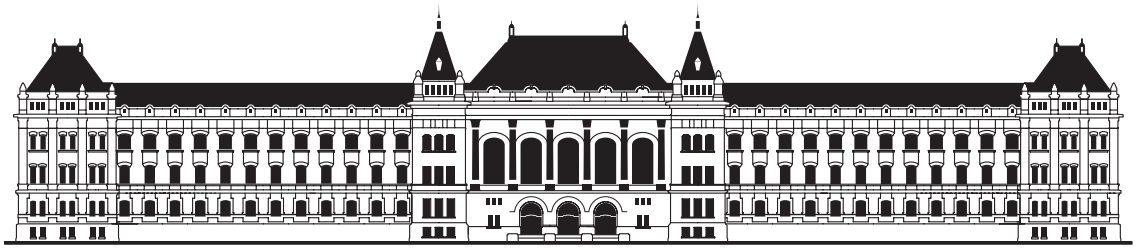


M Ű E G Y E T E M 1 7 8 2

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

Poros plazma kísérletek támogatása multiprocesszoros környezetben

Diplomatervezés 1. - beszámoló

Készítette
Bakró Nagy István

Konzulens
Hartmann Péter & Reichardt András

2014. május 21.

Tartalomjegyzék

FELADATKIÍRÁS

A modern alacsony hőmérsékletű plazmafizikai kísérletek egy új, érdekes és izgalmas területe a poros plazmák kutatása. Egy elektromos gázkisülésbe helyezett apró (mikrométer méretű) szilárd szemcse a kisülési plazma atomi részecskéivel kölcsönhatva elektromosan feltöltődik. A sok töltött szemcséből kialakuló elrendezésben a szilárdtestfizikai jelenségek széles spektruma figyelhető meg, pl. kristályrács kialakulása, fázisátalakulás, diszlokációk dinamikája, transzport folyamatok, stb. Poros plazmákat jelenleg leginkább alapkutatásokban alkalmaznak, de jelentőségük az elektronikai gyártásban, fúziós reaktorok üzemeltelésében, Teraherz technológiában egyre inkább előtérbe kerül.

A kísérleti adatgyűjtés és feldolgozás nagyrésze részecske-követő velocimetrián (particle tracking velocimetry) alapul, vagyis első lépésben egy nagysebességű kamera segítségével nagyfelbontású képek készülnek, amely képek segítségével a porszemcsék pontos (a kamera felbontásánál pontosabb) koordinátáit kell meghatározni. A képek elemzése ezidáig csak a mérést követően, hosszú idő alatt volt megvalósítható a vizsgálandó nagy adatmennyiség miatt. A multiprocesszoros környezetek segítségével a feldolgozás gyorsítása lehetséges akár több nagyságrenddel is.

A jelölt feladata, hogy a meglévő kísérleti elrendezés, amely az MTA Wigner Fizikai Kutatóközpont Szilárdtestfizikai és Optikai Intézezetben található, kiegészítésével a mérés közbeni feldolgozással a mérést segítő analízist hajtson végre. Ennek eredményével a mérés előkészítése és elvégzése lényegesen gyorsulhatnak.

A jelölt feladata

- Mutassa be a mérési elrendezést és elemezze a kapott adatokat! (Mutassa be a mérést!)
- Elemezze a lehetséges multiprocesszoros környezeteket, a feladat szempontjából lényeges paraméterek és feladatvégrehajtási elvárások szempontjából!
- Készítsen programot, amely az azonnali (valós idejű) analízisben résztvevő paramétereket számítja ki, a multiprocesszoros környezet kihasználása nélkül!
- Készítsen programot, amely a mérési környezetbe illeszkedve a mérésnél valós időben képes a vizsgált paraméterek megjelenítésére! Mutassa be és elemezze az elkészített programot!
- Hasonlítsa össze a multiprocesszoros és a nem-multiprocesszoros környezetre elkészített programokat erőforrás igény illetve egyéb paraméterek szempontjából!

Irodalom:

- [1] Hartmann P, et. al. ; "Crystallization Dynamics of a Single Layer Complex Plasma"; Phys. Rev. Lett., 105 (2010) 115004
- [2] Hartmann P, et. al. ; "Magnetoplasmons in Rotating Dusty Plasmas", Phys. Rev. Lett. 111, 155002 (2013)
- [3] Hartmann P, Donkó I, Donkó Z; "Single exposure three-dimensional imaging of dusty plasma clusters"; Rev. Sci. Instrum., 84 (2013) 023501/1-5;

Tanszéki konzulens: Reichardt András, egy. tanársegéd
Külső konzulens: Hartmann Péter, PhD., tud. főmunkatárs (MTA Wigner FK, SZFI)

Budapest, 2014.03.10.

1. fejezet

A poros plazma kísérlet

1.1. Bevezető

A poros plazma kísérlet ionizált nemesgáz és az abba szórt porrészecskék megfigyeléséből áll. Adott alacsony nyomású gáz terében elhelyezett elektródákra kapcsolt váltakozó feszültséggel lehetséges plazmát létrehozni. A váltakozó villamos tér a töltéssel rendelkező elektronokra olyan erővel hat, hogy azok leszakadnak az atomról ezzel ionizálva azokat. A leszakadó elektronok a térben szabadon a ráható erőknek megfelelően mozognak. Ez makroszkópikus skálán a gáz vezetővé válását jelenti. A szabad elektronok mozgásának során az ionizált atomokon szórodnak, ami ködfénykisüléshez vezet.

A kísérlet során az előbb ismertett plazma terébe porrészecskéket szórunk. A porrészecskék alatt $10nm - 100\mu m$ nagyságú részecskéket értünk, például SiO_2, Al_2O_3 vagy melamine-formaldehyde-t. A porrészecskék a plazmával interakcióba lépve negatívan feltöltődnek. A porrészecskék töltésének és tömegének hányadosa a szabad elektronokhoz és az ionokhoz képest sokkal kissebb, ezáltal a mozgására való hatása elhanyagolható.

A porrészecskék transzportjának megértéséhez szükséges a ráható erők azonosítása. A különféle erők nagysága a porrészecskék nagyságától különféleképpen skálázódik. Elhanyagolásokat ennek megfelelően tehetünk.

Gravitációs F_g erő: Mikrogravitációban végzett kísérletek kivételével a porrészecskére ható gravitációs erő lineárisan arányos a tömegével. Nanométer nagyságú részecskék esetén elhanyagolható, de a jelen esetben használt mikrométer nagyságú részecskék esetén dominánsak,

Villamos tér keltette F_e erő: A porrészecske töltésével és a villamos tér nagyságával arányos. A megfelelően irányított villamos térrel lehetséges a részecskék levitációja,

Háttératomon való szóródás F_n: A porrészecske driftje során a háttératomokkal való szóródásának makroszkópikus erőként való számításba vétele,

Hőmérséklet grandiensi F_{th} erő: A gáz hőmérsékletének gradiense okozta gázatomok diffúzív jellegű mozgása által okozott indirekt erőhatás,

Ion sodrási F_i erő: Az ionokra ható villamos tér okozta erőnek indirekt hatása a porrészecskékre.

