Elektrosztatikus mérés szimulációja multiprocesszoros környezetben

Bakró-Nagy István
Szélessávú Hírközlés és Villamosságtan Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem
Budapest
bakro.istvan@gmail.com

Reichardt András
Szélessávú Hírközlés és Villamosságtan Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem
Budapest
reich@evt.bme.hu

***Abstract*—Fémezett, töltött felület felett lévő fémtűre ható erő számítását végeztük el. A szimulációk során a lehetséges párhuzamosításokat felhasználva a számítási időt csökkentettük. Kidolgoztunk egy egyszerű szimulációs keretrendszert a felületmérés pontatlanságainak kiegyensúlyozására, az elérhető felbontás javítására.**

***Index Terms*—numerikus szimuláció, parallel computing, OpenCL, AFM**

I. Motiváció

Korábban elvégzett atomerő mikroszkópos mérések (AFM) validálása során az alábbi probléma keletkezett. A fémezett felület topológiáját mérések eredményeként ismerjük adott pontossággal. A mérések egy négyzetes háló felett történtek, amelynek mindkét irányban (xy) azonos felbontása volt. A mérés során a fémezett felülettől mindig azonos távolságra lévő, hosszú hegyes tűre ható erőt mértük, amikor a felület és a tű között adott, állandó feszültséget kapcsoltunk.

A tűre ható erőt mérve a felület tű alatti tartományban található töltéseloszlásról kapunk információt. Végső cél olyan szimulátor építése, amelynek segítségével közel valós időben lehetséges felületmérés alapján a felületli töltéseloszlásról információt kapni.

A cikkben felhasznált mérési eredmény (1. ábra) egy 512×512 méretű tömb, amely értékei $0 - 255$-ig terjed.

Fig. 1. Méréssel kapott felület (balra) és szimulációval kapott töltéstérkép

II. Szimuláció

A. Fizikai probléma leírása

A megoldandó feladat egy elektrosztatikus feladat. A határfeltételek a felület fémezése, amely zérus potenciálú és az adott potenciálú (U_{pot}) tű fémes felülete. A határok többi részén homogén Neumann feltételt alkalmazhatunk a szimmetriák és a töltésmentesség miatt.

A belső térben nincsenek töltések, így itt a Laplace-egyenlet érvényes (1).

$$\Delta\, U(x,y,z) = 0 \tag{1}$$

Ennek megoldására lehetséges direkt és iteratív megoldó algoritmusokat alkalmazni. Mindkét módszer esetében a 3 dimenziós probléma esetében téren diszkretizáljuk a problémát, amelynek vízszintes felbontása Δ_x és Δ_y, míg függőlegesen Δ_z. Az így kapott térbeli háló minden pontjához hozzárendeljük az $U_{i,j,k}$ potenciált ($1 \leq i \leq i_{max}$, $1 \leq j \leq j_{max}$, $1 \leq k \leq k_{max}$).

A párhuzamosítási szándékok miatt az iteratív megoldást választottuk. Ekkor nem teljesen pontos megoldást kapunk, azonban kevesebb lépéssel juthatunk el a kívánt eredményhez. A számítási pontosság növelhető keményebb konvergenci követelmény megszabásával, ami további iterációkat jelent.

Az iteratív megoldás során a megoldás aktuális értékének kiszámításához az előző megoldásból indulunk ki. A (2) alapján végezzük az iterációt.

$$\begin{aligned} U_{ijk}^{n+1} = \\ \Delta_1 \cdot \left(U_{i-1,j,k}^n + U_{i+1,j,k}^n + U_{i,j-1,k}^n + U_{i,j+1,k}^n\right) + \\ \Delta_2 \cdot \left(U_{i,j,k-1}^n + U_{i,j,k+1}^n\right) \end{aligned} \tag{2}$$

ahol $U_{i,j,k}^n$ az az n-dik iterációs lépésben az i, j, k koordinátájú pontban mérhető potenciált jelöli, Δ_1 a vízszintes felbontásból, Δ_2 a függőleges felbontásból adódó állandó. Az iterációs eljárás előnye, hogy implementációja egyszerűbb és a (2) szerinti "simítás" gyorsabb, mint a direkt megoldás.

B. Szimuláció felépítése

Mivel a Coulomb-kölcsönhatás a távolság négyzetével fordítottan arányos, így egy pont szomszédjait, pontosabban egy redukált környezetét szükséges csupán szimulálni. Ezzel az elhanyagolással már numerikusan kezelhető bonyulultságú problémára jutunk.

