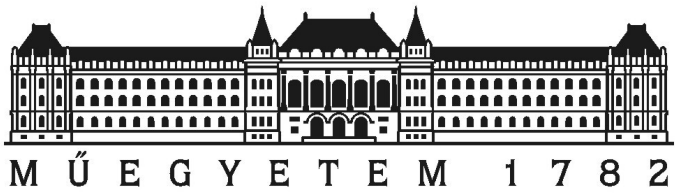


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

GPUval támogatott passzív radar feldolgozási algoritmusok optimalizálása

TDK dolgozat

Tomka Benedek Donát

2021

Tartalomjegyzék

Kivonat

A radar-rendszerek között megkülönböztetünk aktív és passzív megoldásokat. Utóbbi elrendezések kisugárzott mérőjel nélkül képesek követni céltárgyakat, pusztán a külső, egyébként is jelenlévő EM sugárzás vizsgálatával.

Pratikusan érdemes a nagy lefedettségű FM rádiók vagy DVB-T (földfelszíni TV) adások jeleit használni a megfigyelésre. Előbbit kedvező lefedettsége és nagy hatótávolsága, utóbbit jó felbontása és korrelációs karakterisztikája miatt. Erősen kutatott téma a 2.4GHz-es WiFi sugárzók alkalmazása passzív radaros felhasználásra.

Ezen rendszerek működési elve régóta ismert, viszont sokáig nem volt alkalmazható terepi felhasználásra a nagy jelfeldolgozási igény miatt. Míg az aktív rendszerek pontosan ismerik a megvilágító jel karakterisztikáját, és optimalizációkkal élhetnek a jellegét illetően, addig a passzív radaroknál elemi műveletszámban rendkívül magas korrelációkat - FFT-ket kell számolni. A mai technológiával már relatíve kis méret és költség mellett alacsony teljesítményigénnyel megoldható a jelfeldolgozási probléma.

A magas sebességű jelfeldolgozásra célműszerekkel, célhardverrel eddig is lehetőség volt. Mára az általánosan elérhető, középkategóriás – akár notebook, mobileszköz felhasználásra szánt – videokártyák is képesek lehetnek a valós idejű jelfeldolgozásra, így kézenfekvő az algoritmusok átírása és optimalizálása a videokártyás környezetre. További lehetőség még az eljárások FPGA-ra vagy ASIC-ra ültetése, melyre a munkám során használt környezet lehetőséget ad.

A masszívan párhuzamos, és főleg a GPU programozás sajátosságai miatt azonban egyes részfeladatok sebessége akár vissza is eshet - ezek vizsgálata és optimalizációja külön odafigyelést, és részletes verifikációt igényel.

A dolgozat fő témája az a passzív radar jelfeldolgozási és megjelenítési lánc melyet OpenCL futtatókörnyezeten általánosan elérhető GPU-ra fejlesztek, mely hatékonyan képes leváltani a CPU alapú megoldást, ezzel tényleges valósidejű működést tesz lehetővé.

Bemutatom a működés elvét, a jelfeldolgozási lépéseket, azok megvalósítását és teljesítményét. Kitekintek a továbbfejlesztési lehetőségekre és további optimalizációs irányokra.

Abstract

There are two distinct categories of RADAR systems: active and passive types. The mentioned passive radar systems can detect and track targets without any emitted RF, only based on third-party transmissions which are present in the environment.

It is practical to use highly developed FM radio, or DVB-T (terrestrial wireless television system) signals as illumination sources, the first for its range, the former for its spectacular resolution and correlation characteristics. 2.4G systems like WiFi are also under ongoing research.

The operating principle of these solutions is old, but it was not feasible or practical to run such a system since it demands great processing power. While active systems know the exact illumination signal and can optimize specifically, passive systems must use very high atomic instruction count operators like FFT or correlation. But today's technology opens the way for small, inexpensive, and low-power applications to come.

Processing was possible for some years with application-specific devices and hardware. But now commercial and common mid-range (even notebook or mobile) GPU-s could be powerful enough to process the data in real-time so it is advantageous to port and optimize the algorithms for this platform. The possibility remains to move the processing to an FPGA, and the used development environment has a way to do that.