A poros plazma analízise során fontos szerepet játszik a porrészecskék csatolása. A csatolást gyengének és erősnek kategorizáljuk aszerint, hogy a részecskék szomszédja általi átlagos potenciális energia az átlagos termikus (kinetikus) energiához képest kissebb vagy nagyobb. A csatolást a Coulomb csatolási paraméterrel (Γ) lehet számosítani, ami a szomszédos részecskék Coulomb potenciáljának és termikus energiájának hatványa:

$$\Gamma = \frac{Q^2/4\pi\varepsilon_0 d}{kT_d} \tag{1.1}$$

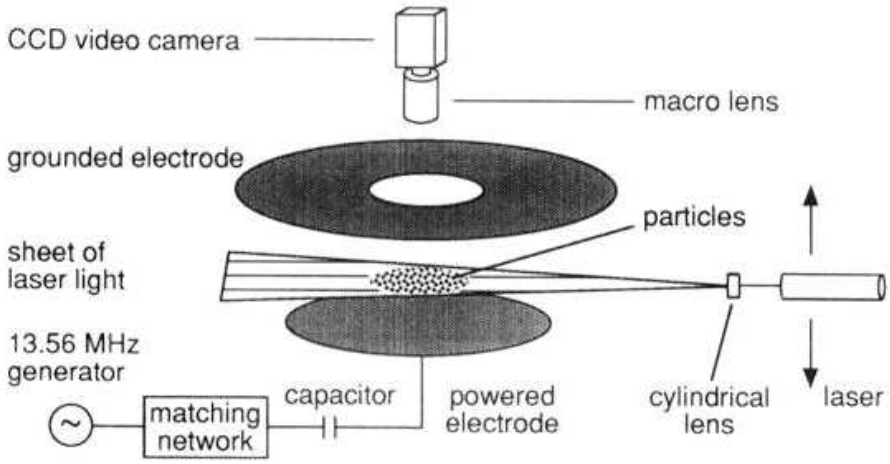
ahol Q a részecskék töltése, d az átlagos részecske távolság és a T_d a por hőmérséklete. Ha a Γ értéke egy kritikus érték fölé $\Gamma > 150$ növekszik, akkor Ikawa jóslata szerint a porrészecskék Coulomb kristályszerkezetbe rendeződik.

Az iparban anyag porlasztására és mintázat maratására használják. Továbbá szennyeződésként a VLSI áramkörök gyártásakor jelentkezik, ami a kihozatal csökkenését és a teljesítmény csökkenéséhez vezet.

1.2. A kísérlet

A kísérlet lebonyolítására egy speciális kamrára van szükség, amiben lehetséges a plazma létrehozása, a porrészecskék szórása illetve a megfelelő villamos tér létrehozása. A kamrának hermetikusan jól zártnak kell lennie, hogy csak a kívánt gázt tartalmazza. A középvákuumú működéshez kétlépéses vákuumszivattyút kell alkalmazni.

A kamra sematikus ábrája a 1.1. ábrán látható. A porrészecskék levitációjáért a villamos tér felelős. A síkba való zárás parabolikus potenciállal lehetséges. Az ilyen tér létrehozása a következő elektróda elrendezéssel lehetséges: alul elhelyezett korong alakú elektróda, ami felett egy gyűrű alakú elektróda helyezkedik el. Az ilyen elektródarendszerre kapcsolt váltakozó feszültség a beszórt porrészecskéket lebegtetni tudja. Továbbá a részecskék követéséhez lézerrel megvilágítjuk és nagysebességű kamerával felvételt készítünk róla.

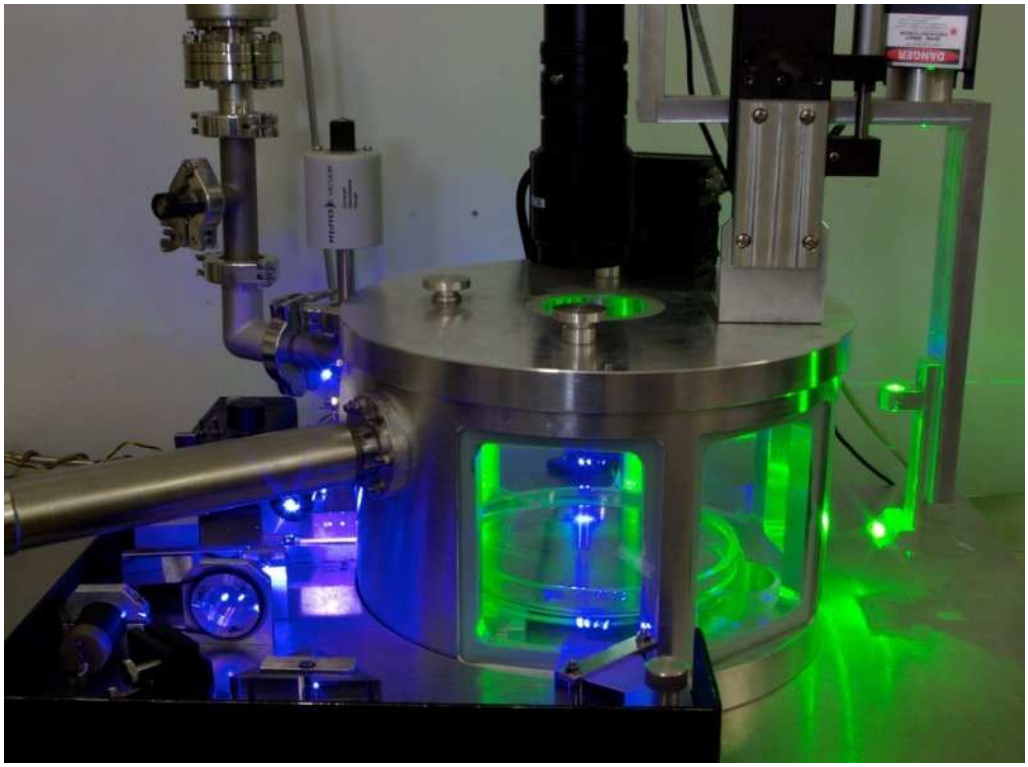


1.1. ábra. *A mérési elrendezés sematikus ábrája (forrás: [4])*

A kamrát (1.2. ábra) Hartmann Péter külső konzulensem építette és a MTA Wigner FK SZFI Gázkisülési Laboratóriumában található.
A paraméterei:

- Kamra belső átmérője: $25cm$
- Kamra belső magassága: $18cm$
- Alsó elektróda átmérője: $18cm$
- Felső gyűrű elektróda belső átmérője: $15cm$
- Felső elektróda távolsága az alsótól: $13cm$
- Argon gáz nyomása: $1.2 \pm 0.05Pa$
- Gáz átfolyása: $\sim 0.01CCPM$
- RF gerjesztés: $7W$ @ $13.56MHz$
- Porrészecske: melamine-formaldehyde
- Porrészecske átmérője: $4.38 \pm 0.06\mu m$
- Porrészecske tömege. $6.64 \cdot 10^{-14}kg$
- Látható porrészecskék száma: ~ 2500
- Megvilágító lézer: $200mW$ @ $532nm$
- Kamera: $1.4MPixel$ @ $100FPS$

A kamra alsó elektródáját lehetséges egy motorral forgásba hozni. Ezáltal a kamrában lévő gáz is forgásba jön, ami a porrészecskékre az F_i ionsodrási erővel hatva nagy mágneses tér Lorentz erejének feleltethető meg.