Ezen módon egy mért pont 3×3-as környezetét vesszük figyelembe, a középső pont felett lévő felső elektródát

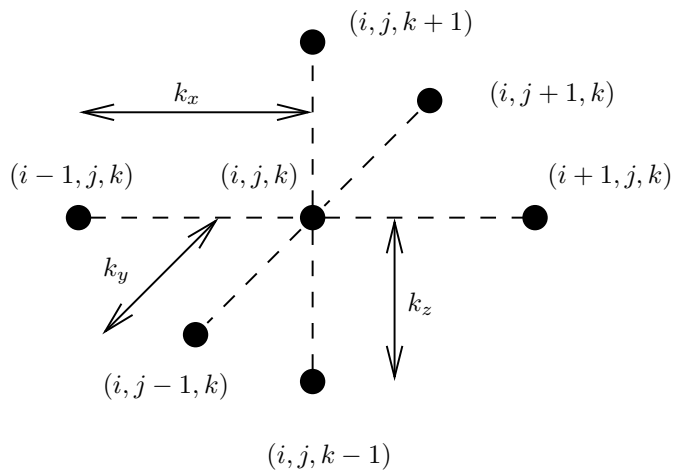

Fig. 2. Diszkretizálás során alkalmazott felosztás

feltételezve. A szimulációs felületen belül a mért magassági értékeket mint mintákat 2D-ben interpoláljuk.

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

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

III. A szimulátor bemutatása

A prototípus algoritmus fejlesztése MATLAB környezetben történt, ami később 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 következőkben magát az algoritmust és az OpenCL keretrendszerben történő implementációját mutatjuk be. Majd az eredmények bemutatása során kerül összevetésre kerül a MATLAB referencia, a MATLAB Parallel Toolbox segítséggel, az OpenCL processzoron és az OpenCL GPU-n való futtatási ideje.

A. A lépések részletezése

1) Interpoláció: A korábban elmondottak alapján a felületet további mérési pontokkal egészítjük ki. Az extra mérési pontokat a legegyszerűbb síklapos közelítéssel alkothatjuk meg, a magasságmérési felbontás figyelembevételével. 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 3. ábrán.

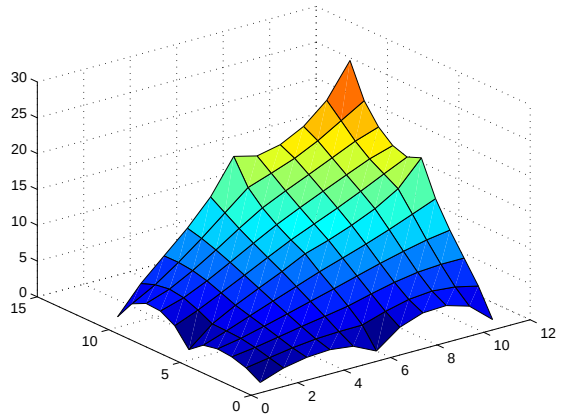

Fig. 3. 3×3 mérési pont 11×11 pontba való interpolációja

2) A szimulálandó tér mérete: A szimulálandó tér (hasáb) magasságát 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.

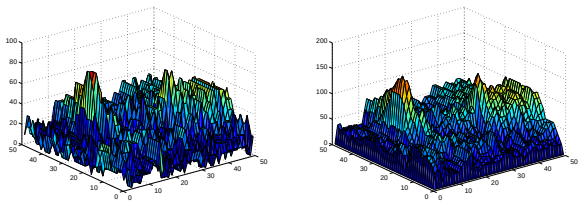

Fig. 4. A magasságmérés eredményének részlete (balra) és a hozzá tartozó szimulálandó tér magassága.

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) 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) Adatok mentése: Tesztelhetőségi megfontolások végett a nem csak a tűre ható erőt (villamos térerősséget) exportáljuk, hanem a konvergencia szintjének változását és az interpolált felületet is. Az exportálandó menyiségek könnyen kézben tartható mérete miatt egyszerű CSV fájlként kerülnek mentésre. Ezen fájlok további poszt-processzálása MATLAB avagy munkalap kezelő szoftverrel is elvégezhető.

B. OpenCL architektúrája

Az Open Computing Language (OpenCL) keretrendszer [1] közös nyelvet, magas szintű programozási interfészt és hardware absztrakciót nyújt a fejlesztőknek adat- vagy feladat párhuzamos számítások gyorsítására különböző számítóegységen (CPU, GPU, FPGA, DSP, ...). Az OpenCL modellje a különböző "device"-okat, amik több "compute unit"-ot (processzor-magot) tartalmaznak heterogén módon kezeli.

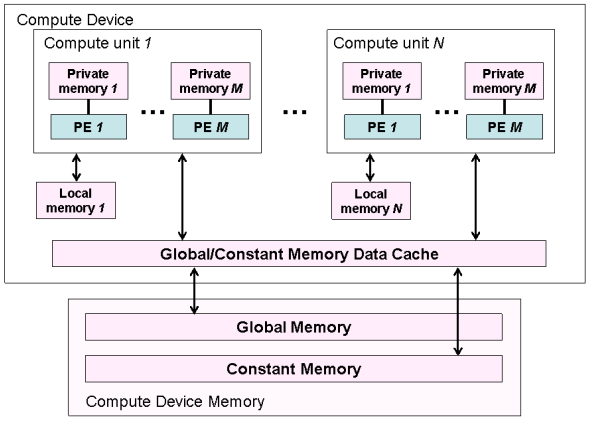

Fig. 5. OpenCL "device" architektúra [1]

A "compute unit"-ok kiéheztetésének elkerülés végett (több ezer) "work-item" virtuális osztozik rajta. Továbbá ezen "work-item"-ek "work-group"-okba vannak rendezve, később részletezett megfontolások végett. A "compute unit" kiéhesztetését a "device"-on található memória chipek lassúsága okozza. Ennek hárdveres megoldása a több szintű prediktív cache memória beiktatása a "compute unit" és a külső memória közé. Mivel a bank szervezett külső memóriák hozzáférési ideje relatíve nagy így a memória szervezésére nagy hangsúlyt kell fektetni.

