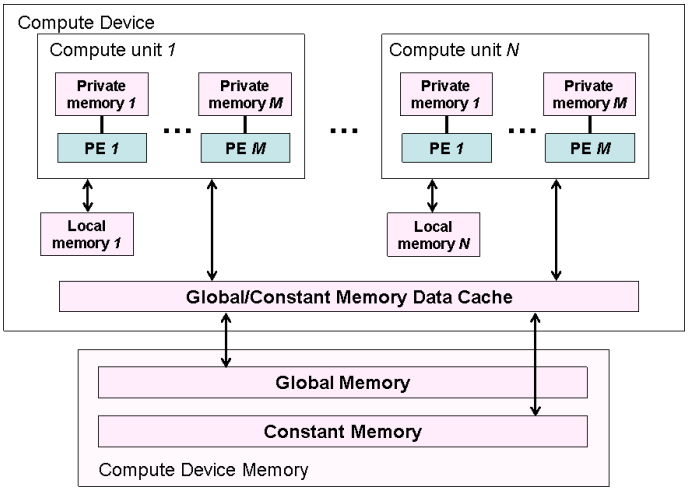
3. fejezet

A multiprocesszoros OpenCL környezet

3.1. OpenCL architektúrája

Az Open Computing Language (OpenCL) keretrendszer [7] általános modellt, magas szintű programozási interfészt és hardware absztrakciót nyújt a fejlesztőknek adat- vagy feladat-párhuzamos számítások gyorsítására különböző számítóegységen (CPU, GPU, DSP, FPGA, ...). A hárdvergyártók implementálják az OpenCL szabványban írtakat, ami által saját platformot hoznak létre. Egy ilyen platformon belüli eszközök alatt a korábban említett számítóegységeket értjük. OpenCL keretrendszerben történő programozás során két programot kell írni Az egyik a kernel, ami az eszközön (device-on) futatott szálra fog leképeződni. A másik a gazda processzoron (host-on) futó host-program, ami elvégzi az I/O műveleteket, a probléma összeállítását, a memória allokálást, az allokált terület inicializálását, a kernel argumentumainak beállítását, illetve a kernel meghívását az eszközön. A kernel futása végeztével a host-program kiolvassa az eszközből a kívánt eredményt.

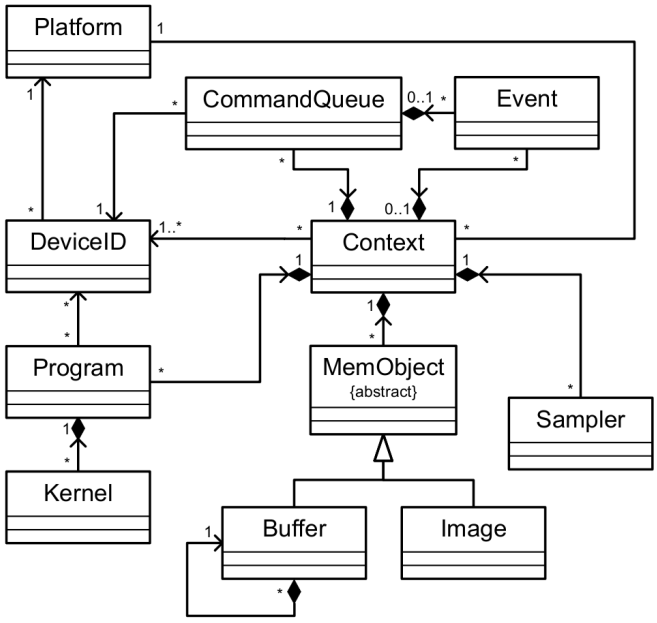
Az eszközök multiprocesszoros architektúrával és ezek kiszolgálására képes memória architektúrával rendelkeznek. Az architektúra heterogén való kezelésére a 3.1 ábrán vázolt modelljét nyújtja. Egy eszköz több compute unit-ot (processzor-magot) tartalmazhat, amikhez lokális memória tartozik és elérése van az eszköz globális memóriájához.