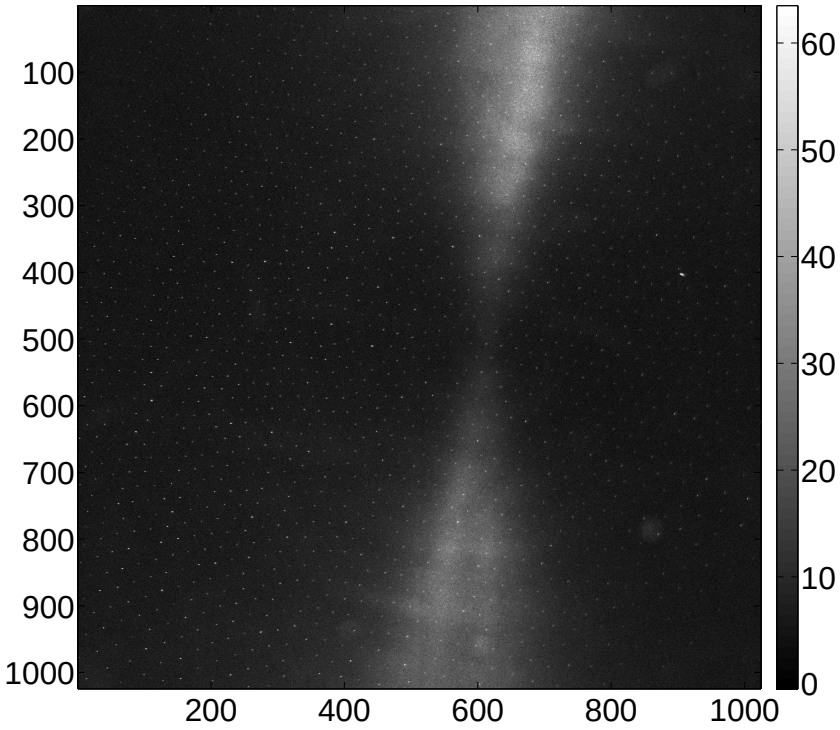
1.2. ábra. *A konkrét kamra működése közben*

1.3. A mérendő mennyiségek és a származtatott értékek

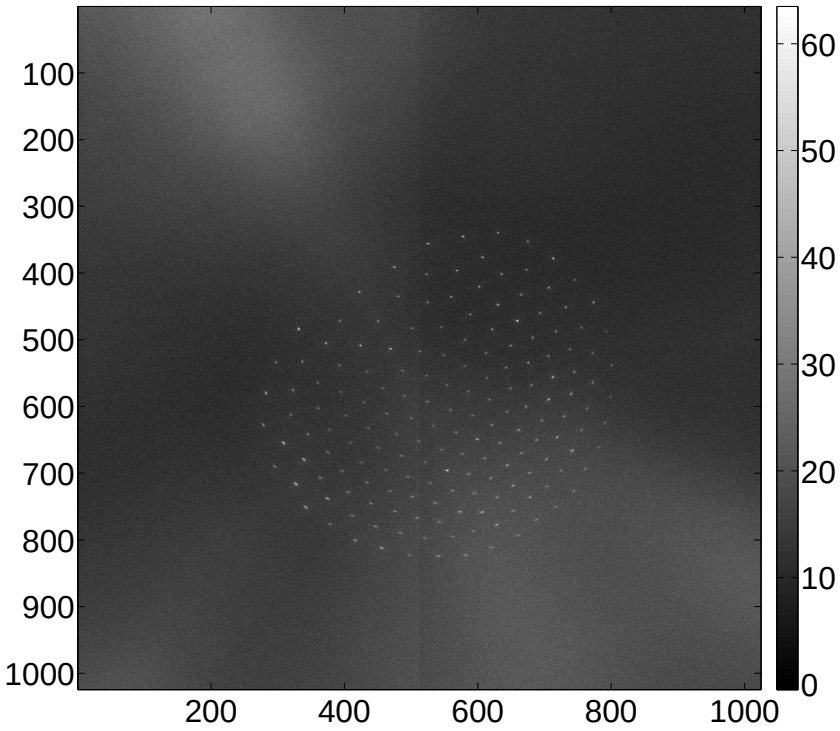
A kísérlet előkészítése a következő lépésekből áll:

1. Elővákum szivattyú bekapcsolása,
2. Középvákuum szivattyú bekapcsolása,
3. Argon palack megnyitása a megfelelő áramlás szintjére,
4. RF gerjesztés bekapcsolása,
5. Megvilágító lézer bekapcsolása,
6. Porrészecskék beszórása,
7. Kamera bekapcsolása és a megjelenítő szoftver futtatása,
8. Ha sok összetapadt porrészecske látható, illetve ha túl sok porrészecske látható, akkor az RF gerjesztés gyors ki-be kapcsolása után a 6.-től való folytatása a folyamatnak.

A 1.3. ábrán látható álló és forgó alsó elektródájú kísérlet során a porrészecskékről készült kép.



(a) Álló elektróda esetén sok (~ 1500) részecske



(b) Forgó elektróda esetén kevés (~ 100) részecske

1.3. ábra. *Két különböző kísérlet során készült fénykép.*

A fényképek alapján a részecskék pozíciójára és időbeli eloszlására vagyunk kíváncsiak. A kamera objektíve által leképezett kép a függőleges irányú pozíciót nem tartalmazza. Mivel jelen kísérletek során az ezirányú mozgása a részecskéknek nem számottevő, így az $x - y$ pozícióját jól lehet számítani a kép alapján. Nagy pontosság esetén szükséges a képek előzetes feldolgozása a perspektivikus torzítás kiküszöbölése végett. Természetesen ez lehetséges a pozíciók számítása után is, aminek a számításigénye kissebb is.

2. fejezet

A részecskék detektálása

2.1. Detektálási módszerek

A korábban látható 1.3. ábrán látható képeken kell a részecskéket felismerni és azoknak a koordinátáit exportálni. Erre több lehetőség adódik, amit a [5] részletez. A detektáló módszereket a számítás igényeiknek növekvő sorrendjében mutatom be.

2.1.1. Küszöb módszer

Legkézenfekvőbb módszer, hogy a kép pixeleinek világosságát összehasonlítjuk egy küszöb értékkel és ha az nagyobb ennél, akkor ezeket megjelöljük, mivel ott részecskét feltételezünk. A módszer egyenletes háttér-világosság esetén jól működik és extra gyorsan számítható.

2.1.2. Küszöb módszer szűréssel

A háttér világossága a 1.3a. ábrán jól láthatóan nem egyenletes. A korábban említett egyszerű küszöb módszer itt nem alkalmazható, mivel a világos és sötét területek más és más küszöbértéket kívánnának meg. A megoldása erre, mint megannyi villamosmérnöki mérési feladatra, hogy a jel helyett a differenciális jelet mérjük. Jelen esetben ez azt jelenti, hogy először előállítjuk a részecske nélküli háttérképét, majd a méréssel kapott képből kivonva ezt a differenciális képet megkapjuk. A differenciális képen a részecskéket a korábban említett küszöb módszerrel lehet detektálni a részecskéket.

Ehhez csupán a mérési képből kell szűréssel származtatni a háttérképet. A [5, 6] cikkekben és általában erre véges Gauss szűrőt használnak, ami egy lineáris véges impulzusválaszú (FIR) aluláteresztő szűrő. A szűrés egy adott pixel környezetének súlyozott átlagolását jelenti. A súlyozás során szoroznunk kell, ami a bináris megvalósítás végett jóval lassabban történik, mint az összeadás avagy az összeha-

sonlítás.

Mivel a részecskék mérete a képen véges és kis szórású, így a korábban említett FIR Gauss szűrő jól tudja a részecskéket eliminálni. Viszont a mérési képek 100 FPS sebességgel és 1024×1024 mérettel érkeznek be. Ez 100 MByte/s adatfolyamot jelent. Ahhoz, hogy ezt közel real-time feltudjuk dolgozni a Gauss szűrés nem járható út. Hatékonyabb szűrőre van szükségünk! A megoldást a medián szűrőben látom, ami egy nemlineáris, de véges „impulzusválaszú" szűrő. A szűrő az adott pixel környezetének mediánjának számítását jelenti. A nemlinearitás jelen esetben nem okoz gondot, mivel csak detektálásra használjuk (a pozíció kinyerése az eredeti kép alapján készül, de ez később részletezve lesz.)

A Gauss szűrő $n \times n$ környezet (ablak) esetén n^2 szorzást és $n^2 - 1$ összeadást jelent. Míg a medián szűrő $n \times n$ ablak esetén: bubborék rendezés során $O(n^2)$, javított bubborék rendezés során $O(n^2/2)$ illetve quicksort esetén $O(n \log n)$ összehasonlítást és cserét. Az összehasonlítás és a csere nagyságrendekkel gyorsabban végrehajtható, mint a szorzás és összeadás.