Az OpenCL négy memória szintet különböztet meg, ami az I táblázatban és az 5. ábrán látható. Ahhoz, hogy a rendszerben rejlő teljesítményt kihozzuk három fontos kérdést kell a szimulátor magjának implementálásakor megválaszolnunk:

- **Mennyit?**: Tisztában kell lennünk az aktuális memória fogyasztással és a szükséges memóriamérettel.
- **Honnan-hova?**: Fontos, hogy a lehető legközelebb legyen az adat a "work-item"-hez.
- **Mikor?**: Mivel a memória művelet alatt a "work-item" nem dolgozik, így átadja a helyét egy másiknak. Ennek a megfelelő szinkronizációjával nagyobb kihasználtság érhető el (load balance).

OpenCL keretrendszerben történő programozás során két programot kell írnunk. Az egyik a "host"-on fut, ami elvégzi a probléma összeállítását, memória allokálását, argumentumok beállítását és a másik program a kernel meghívását a "device"-on. A kernel futása végeztével a "host" program kiolvassa a "device"-ból a kívánt eredményt.

TABLE I
OPENCL MEMÓRIA SZINTEK

	Global	Constant	Local	Private
Host	Dinamikusan R/W	Din. R/W	Din. R/W	
Kernel	R/W	Statikusan R	Satik. R/W	Statik. R/W
Sebesség	Lassú	Gyors	Gyors	Regiszter
Méret	1 Gbyte $<$	$\sim$ 64 Kbyte	$\sim$ 16 Kbyte	$<$ 1 Kbyte

TABLE II
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

C. 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 II. táblázatban látható.

Ha a tér ahol a laplace egyenletet meg kell oldanunk nagyon nagy, akkor érdemes szétbontani kissebb alterekre és azokhoz rendelni egy-egy "work-item"-eket. 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" 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 mérva. 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 is generikusan paraméterrel állítható. Az algoritmus generikusságát csupán a futási időben történő dinamikus memória allokációval lehetséges megvalósítani. A korábban említettek végett (I táblázat) az allokáció csak a "host" programban történhet.

D. Memória szervezés

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.

TABLE III
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

Ilyenkor az egy mérési pontra vonatkoztatott szimulációs idő a referenciánál is lassabb.

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, mielött az 2 szerinti iteratív megoldót futtatnánk először a globális memóriából a lokális memóriába töltjük át a kérdéses pontokat, majd számolunk rajt és a végén visszatöltjük a globális memóriába. E javítással a referenciával azonos sebességet tudunk elérni.

3) Globális memória és minden adódó alkalomkor a lokális memória használata: Nagyobb erőfeszítést igényel, hogy a globális memórival való kommunikációt a lokális memória közbeékelésével tegyünk minden alkalomkor. Ezt úgy lehet felfogni, mintha a globális memóriát lokális memória méretű kvantumokban tudnám csak elérni. Ekkor nagy odafigyelést kíván a memóriacímzés megfelelő prgramozása, de eredményképp gyorsulás elérhető.

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

IV. EREDMÉNYEK

A. 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 esetén ilyenkor közel a negyedére csökken a futási idő.

B. OpenCL implementációk

OpenCL keretrendszer segítségével írt programot a GPU-n futtatva a III 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 memórája okozta kiéheztetésnek tudhatjuk be. A lokális memória használata a futási időt drasztikusan le tudja csökkenteni, ami a korábban ismertetett memória szervezési gondolatok helyességét igazolja.

V. ÖSSZEGZÉS

A cikkünkben ismertettük az AFM mérések validálásának problémáját. A szimulálandó problémát ismertetve adódott a párhuzamosítás lehetősége. A hordozhatóság és a gyorsabb végrehajtás végett a szimulátort OpenCL környezetben implementáltuk. Ismertettük az implementáció során észben tartandó gondolatokat, azok általánosságának elvesztése nélkül.

Az eredményeket ismertetve látványos gyorsulást tapasztaltunk, ami nehezebben kezelhető problémáknál azonos futási idő mellett nagyob pontosságot jelenthet.

REFERENCES

[1] The Khronos OpenCL Working Group, *"OpenCL - The open stan- dard for parallel programming of heterogeneous systems."* http://www. khronos.org/opencl/, 2014.