3.1. ábra. *OpenCL device architektúra (forrás: [7])*

3.2. OpenCL programozási modell

A programozási modell a hozzá tartozó osztálydiagrammon (3.2. ábra) figyelhető meg.



3.2. ábra. *OpenCL context osztálydiagrammja (forrás: [7])*

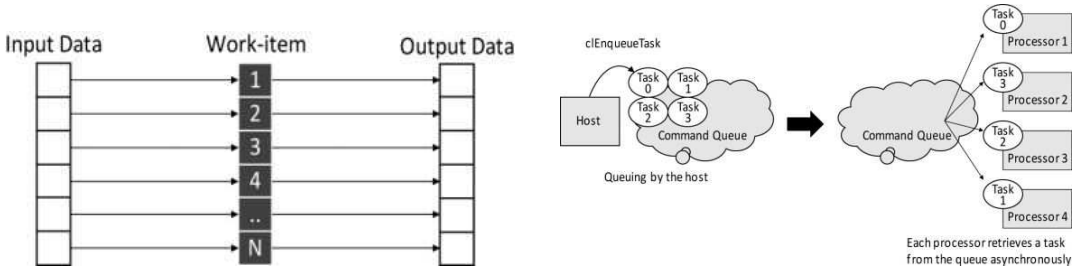
A futtatáshoz szükséges, hogy a kontextushoz platformot, majd azon belül eszközt, az eszközhöz programot (kernelt) és memóriát rendeljünk. Figyelembe kell vennünk azt a megkötést, hogy csak az egy platformon belüli eszközök programozhatóak heterogén módon. Például: Intel platform esetén lehetséges CPU-t, processzorkártyát és Intel-es GPU-t programozni, viszont nVidia kártyákat már nem.

A programozással megoldandó problémát kétféleképpen lehetséges a feldolgozó egységekhez (work-item) avagy processzorokhoz rendelni: adat parallel módon vagy

taszk parallel módon.

Adat parallel módon (3.3a ábra) a feldolgozandó adat egy részéhez rendelünk egy feldolgozó egységet. Fontos figyelembe venni az eszköz korlátos számú feldolgozó egységének számát. Ha nem elég a feldolgozó egysége akkor a feladat megfelelő partícionálásával lehetséges kordában tartani a szükséges erőforrás számát.

Taszk parallel módot (3.3b ábra) olyan esetben célszerű használna, ha a bemenet dinamikus mérete a futási időben rendkívül változik illetve a végrehajtandó feladat lazán függenek össze.