2.1.3. Adaptív küszöb módszer szűréssel

Látható a 1.3a. ábrán, hogy a sötétebb háttér kiterjedése nagyobb, ennek megfelelően az automata kamera ezekre a területekre állítja be az expozíciót. Így a sötét területek részletesebbek, míg a világos területek részletszegények (túlexponáltak) lesznek. Számunkra ez azt jelenti, hogy a sötétebb területeken jobban megbízhatunk a részecskék képében, míg a világosabb területeknél nem. Ezt a döntési küszöb értékének adaptációját jelenti a háttér világosságához. Az adaptív küszöböt a következő kifejezéssel határoztam meg:

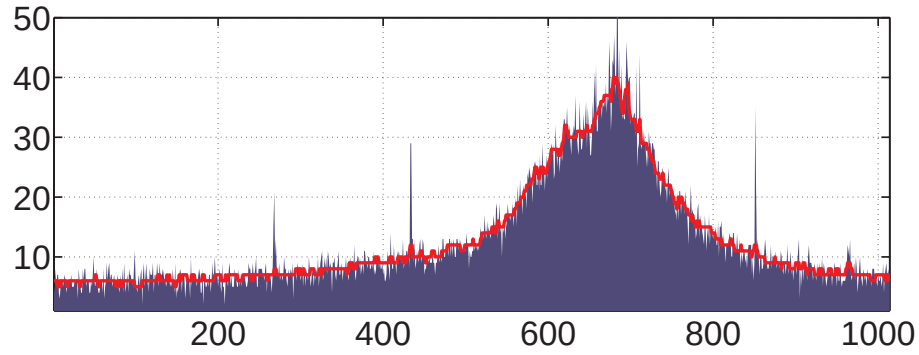
$$K = \mathrm{Mean}\{P - \hat{\mathrm{M}}P\} + \\ + \delta \cdot \mathrm{STD}\{P - \hat{\mathrm{M}}P\} \cdot \left[1 + a \left(\frac{\hat{\mathrm{M}}P}{\max\left\{\hat{\mathrm{M}}P\right\} - \min\left\{\hat{\mathrm{M}}P\right\}} \right)^b \right] \quad (2.1)$$

ahol P az eredeti kép, $\hat{\mathrm{M}}P$ a medián szűrt kép, Mean$\{\}$, STD$\{\}$ az átlag és a szórás számításának függvénye és a a, b két alakparaméter. [1]

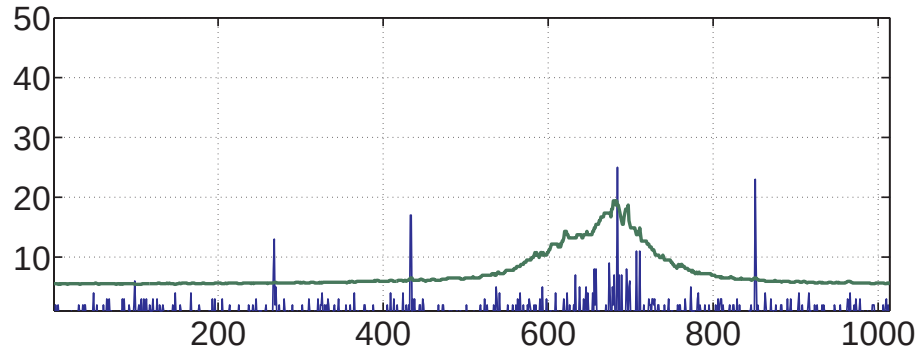
A 2.1. ábrán látható a detektálási algoritmus működése közben. A 2.1a. ábrán a mérési kép (P) látható kitöltött görbével, ami az 1.3a. ábrán látható mérési kép $x = 40$ sora. Piros görbével látható a medián szűrt jelet ($\hat{\mathrm{M}}P$), amin jól érzékelhető

[1] Az adaptív dontést a mérési kép előzetes feldolgozásával is elérhettük volna, ha a Photoshopból ismert Curves tool-al módosítottunk volna rajta. A tool a képet egy nemlineáris fügvényen keresztül engedi át.

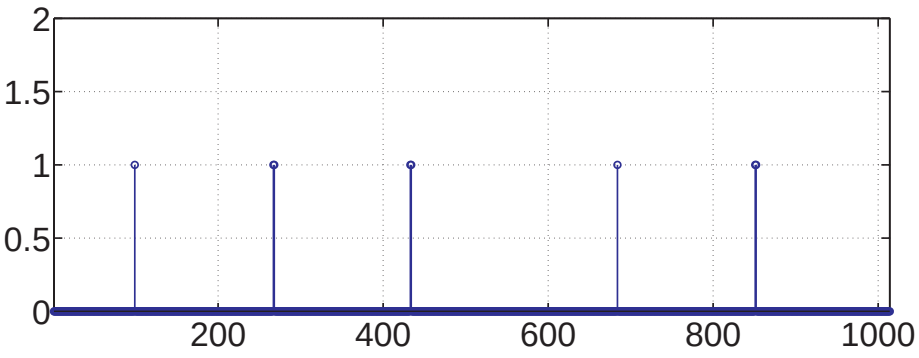
a hirtelen változások eliminációja. A következő 2.1b. ábrán az eredeti és a szűrt különbsége azaz a differenciális kép (∂P) látható a kék görbével. A zöld görbe a (2.1) szerinti döntési küszöb. Az utolsó 2.2b. ábrán a detektált részecskék figyelhetőek meg.



(a) Eredeti (kitöltött) és a medián szűrt (folytonos) jel



(b) A differenciális jel (kék) és a döntési küszöb (zöld)



(c) Detektált részecskék

2.1. ábra. *A medián szűrést alkalmazó adaptív küszöbbel részecskét detektáló algoritmus bemutatása az álló mérési kép (1.3a. ábra)* $x = 40$ *során.*

2.2. A részecskék pozíciójának számítása

Kis felbontású kamera illetve nagyon kis porrészecskék esetén előállhat, hogy a részecskék egy pixelnyi területet foglalnak el a képen. Detektálás szempontjából ez kedvező viszont a pozíciómérés szempontjából nem, mivel ilyenkor a felbontásunk 1 pixelnyi. Ezen javítani a dithereléssel a következőképpen lehet: picit elállítjuk az élességet úgy, hogy egy részecske több pixel nagyságú „maszat" legyen, majd a korábban részletezett detektálást elvégezzük.

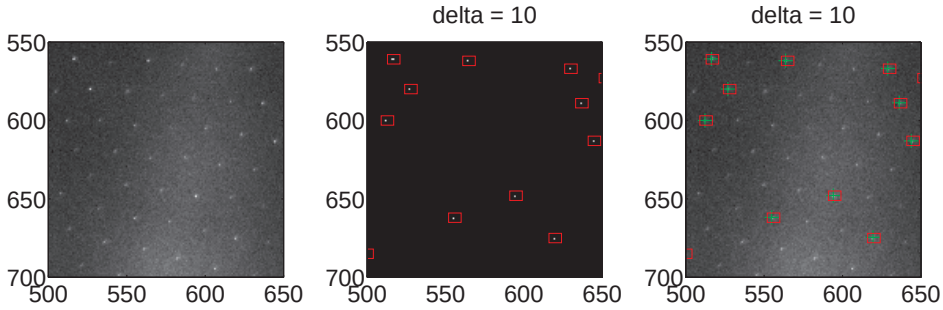
Ennek hatására egy részecske több pixelnyi felületet fog elfoglalni és a detektáló algiritmus is több pixelt fog megjelölni. A pozíció megtalálásához csoportosítani kell a megjelölt pixeleket. Az egy részecskéhez tartozó pixel-csoportot egy téglalap fogja határolni, amit region of interest-el (ROI) szokás illetni. Azonban a részecske detektálása után amorf formájú megjelölt pixeleink lesznek. Ezeket a könnyebb csoportosítás végett kiterjesztem pár pixellel az így kapott képet flood-fill algorimussal egybefüggővé teszem és ennek eredményeképp megkapom a ROI határoló koordinátáit, amit a következő két módszer felhasznál a részecske pozíciójának számítása során.