The specifics of massively parallel and GPU programming might slow down some types of steps in the process. Dealing with optimizing and verifying those remains a goal for now.

The main topic of this paper is my work on a passive radar signal processing and visualization chain which is based on OpenCL and runs on common GPU hardware. It will replace the current CPU-based solution and will be greater in speed and efficiency - enabling actual real-time use and tuning.

The paper discusses the operation principle, the signal processing steps, their implementation, and performance verification. Further developments and possibilities are discussed.

Bevezető

Ellentétben az aktív radarrendszerekkel, melyek jól definiált saját megvilágító jelet sugároznak a céltárgy megfigyeléséhez, a passzív radarok külső források sugárzását használják a céltárgyak felderítésére.
A konvencionális aktív radarok adó- és vevő rendszerből állnak. Így a jelfeldolgozó rész pontosan ismeri a kisugárzott jel időzítését, és így a reflektált jel beérkezésekor az eltelt idő egyszerűen számítható. Ezzel szemben a passzív radarok először veszik az adóról származó direkt jelet, majd annak reflexióit, és bonyolultabb jelfeldolgozási műveletekkel határozzák meg az időeltolódásból a távolságot. Antennasorok használatával a beérkezési fáziskülönbségekből számítható a visszaverődés forrásának direkciója is, így a vevőhöz képest vett relatív pozíciója.
A céltárgy rendelkezhet sebességgel, ekkor az így visszaverődő jel Doppler csúszást szenved, mely szintén matematikailag kezelhető – lehetőséget ad sebességmérésre. Előnyös tulajdonsága tehát a passzív radarrendszereknek az egyszerűbb felépítés, kisebb térfogat és az alacsonyabb költség. Viszont az említett jelfeldolgozási lépések számításigénye rendkívül magas, így fontos a hatékony és gyors számítási módszerek kutatása.

A *Szélessávú Hírközlés és Villamosságtan Tanszék Mikrohullámú Távérzékelés Laboratórium*ában Dr. Dudás Levente és mások fejlesztéseként elkészült egy jelfeldolgozási lánc, mely FM, DVB-T passzív radarok nyers jelén elvégzi a teljes műveletsort, és térképen ábrázolja a követett céltárgyakat.
Ez a megoldás egy felső-középkategóriás többmagos AMD Ryzen processzoron futva bár működőképes, de nem valós idejű jelfeldolgozást valósít meg: a processzor számítási sebessége kihasználva a rendelkezésre álló optimalizálási lehetőségeket egyszerűen kevés a nagyméretű adatblokkokhoz.
Az elvégezendő számítások jellegéből adódik, hogy jelentős részük masszívan párhuzamosítható, hasonló lépés, így érdemes minél több processzormaggal rendelkező egységen párhuzamosan futtatni . Míg az extrém magas magszámú processzorok elérhetősége korlátozott (és igen magas költséggel jár beszerzésük és kompatibilis perifériáik), a hétköznapi videokártyákban lehetőség van gyors, rendkívül párhuzamos futtatásra.