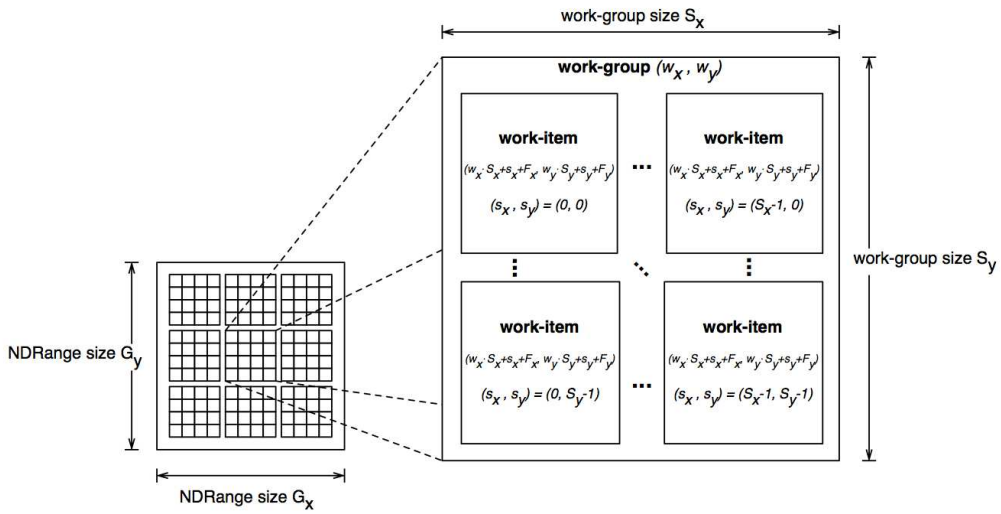


(a) Adat parallel **(b)** Taszk parallel

3.3. ábra. *Feladat hozzárendelése work-item-hez (processzorhoz)*

A processzor-magok megfelelő kihasználtságának elérése végett több ezer work-item virtuálisan osztozik rajta. Ezen work-item-eket work-group-okba rendezzük. A work-itemeket jelen pillanatban az OpenCL specifikációja [7] szerint 3 dimenziós work-group-ba tudjuk rendezni. Egy példát láthatunk egy work-item indexének a globális és lokális megfelelőjére a következő 3.4. ábrán.

Írni a csop
barendezés
motivációjá
ró » SIMD
vector



3.4. ábra. *2D-s work-item-ek work-group-ba rendezése és indexelése (forrás: [7])*

Az OpenCL a fizikailag kialíkított memóriákat négy kategórába sorolja a tipikus memóriák kategorizálása a következők:

- *Regiszterek:* Private memory,
- *Chipen belüli memória (cache):* Local memory,
- *Chipen kívüli memória:* Global memory és Constant Memory.

A privát és a lokális memória kis méretűnek és gyors elérésűnek mondható, míg a globális és konstans memória nagynak, de lassú elérésűnek. A definiált memória szintek tipikus paraméterei a következők:

- *Privát memória:* minden szálnak külön van, amit a compiler menedzsel.
- *Globális és konstans memória:* Bank szervezés jellemző rá, gyors, de nagy késleltetéssel bír. 8 műveletet tud egy ciklus alatt végrehajtan, de csak a parancs kiadása után 400-800 ciklus késleltetéssel. Ha az adaton végrehajtandó műveletek ideje ennél jóval nagyobb, akkor ezt a késleltetést el lehet rejteni.
- *Lokális memória:* Minden work-group-nak sajátja van. Minimális a késletetés, sebessége nagy (38-44 GB/s).
- *Host memória:* Nagy méretű, közepes sebességű, de sebességét a host-ot és a device-t összekötő PCIe interfész korlátozza.

A memóriákra megkötésként szolgál, hogy ki allokálhat, írhat és olvashat belőle. A 3.1. táblázatban látható ezen jogosultságokat és a főbb tulajdonságokat.

3.1. táblázat. *OpenCL memória szintek*

	Host memory	Global memory	Local mem.	Private mem.
Host	Dinamikusan R/W	Din. R/W	Din. R/W	-
Kernel	-	R/W	Satik. R/W	Statik. R/W
Sebesség	Lassú	Közepes	Gyors	Regiszter
Méret	4 Gb $\leq$	1 Gb $\leq$	$16, 32$ Kb	$<$ 1 Kb

A work-group-okba rendezés a lokális memória jogusultsága miatt érdekes. Konkrétan az egy work-group-ba tartozó összes work-item azonos lokális memórián osztozik. Ennek a következménye az, hogy adat parallel módú feldolgozás esetén az egymásra ható adatokhoz tartozó work-item-eket egy work groupba kell rendelnünk. Ha ez nem lehetséges, akkor a globális memóriához kell fordulnunk. A globális memória avagy a bank szervezésű külső (off-chip) memóriák hozzáférési ideje relatíve nagy így ezek használatát lehetőleg el kell kerülni és a programozónak kell „cachelni" a lokális memóriába.

Mivel a work-item-ek konkurrensen hajtódnak végre, így az általuk közösen elérhető memóriákra (globális, lokális) nézve versenyhelyzetben vannak. Az OpenCL

ezt a problémát a laza memóriamodell használatával oldja meg. Work-item-ek közötti szinkronizációra egy korlátot alkalmaz a programban, amit csak akkor léphet át, ha az összes többi work-item (az azonos work-group-ban) ezt a korlátot már elérte. Erre a `barrier(FLAG)` függvényhívás szolgál. Fontos megjegyezni, hogy ez a szinkronizáció csak egy adott work-group-on belül történik, a work-group-ok közötti és akülönböző work-group-okon belüli work-item-ek szinkronizációra nincs lehetőség.