Maximum keresés

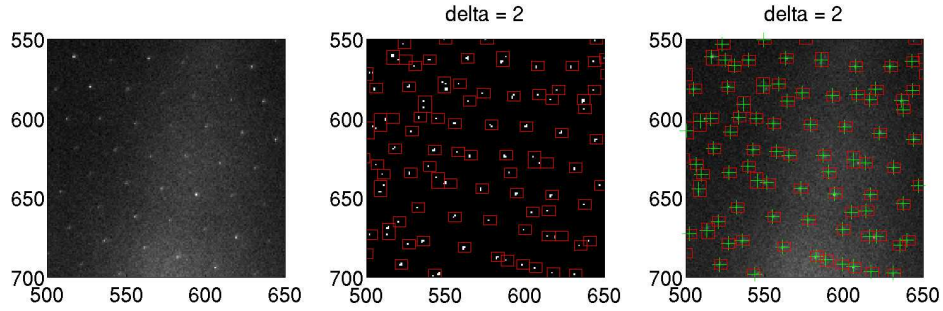
Legegyszerűbb eset, ha a ROI-n belül az eredeti pixelek közül a legvilágosabbat veszem a részecske pozíciójaként.

Szubpixel felbontás momentum módszerrel

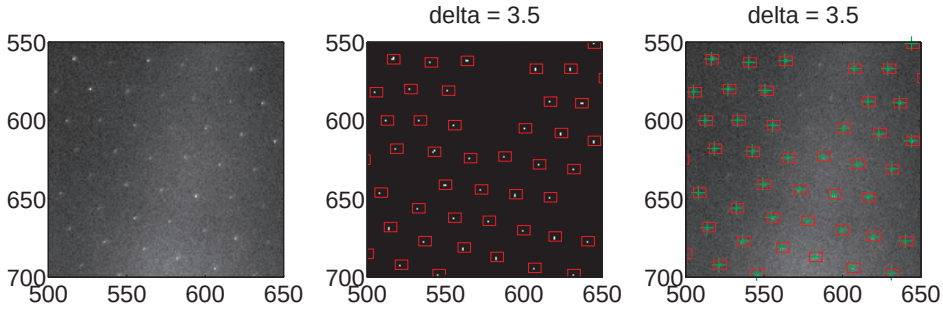
Szofisztikáltabb, ha a ROI-n belül az eredeti pixelek világosságát, mint tömegpont tömegének veszem és a ROI-által határolt test súlypontját megkeresem. A módszerre kritikusan hat az előbb említett kiterjesztés mértéke. Ha túl nagy a kiterjesztés, akkor azzal hibát viszek be pozicíó mérésébe, míg ha túl kicsi akkor meg egy részecskének akár két képe/pozíciója keletkezhet.
Az algoritmus hatékonyságát különböző δ érték mellett a következő 2.2. ábrán látható.



(a) $\delta = 10$



(b) $\delta = 2$



(c) Optimális $\delta = 3.5$

2.2. ábra. *Az adaptív küszöb módszerrel detektált részecskék momentum módszerrel számított pozíciója.* ***Bal oszlopban*** *az eredeti mérési kép,* ***középső oszlopban*** *a detektálás eredménye és a ROI, az* ***utolsó oszlopban*** *az eredeti mérési képen a ROI és a detektált részecske pozícióját jelző kereszt.*

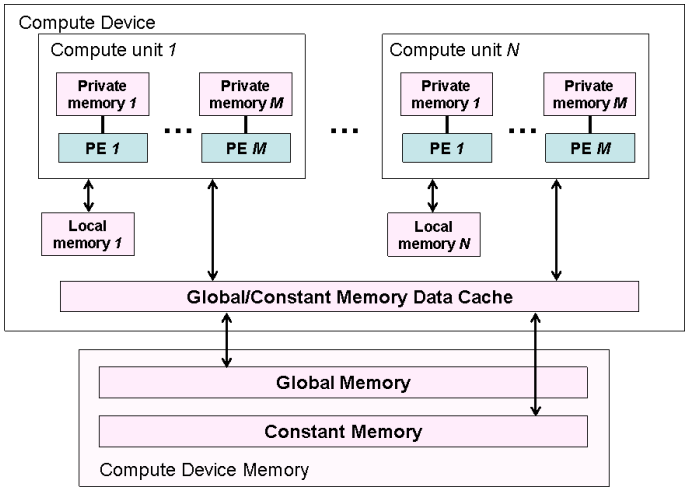
Az algoritmus jól párhuzamosítható, ami a nagyteljesítményű multiprocesszoros környezetben kedvező futási időt eredményezhet. A párhuzamos program létrehozásának segítségére az OpenCL keretrendszert választottam, aminek a bemutatása következik.

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, FPGA, DSP, ...). A hárdvergyártók implementálják az OpenCL szabványt, ami által saját platformot hoznak létre. Egy ilyen platformon belüli eszközök alatt főként GPU-kat, de CPU-kat és FPGA-t ... is értünk. OpenCL keretrendszerben történő programozás során két programot kell írnunk. Az egyik a kernel, ami az eszközön 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 argumentumok beállítását illetve a kernel meghívását az eszközön. A kernel futása végeztével a host-program kiolvassa az eszközből a kívánt eredményt.



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

Az eszközök multiprocesszoros architektúrával és ezek kiszolgálására képes memória architektúrával rendelkeznek, amit a 3.1 ábra vázol. Egy eszköz több compute unit-ot (processzor-magot) tartalmaz. Az OpenCL négy memória szintet különböztet meg, amikre a következőképpen hivatkozik:

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

A regiszterek és lokális memória kis méretűnek és gyors elérésűnek mondható, míg a globális memória nagynak, de lassú elérésűnek. A memóriákra megkötésként szolgál, hogy ki allokálhat, írhat és olvashat belőle. A 3.1. táblázatban látható ezen jogosultságok.

3.1. táblázat. *OpenCL memória szintek*

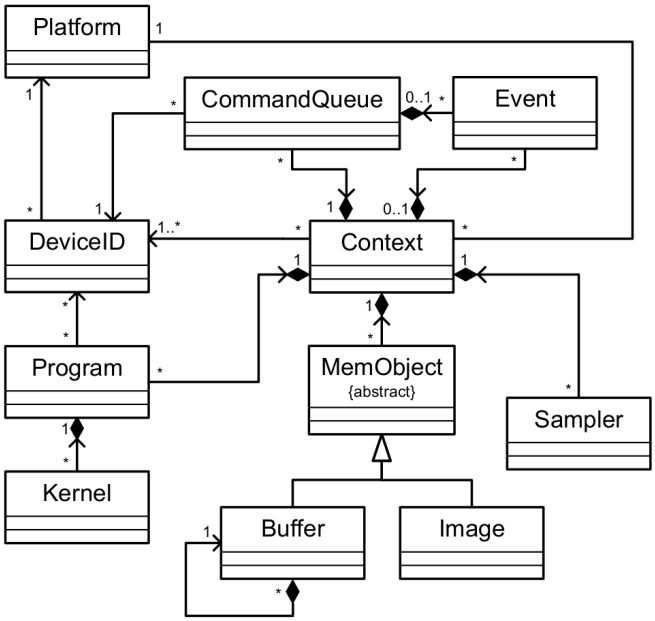
	Global memory	Constant mem.	Local mem.	Private mem.
Host	Dinamikusan R/W	Din. R/W	Din. R/W	
Kernel	R/W	Statikusan R	Satik. R/W	Statik. R/W
Sebesség	Lassú	Gyors	Gyors	Regiszter
Méret	1 Gbyte $<$	$\sim$ 64 Kbyte	$\sim$ 16 Kbyte	$<$ 1 Kbyte

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 processzormaghoz.
- *Mikor?* Mivel a memória művelet alatt a futtatott kernel nem dolgozik, így átadja a helyét egy másiknak. (Ez Direct Memory Access (DMA) blokk létezése allatt igaz). Ennek a megfelelő szinkronizációjával nagyobb kihasználtság érhető el (load balance).