Összefoglalva: nagy hangsúlyt kell a memóriaszervezésre fordítani, hogy a processzormagok megfelelően legyenek az adatokkal táplálva az elérhető számítási kapacitás kiaknázása végett.

3.3. Futási környezet bemutatása

A következő eszközök teljesítményét vizsgálom:

- A laptopomban található **Intel Core i5 M520** processzor,
- A laptopomban található kis teljesítményű **nVidia GT330M** videókártya,

Ezen eszközök legjelentősebb paraméterei a 3.2 táblázat tartalmazza.

3.2. táblázat. *Használandó eszközök összehasonlítása*

	Intel Core i5 M520	nVidia GT330M
MAX COMPUTE UNITS	4	6
MAX CLOCK FREQUENCY	2400 MHz	1265 MHz
MAX WORK GROUP_SIZE	8192	512
GLOBAL MEM SIZE	$4GB$ ($2GB$)	$1GB$ ($768MB$)
LOCAL MEM SIZE	$32KB$	$16KB$

Az összehasonlíthatóság végett a legkissebb memóriájú eszközre fogom a problémát skálázni, ami a GT330M videókártya. A CPU memóriája jóval nagyobb, így a kód mindegyiken tud futni.

4. fejezet

Multiprocesszoros program

4.1. A szimulátor dekomponálása a párhuzamos feladatokra

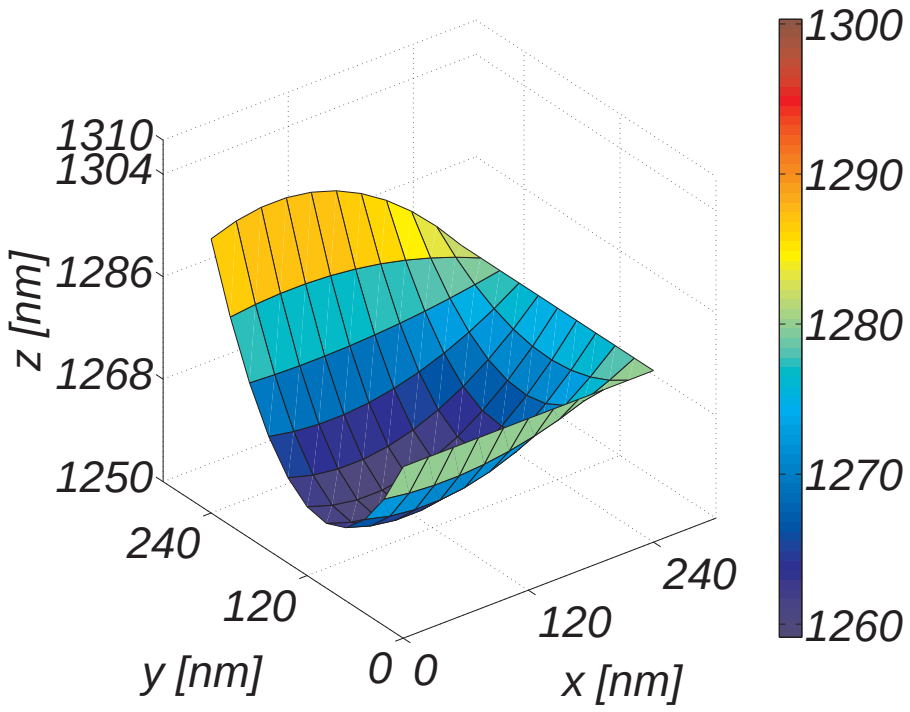
4.2. A szimulátor bemutatása

A prototípus algoritmus fejlesztése MATLAB környezetben történt, ami a későbbi referenciaként szolgál. Alap MATLAB utasításokat használva több órát vesz igénybe a szimuláció futtatása. A MATLAB Parallel Toolbox-nak segítségével a szimulációt lehetséges párhuzamosan több processzormagon futtatni. Ezzel párszoros sebesség növekezés érhető el a MATLAB magas nyelvű programnyelve végett. A következőkben magát az algoritmust és az OpenCL keretrendszerben történő implementációját mutatom be. Majd az eredmények bemutatása után összevetésre kerül a MATLAB referencia, a MATLAB Parallel Toolbox segítséggel, az OpenCL processzoron és az OpenCL GPU-n való futási ideje.

4.3. A lépések részletezése

4.3.1. Interpoláció

A korábban elmondottak alapján a felületet további virtuális pontokkal egészítjük ki. Az virtuális mérési pontokat a legegyszerűbben bilineáris interpolációval (síklapos közelítéssel) becsülhetjük. A szimulátorban egy általánosabb módszert alkalmazunk, ami egy 2D-s mozgó átlagoló szűrővel való simítás. A szűrővel aluláteresztést tudunk elérni, ami a minta magasságának mintavételezése utáni rekonstrukcióját jelenti. Egy ilyen interpoláció eredményét láthatjuk a 4.1. ábrán.



4.1. ábra. 3×3 *mérési pont* 11×11 *pontba való interpolációja*

4.3.2. Szimulálandó térfogat méretének számítása

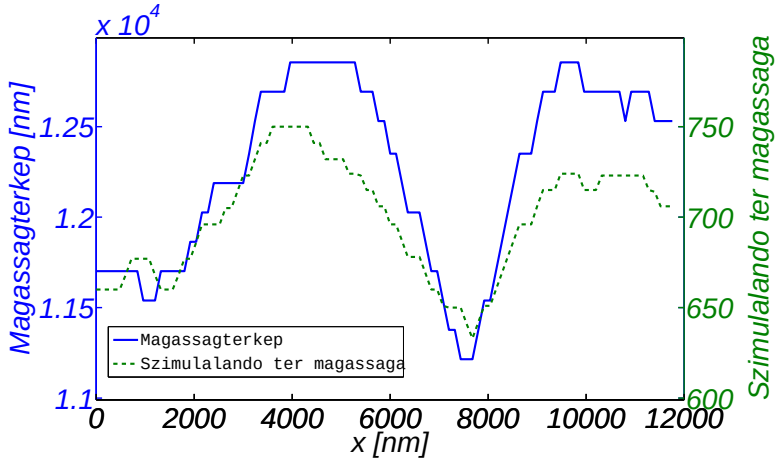
A szimulálandó tér (hasáb) alapja adott az előzőleg említett interpolált felületként, míg a magassága nem. Ezt a következő két mennyiség közül a nagyobbikkal határoztuk meg:

- Középső pont fölött lévő tű közepének magassága,
- A (3×3) környezet legalacsonyabb és legmagasabb pontjának különbsége.

A magasságtérkép egy vonalának részlete látható a 4.2. ábrán, továbbá a szimulálandó tér (hasáb) magassága.

4.3.3. Iteratív megoldó algoritmus

Az iterációhoz a térháló pontjaihoz két mátrixot (tömbböt) rendelünk, ami a pontok potenciáljának aktuális ($\mathbf{U_{now}}$) és előző ($\mathbf{U_{prev}}$) értékeit tartalmazza. Az aktuális értékeket (2.2) szerint számítjuk, majd az egész térre számítjuk az előzővel vett különbségének négyzetösszegét (normáját). E mérték képviseli a konvergencia szintjét, amit az iteráció során vizsgálva jutunk el a kívánt konvergencia szintre. Ha nem értük el a konvergencia szintet, akkor az előző két mátrixot felcserélve iterálunk tovább.