3.2. OpenCL programozási modell

A programozási modell középpontjában a kontextus áll, ami az OpenCL osztálydiagrammján (3.2. ábra) figyelhető meg. 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



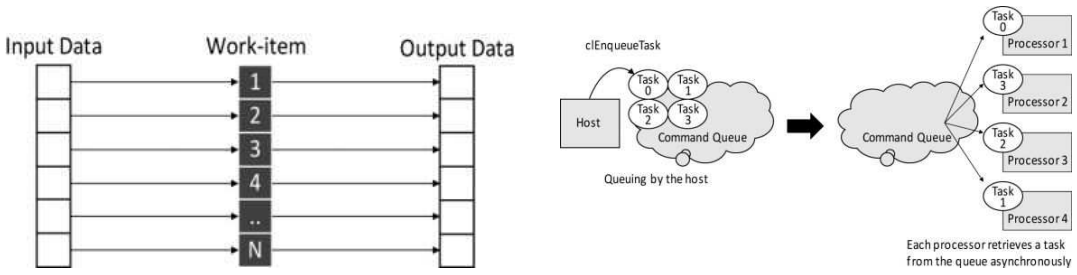
3.2. ábra. *OpenCL context osztálydiagrammja (forrás: [7])*

belüli eszközök programozhatóak heterogén módon. Például: Intel platform esetén lehetséges CPU-t, processzorkártyát és Intel-es GPU-t programozni.

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.



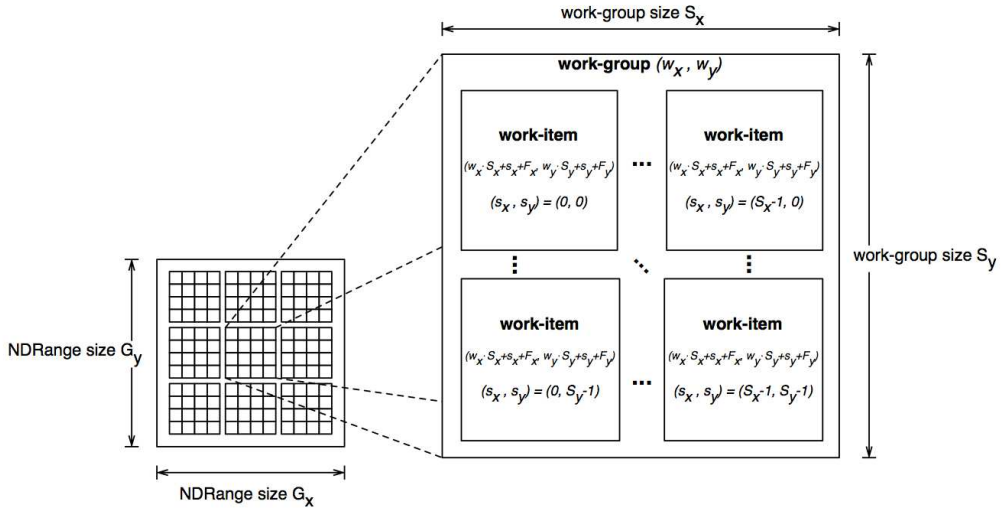
(a) Adat parallel **(b)** Taszk parallel

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

A processzor-magok megfelelő kihasználtságának elérése végett több ezer work-item virtuálisan osztozik rajta. Továbbá ezen work-item-eket work-group-okba ren-

dezzük.

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.



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

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. Az alkalmazott szinkronizáció egy korlátot tesz a programban, amit csak akkor léphet át, ha az összes többi work-item az azonos work-group-ban ezta 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 szinkronizációra nincs lehetőség.

Összefoglalva: nagy hangsúlyt kell a memóriaszervezésre fordítani, hogy a processzormagok megfelelően legyenek az adatokkal táplálva.

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	1265
MAX WORK GROUP_SIZE	8192	512
GLOBAL MEM SIZE	$\sim 4Gbyte$	$1Gbyte$
LOCAL MEM SIZE	$32Kbyte$	$16Kbyte$

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

4. fejezet

Fejlesztőkörnyezet összeállítása

OpenCL kód fejlesztése történhet Windows alatt NVIDIA Nsight Visual Studio Edition [8] és Linux alatt GCC-vel [9]. Az Open Source fejlesztőrendszer ingyenessége és az általa generált program hordozhatósága végett a Linux alatti fejlesztés mellett döntöttem. Az OpenCL-t támogató hardverek legtöbbször CPU-k, GPU-k és az Intel MIC [10] kártyái. Ezekre való OpenCL kód fejlesztéséhez a gyártók biztosítanak Software Developement Kit-et (SDK). Ezek telepítése szinte bármelyik Linux disztribúción sikerülhet a megfelelő követelmények előzetes telepítése után. A Linux diszrók közül a CentOS-re [11] esett a választás, ami csupán a fejlesztőkörnyezet egyszerűbb telepítése végett történt így.

4.1. Software Developement Kit-ek (SDK) telepítése

4.1.1. nVidia támogatás telepítése

A legtöbb mai Linux disztrók tartalmaznak drivert az nVidia videó kártyákhoz. Ez az open source Nouveau, ami még nem támogatja az OpenCL-t. Így a hivatalos nVidia drivert fel kell telepítenünk. Ehhez először le kell tiltanunk a Nouveau betöltését. Ezt két helyen is meg kell tennünk: egyrészt a `/etc/modprobe.d/blacklist.conf` fájlhoz hozzá kell adnunk a következő sort:

```
blacklist nouveau
```

majd újragenerálni az INITial RAM File System-et (initramfs), ami a rendszer inicializásáért felelős:

```
$ mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).img.bak
$ dracut -v /boot/initramfs-$(uname -r).img $(uname -r)
```

másrészt a rendszer indító GRand Unified Bootloader-ben (GRUB) is le kell tiltani a betöltését a kernel opció alábbi paranccsal való kiegészítésével:

```
nouveau.modeset=0
```

Továbbá a telepítéshez szükséges követelményeket a következő parancsokkal telepíthetjük:

```
$ yum groupinstall "Development Tools"
$ yum install kernel-devel kernel-headers dkms
```

Ekkor a rendszer újraindítása után készen állunk a hivatalos nVidia driver telepítésére. A drivert a következő linken lehet letölteni [12]. A grafikus felületet a telepítés idejére le kell állítani az X grafikus kiszolgálót

```
$ init 3
```

paranccsal, majd a konzolban telepíthető a driver, ami a legtöbb munkát elvégzi helyettünk. Ezután az

```
$ init 5
```

paranccsal áttérhetünk a grafikus felületre, ahol a megfelelő környezeti változókat kiegészíthetjük. Legcélratörőbb, ha a `~/.bashrc` fájlt módosítjuk és hozzáadjuk a következő sorokat:

```
PATH=$PATH:$HOME/bin:/usr/local/cuda/bin
export PATH

CUDA_INSTALL_PATH=/usr/local/cuda
export CUDA_INSTALL_PATH

LD_LIBRARY_PATH=/usr/local/cuda/lib64:/opt/intel/opencl/bin
export LD_LIBRARY_PATH

NVSDKCOMPUTE_ROOT=/usr/local/cuda/lib64
export NVSDKCOMPUTE_ROOT

INTELOCLSDKROOT=/opt/intel/opencl
export INTELOCLSDKROOT
```

Mivel az nVidia limitálja a kernel futási időt 5 másodpercben limitálja, hosszabb kernel futási idő esetén a rendszer lefagy. Ezt a korlátozást a `/etc/X11/xorg.conf` fájl `Device` részének a következővel való kiegészítésével érhetjük el:

```
Option "Interactive" "boolean"
```

Érvényre juttatásához az X újraindítása szükséges (CTRL+ALT+Backspace). Ezután nagyobb problémák esetén már nem fogja lefagyasztani a rendszert a watchdog.