4.2. ábra. *A mérési eredmény egy vonal menti részlete (folytonos vonal) és az ezen mérési pontokhoz számított szimulációs tér magassága (szaggatott-vonal)*

4.3.4. Adatok mentése

Tesztelhetőségi megfontolások végett nem csak a tűre ható erőt (villamos térerősséget) exportáljuk, hanem a konvergencia szintjének változását és az interpolált felületet is. Az exportálandó menyiségek „kis" mérete miatt egyszerű *.csv fájlként kerülnek mentésre. Ezen fájlok további poszt-processzálása MATLAB vagy munkalap kezelő szoftverrel is elvégezhető.

4.4. Implementációhoz szükséges megfontolások

A következőkben egy kissebb teljesítményű notebook videókártyát veszek alapul a megfontolások demonstrálására. Ez az nVidia GeForce 330M, 575 MHz-en futó 48 CUDA core-al, 1024GB memóriával és OpenCL 1.0 kompatibilitással. A videókártya továbbiakban fontos paraméterei a 4.1. táblázatban látható.

4.1. táblázat. *nVidia GeForce 330M OpenCL tulajdonságai*

MAX_COMPUTE_UNITS	6
MAX_WORK_GROUP_SIZES	512 512 64
GLOBAL_MEM_SIZE	1073020928
MAX_CONSTANT_BUFFER_SIZE	65536
LOCAL_MEM_SIZE	16384

Ha a tér, ahol a laplace egyenletet meg kell oldanunk nagyon nagy, akkor érdemes szétbontani kissebb alterekre és azokhoz rendelni egy-egy „work-item"-et. Mivel a diszkrét Laplace egyenlet egy pontja a szomszédos pontokkal szoros kapcsolatban van, így az összefüggő „work-item"-eket egy „work-group"-ba érdemes szervezni, mivel így az átlapolódó pontok értékét a szomszédos „work-item"-ek is tudják írni és

olvasni. Az ilyen típusú problémának méretét a MAX_WORK_GROUP_SIZES tulajdonság korlátozza.

Jelen esetben a mérési eredmény egy pontjához tartozó tér átlagosan $11 \times 11 \times 30$ pontból áll. Tehát a korábbi nem áll fenn és egyszerű megfeleltetéssel szétoszthatjuk a feladatot. A teljes tér $512 \times 512 \times 11 \times 11 \times 30$ méretű, ami $951k$ pont. A tárolásához single-precision mellett ennek a számnak a 4-szerese szükségeltetik byte-okban mérve. Mivel ez a videókártyán nem áll rendelkezésre, így szétbontjuk kissebb feladatrészekre.

Ezen feladatrészek méretét egy paraméter állításával lehet változtati és az implementált algoritmus ettől generikusan függ. Emellett az interpoláció mértéke N_{ip} is paraméterrel generikusan állítható. Az algoritmus generikusságát csupán a futási időben történő dinamikus memória allokációval lehetséges megvalósítani. A korábban említettek végett (3.1 táblázat) az allokáció csak a „host" programban történhet.

4.5. Memória szervezés

4.5.1. Csak globális memória használata

Az algoritmus pszeudó kódjának direkt leképezése esetén a „host"-on allokálunk memóriát a „device" globális memóriájában, majd a megfelelő adatokat ide másoljuk és a kernel is itt ír és olvas. A problémát a globális memória nagy hozzáférési ideje jelenti, ami miatt sok „work-item" tétlenül a memóriára fog várakozni. Ilyenkor az egy mérési pontra vonatkoztatott szimulációs idő a referenciánál is lassabb.

4.5.2. Globális memória és adott esetben lokális memória használata

Kis erőfeszítéssel nagy javulást lehet elérni, ha a mérési ponthoz tartozó szimulációs tér éppen belefér a lokális memóriába. Tehát, mielőtt az (2.2) szerinti iteratív megoldót futtatnánk először a globális memóriából a lokális memóriába töltjük át a kérdéses pontokat, majd számolunk rajta és a végén visszatöltjük a globális memóriába. E javítással a referenciával azonos sebességet tudunk elérni.

4.5.3. Globális memória és minden adódó alkalomkor a lokális memória használata

Nagyobb erőfeszítést igényel, hogy minden alkalommal a globális memóriával való kommunikációt a lokális memória közbeékelésével tegyük. Ezt úgy lehet felfogni,

4.2. táblázat. *OpenCL futási idő eredmények* 12×12 *mérési pontra*

	Globális memória	Lokális memória, ha befér	Lokális memória bufferelés
Globális tranzakciók száma átlagosan	$12 \times 12 \times 32.3$	$12 \times 12 \times 32.3$	$12 \times 12 \times 32.3$
Lokális tranzakciók száma átlagosan	0	$0.48 \times 12 \times 12 \times 30$	$2.08 \times 12 \times 12 \times 32.3$
Futási idő	5990 ms	2530 ms	510 ms
Fajlagos futási idő	410 ms	170 ms	3.5 ms

mintha a globális memóriát lokális memória méretű kvantumokban tudnám csak elérni. Ekkor nagy odafigyelést kíván a memóriacímzés megfelelő prgramozása, de eredményképp gyorsulás érhető el.

Összegezve elmondható, hogy az aktuálisan használt adat tárolását a lehető legközelebb kell tartani a „compute-unit”-hoz.

5. fejezet

Konvergencia vizsgálata

5.1. Interpolációk közötti különbségek

5.2. Különböző alap formák esetén

6. fejezet

Eredmények

6.1. MATLAB implementációk

A referenciaként szolgáló MATLAB algoritmus lineáris programszervezést alkalmazva az elérhető fajlagos futási idő $\sim 100ms$.

A kód minimális változtatásával elérhető a párhuzamos végrehajtás. Ezt a `for` ciklusok Parallel Toolbox beli `parfor` utasítására cserélve érhetjük el. 4 processzormaggal rendelkező PC-n legjobb esetben a negyedére csökkenhet a futási idő. Valójában ez sose történik meg. A MATLAB Parallel Toolbox-a az egyes szimulációkat adott processzormagra osztja el. Korábban láttuk, hogy ezen szimulációk lépésigénye nagyban eltér egymástól, így előáll az a sajnálatos eset, hogy 3 mag tétlenül a negyedikre vár, ami miatt ez nem tekinthető járható útnak.

6.2. OpenCL implementációk

OpenCL keretrendszer segítségével írt programot a GPU-n futtatva az 4.2. táblázatban látható eredményeket kapjuk. Csupán a globális memóriát használva a referenciához képest romlik a teljesítmény. Ezt a videókártya prediktív cache nélküli kialakításának és a globális memórájának a kiéheztetésének tudhatjuk be. A lokális memória használata a futási időt drasztikusan le tudja csökkenteni, ami a korábban ismertetett memória szervezési megfontolások helyességét igazolja.

xx

A programot a különböző eszközökön futtatva a 6.1. táblázatban látható futási eredményeket produkálta. A táblázatban látható futási idők 100 futási idő átlaga. A táblázathoz felvettem egy fiktív mérőszámot (teljesítmény tényező) a különböző architektúra összehasonlítására. A mérőszám értéke minél kissebb, annál gyorsabban hajtódik végre egy utasítás.

6.1. táblázat. *Az eszközök erőforrásainak és a rajta futtatott programok futási idejének összehasonlítása.*

	Intel Core i5 M520	nVidia GT330M
MAX COMPUTE UNITS [1]	4	6
MAX CLOCK FREQUENCY [MHz]	2400	1265
MAX WORK GROUP_SIZE	8192	512
GLOBAL MEM SIZE	~ 4 GByte	~ 1 GByte
LOCAL MEM SIZE	32 KByte	16 KByte
Futási idő (T)	478.71 ms	191.94 ms
Teljesítmény tényező $\left(P = \frac{1}{\text{UNITS}\times\text{FREQUENCY}}\right)$	$104.16 \cdot 10^{-6}$	$131.75 \cdot 10^{-6}$
Fajlagos utasításszám (T/P)	$4.59 \cdot 10^3$	$1.45 \cdot 10^3$

Látható, hogy a GPU-n való futtatás közel 666× gyorsulást jelent. Ezt két dolognak tudom be:

- *Memória:* A CPU memóriája DDR3 @ 1066 MHz 64 bites busz szélességgel, míg a GPU memóriája GDDR3 @ 1066 MHz 128 bites busz szélességgel,
- *Processzor mag:* A CPU 4 compute unit-al rendelkezik, ami 4 szálra, ami 2 processzormagra az Intel HyperThread technológiájával képeződik le, míg a GPU 6 compute unit-al rendelkezik, ami 48 CUDA core-ra képeződik le.