4.1.2. Intel támogatás telepítése

A következő oldalról letölthetjük az SDK-t [13]. A kicsomagolás után az `./install-cpu.sh` program futtatásával telepíthető. Ezután még szükséges a `LD_LIBRARY_PATH` beállítása.

4.1.3. Eclipse – Integrated Developement Environment

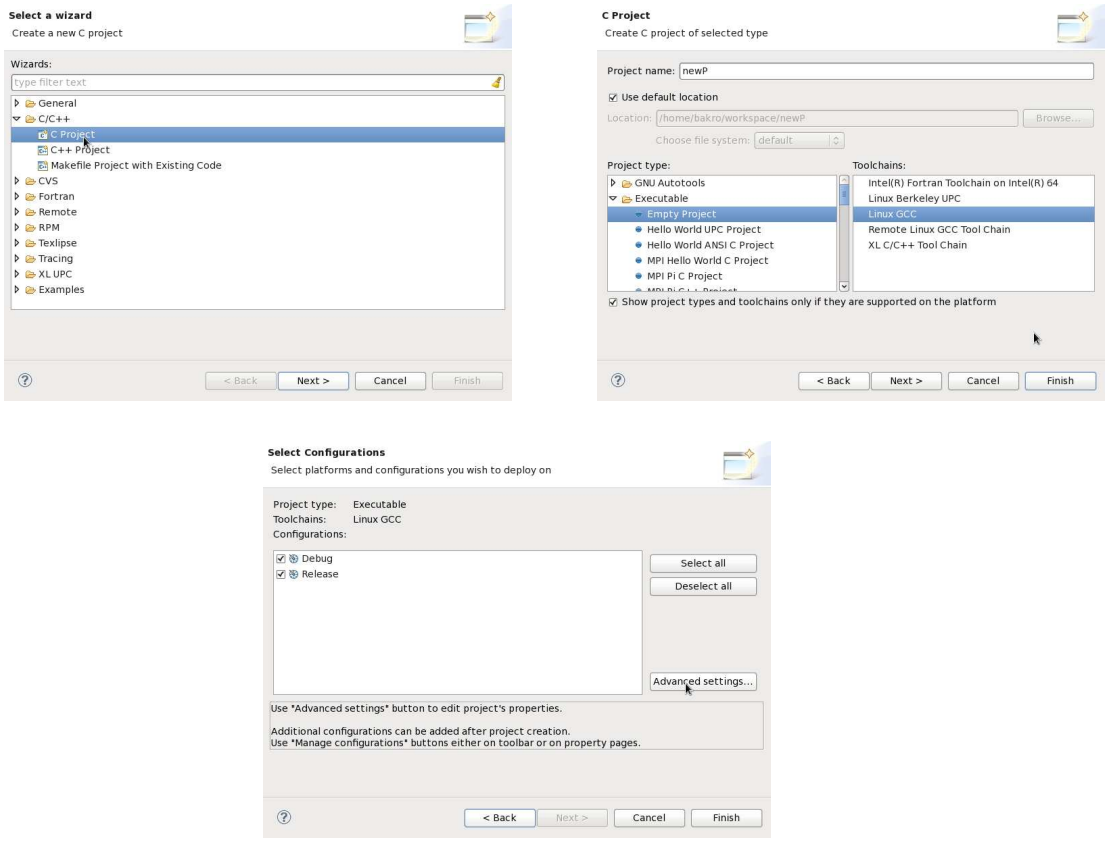
A fejlesztés és hibakeresés egy Integrated Developement Enviroment (IDE) segítségével könnyebb. Az open source Eclipse [14] fejlesztőkörnyezet a különböző pluginjaival épp megfelelő erre a célra. Például a C-nyelv fejlesztését segítő C/C++ Developement Tooling (CDT), a verziókövetést menedzselő EGit és a hibakeresést támogató GDT. A sok Eclipse változat közül az OpenCL fejlesztéshez legjobban az Eclipse for Parallel Application Developers verzió illik, mivel a korábban említett pluginokat már eleve tartalmazza.

4.2. Új (Hello World) projekt létrehozása

Az OpenCL fejlesztését konyhanyelven bemutató OpenCL Programming Guide [15] könyvben szereplő Hello World programot a következő linken lehet letölteni [16]. A kód fordítása elött egy Eclipse projektet létrehozunk és a fordításhoz szükséges beállításokat elvégezzük.

4.2.1. Empty C project létrehozása

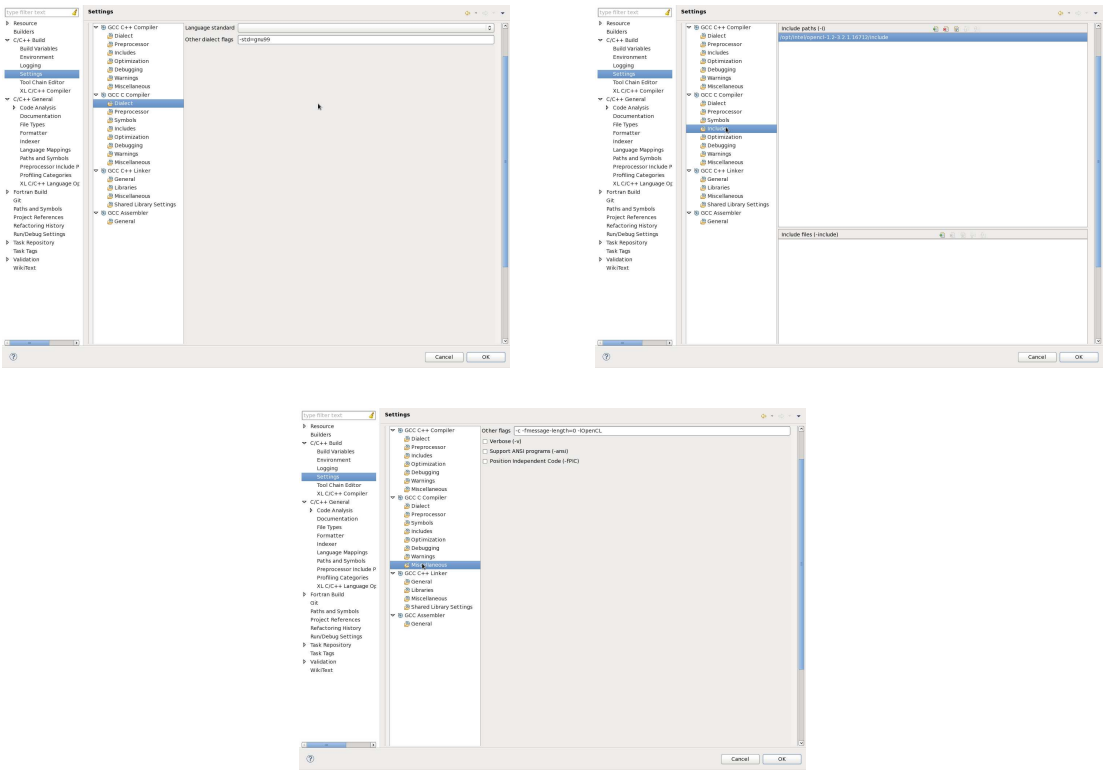
Először egy üres C projektet hozunk létre, ami folyamatát a 4.1 ábrán látjuk. A korábban említettek szerint fordítónak a Linux GCC-t állítjuk be.



4.1. ábra. *Új Eclipse projekt létrehozása*

4.2.2. Compiler beállítása

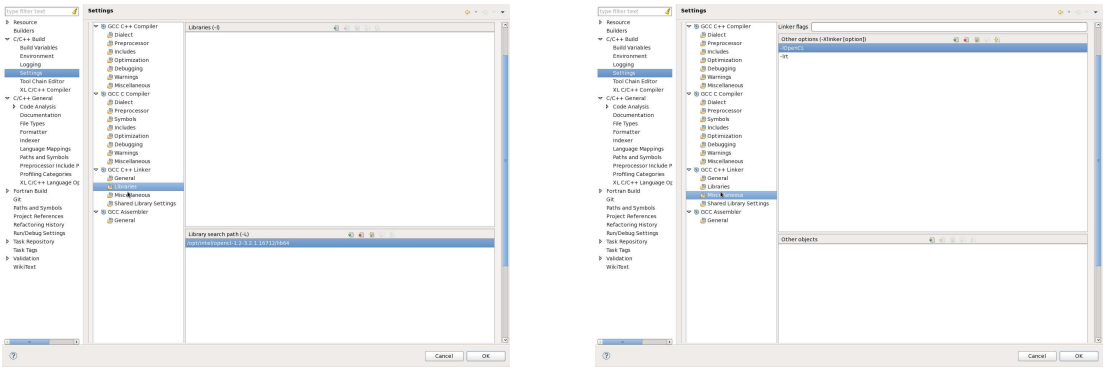
A létrehozott projektre jobb gombbal kattíntva a tulajdonságára kattintva állíthatjuk be a fordítót a képnek 4.2 megfelelően. A beállítások kiterjednek a GNU-C99 nyelv szerinti fordításra és a korábbi 4.1 részben telepített SDK-ban található include mappa beállítására.



4.2. ábra. *Compiler beállításai*

4.2.3. Linker beállítása

A linkert a 4.3 ábra szerint állítjuk be.



4.3. ábra. *Linker beállításai*

5. fejezet

Multiprocesszoros program

5.1. A program lépéseinek bemutatása

A 100 FPS-el érkező képeket **tizesével** dolgozom fel, mivel a megjelenítésnek nem fontos real-time működésűnek lennie. A program ciklikusan a következő lépéseket hajtja végre:

1. Képek beolvasása a host-memóriába,
2. Képek leküldése a host-memóriájábó az eszköz globális memóriájába,
3. Eszközön futtatandó kernel inicializálása, argumentumainak beállítása,
4. Kernel futtatása az eszközön,
5. Kernel futása után az eredmény az eszköz globális memóriájából a host-memóriájába való visszatöltése,
6. Posztprocesszálás,

A kernel megírása során a korábbi 3. részben említetteket figyelembe kell venni. Főként a véges lokális és globális memóriát. A kernel adat-parallel módon lett megírva. A lokális memória 16KByte méretű, amit két 8KByte nagyságú A és B bufferre osztottam fel.
A kernel program lépései a következők:

1. A work-item globális és lokális indexének meghatározása,
2. Medián szűrés:
   (a) A kép egy részének a globális memóriából a lokális A bufferbe való másolása,
   (b) Az összes work-item másolási folyamatának megvárása,
   (c) Medián szűrés az A bufferből a B bufferba,
   (d) Az A bufferba az eredeti (A buffer) és a szűrt (B buffer) különbségének az eredményét (differenciális kép) elhelyezni,
   (e) Döntési szint számítása és detektálás/megjelölés a B bufferba,

(f) A B bufferben lévő eredmény a globális memóriába való kiírása hibakeresés biztosítása végett.

3. Kiterjesztés és a flood-fill algoritmussal a ROI meghatározása:
   (a) Megjelölt pixel keresése,
   (b) Adott környezetére való kiterjesztése,
   (c) A kiterjesztés során a két legtávolabbi pont lesz a ROI határpontjai.
4. Részecske pozíciójának számítása momentum módszerrel,
5. Eredmény mentése a globális memóriába

A kernel lefutása után az eszköz globális memóriájából az eredményeket a hoszt-memóriájába töltjük. A számításigényes szűrés, detektálás és momentum számítás az eszközön hajtódott végre. A részecskék pozíciójának eloszlásának számítása memóriaigényes, de nem számításigényes feladat, így az a host-programban került megvalósításra.

6. fejezet

Összehasonlítás

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 3× 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

Dolgozatomban bemutattam a porosplazma kísérletek apparátusát. A kísérlet során a kristályrácsba rendeződő részecskékről egy nagysebességű kamerával fényképek készülnek. A dolgozatomban ezen képeket, kellett feldolgoznom és a részecskék pozícióját detektálnom. A pozíciók a fizikai modell/szimuláció validálására szolgálnak.

Ismertettem a részecske detektálásának módszerét szűrés és adaptív döntési küszöb használatával. Az elterjedt FIR Gauss szűrő helyett a hatékonyabb medián szűrőt javasoltam és alkalmaztam. A pozíció számítására a momentum módszert implementáltam, ami nagyobb számítási energiát igényel, de szubpixeles felbontást tudtam vele elérni. Konstatáltam, hogy az így kialakult program masszívan párhuzamosítható.

Ezután áttekintettem az OpenCL keretrendszert, amit a párhuzamos program megírásának segítségére használtam. Az itt ismertetett megállapításokat figyelembe véve állítottam össze a párhuzamos program lépéseit, amit részleteztem is.

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 host-program real-time mérésbe helyezése egy producer-consumer sémájú szál megoldás alkalmazásával,
- Az eredmény grafikus megjelenítése pl.: OpenGL használtatával,
- Az OpenCL szabvány által specifikált vektor műveletek támogatásának kiaknázása, ami az Intel Xeon PHI processzorkártyában rejlő teljesítményt ki tudná aknázni.

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

[1] P. Hartmann, A. Douglass, J. C. Reyes, L. S. Matthews, T. W. Hyde, A. Kovács, and Z. Donkó, „Crystallization dynamics of a single layer complex plasma," *Phys. Rev. Lett.*, vol. 105, p. 115004, Sep 2010.

[2] P. Hartmann, Z. Donkó, T. Ott, H. Kählert, and M. Bonitz, „Magnetoplasmons in rotating dusty plasmas," *Phys. Rev. Lett.*, vol. 111, p. 155002, Oct 2013.

[3] D. Z. Hartmann P, Donkó I, „Single exposure three-dimensional imaging of dusty plasma clusters," *Rev. Sci. Instrum.*, vol. 84, p. 023501, 2013.

[4] R. L. Merlino, „Dusty plasmas and applications in space and industry," *Plasma Physics Applied*, vol. 81, pp. 73–110, 2006.

[5] Y. Feng, J. Goree, and B. Liu, „Accurate particle position measurement from images," *Review of Scientific Instruments*, vol. 78, no. 5, pp. –, 2007.

[6] *Tracking interacting dust: comparison of tracking and state estimation techniques for dusty plasmas*, vol. 7698, 2010.

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

[8] „Nvidia nsight visual studio edition." `https://developer.nvidia.com/nvidia-nsight-visu`

[9] „Gcc, the gnu compiler collection - gnu project - free software foundation (fsf)." `http://gcc.gnu.org/`.

[10] „Intel® many integrated core architecture (intel® mic architecture)." `http://www.intel.com/content/www/us/en/architecture-and-technology/many-integ`

[11] „Centos project." http://www.centos.org/.

[12] „Nvidia drivers." `http://www.nvidia.com/Download/index.aspx`.

[13] „Intel opencl sdk." `https://software.intel.com/en-us/vcsource/tools/opencl-sdk-xe`

[14] „Eclipse - the eclipse foundation open source community website..” https://www.eclipse.org.

[15] A. Munshi, *OpenCL Programming Guide*. Addison-Wesley, 2011.

[16] „Opencl programming guide - examples.” https://code.google.com/p/opencl-book-sampl