Továbbá figyelembe kell venni, hogy a program futása a többi programmal konkurrensen történik, CPU esetén az operációs rendszerrel, GPU esetén a megjelenítéssel.

7. fejezet

Összegzés

A cikkben összefoglaltam az AFM felületi töltéssűrűség méréstechnikáját, amiben azonosítottam a kapacitás értékének kritikus voltát. Ezidáig a kapacitás értékének a [6, 3] szerinti közelítéseket tartalmazó analitikus eredményt lehetett felhasználni.

Feladatként ezen kapacitás értékét számító szimulátor építését tűztem ki, aminek elfogaható időn belül kell eredményt szolgáltatnia. A párhuzamosítás lehetősége triviálisan adódott. A hordozhatóság és a gyorsabb végrehajtás végett a szimulátort OpenCL környezetben implementáltam, ami a szimulátor heterogén multiproszesszoros környezetben való futtatását lehetővé teszi.

Az eredményeket ismertetve a szimulátor futási idejében látványos gyorsulást tapasztaltam, ami az érdesebb felületek esetén is elfogadhatóan pontos eredményt tud szolgáltatni.

Végül az elkészült programot CPU-n és GPU-n is futtatva a futási idejüket összevetettem és azonosítottam a gyorsulás forrását kitérve a processzormagra és a memóriájára.

További feladatok:

- A szimuláció felhasználásával történő töltéssűrűség származtatása még várat magára, ugyanígy ezen származtatás validálására való mérési összeállítás kidolgolgozása.
- A szimulátor magját képező lineáris egyenletrendszer iterációs megoldó konvergenciájának bizonyítása és az alternatív direkt megoldó vizsgálata is további feladat.

8. fejezet

Függelék

8.1. Referencia program

`refcode.c`

8.1. lista. *A referencia programkódja*

```
main() { return; }
```

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

[1] G. Binnig, C. F. Quate, and C. Gerber, „Atomic force microscope," *Phys. Rev. Lett.*, vol. 56, pp. 930–933, Mar 1986.

[2] B. Vasić, M. Kratzer, A. Matković, A. Nevosad, U. Ralević, D. Jovanović, C. Ganser, C. Teichert, and R. Gajić, „Atomic force microscopy based manipulation of graphene using dynamic plowing lithography," *Nanotechnology*, vol. 24, no. 1, p. 015303, 2013.

[3] H.-J. Butt, B. Cappella, and M. Kappl, „Force measurements with the atomic force microscope: Technique, interpretation and applications," *Surface Science Reports*, vol. 59, no. 1–6, pp. 1 – 152, 2005.

[4] Y. Martin, C. C. Williams, and H. K. Wickramasinghe, „Atomic force microscope–force mapping and profiling on a sub 100 a scale.," *Journal of Applied Physics*, vol. 61, no. 10, pp. 4723–4729, 1987.

[5] H.-J. Butt, „Measuring electrostatic, van der waals, and hydration forces in electrolyte solutions with an atomic force microscope," *Biophys J.*, vol. 60(6), p. 1438–1444., 1991.

[6] S. Hudlet, M. Saint Jean, C. Guthmann, and J. Berger, „Evaluation of the capacitive force between an atomic force microscopy tip and a metallic surface," *The European Physical Journal B - Condensed Matter and Complex Systems*, vol. 2, no. 1, pp. 5–10, 1998.

[7] The Khronos OpenCL Working Group, *The OpenCL Specification, version 1.0*, 6 August 2010.