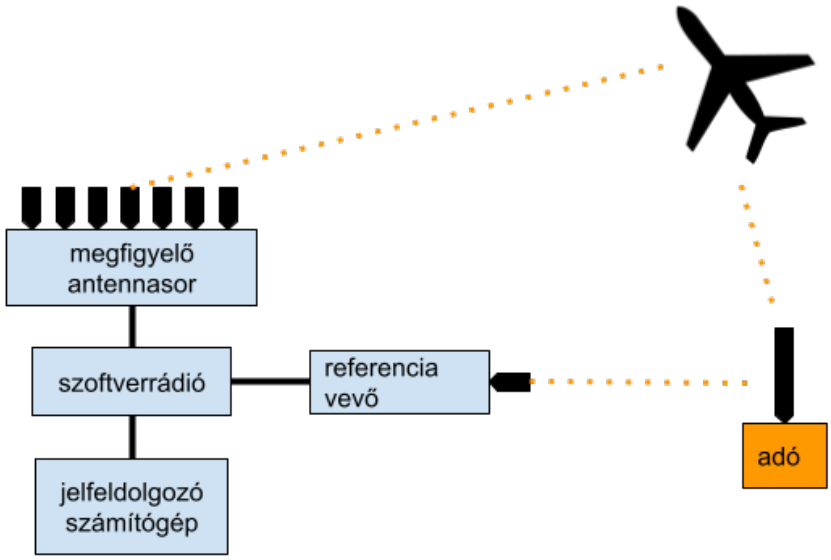
Munkámban arra adok megoldást, hogy ez a lánc valós idejűvé változzon a rendszer és az algoritmusok GPU alapú, lehető legpárhuzamosabb futtatási módra portolásával. Dolgozatomban bemutatom a passzív radar működését, az elkészült megoldást, valamint annak validációs és teljesítmény eredményeit. Továbbá kitekintek a további fejlesztési lehetőségekre és szoftvereszközökre.

Az első fejezetben bemutatom fejlesztéshez használt passzív (-bisztatikus) többcsatornás radarrendszer elvi működését. A második fejezetben ismertetem az általam fejlesztett OpenCL alapú feldolgozási láncot, a tervezési döntéseket. A harmadik fejezetben ismertetem a teljesítménymérések eredményeit és a validációs teszteket. Az utolsó fejezetben összegzem a munka eredményeit és felvázolom a továbbhaladási lehetőségeket.

1. fejezet

Passzív radar

A fejezetben bemutatom a projekt során használt radar eszköz felépítését, és elvi működését. Az 1.6. ábra szemlélteti a rendszer blokkvázlatát, főbb elemeit.



1.1. ábra. A passzív radar blokkvázlata, céltárggyal

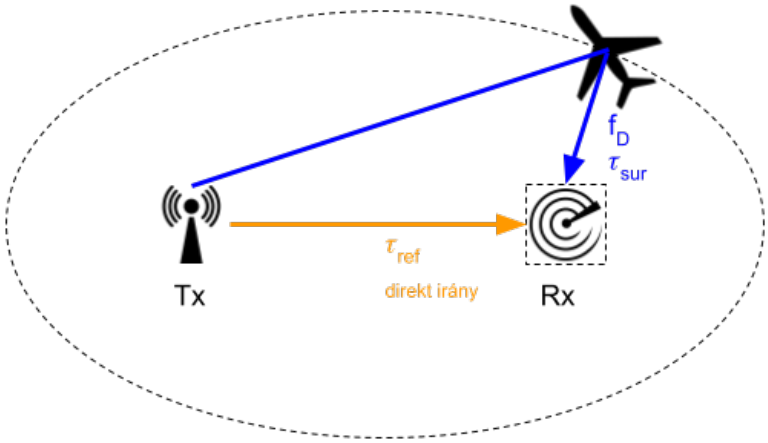
1.1. A bisztatikus radarozás

A *RADAR* (radio detection and ranging) egy olyan eszköz, mely rádióhullámok segítségével képes detektálni és távolságot mérni.[1] Ez a hagyományos (aktív) rendszerekben úgy történik, hogy az eszköz sugároz egy megvilágító jelet, majd várja a visszaverődést. A technológia hajnalán az adó és a vevő készülék nem került integrálásra egyetlen egységben, rendszerint térben izoláltan volt telepítve, mivel ez a megvalósítás egyszerűbben kezelhető elektronikai szempontból. Ezeket a megoldásokat nevezzük bisztatikus radarnak.[2]

A projektben felhasznált, ismertetett passzív radar rendszer egyfajta bisztatikus radarnak tekinthető, ahol az adóállomást nem a megfigyelő fél üzemelteti.

A geometriai elhelyezésből adódóan más módon juthatunk el a megfigyelő számára érdekes pozíció és sebesség értékekhez, mint az integrált (monosztatikus) esetben, de a számítási feladat egyértelműen megoldható.

A következő 1.2. ábra szemlélteti a radar rendszer térbeli elrendezését:

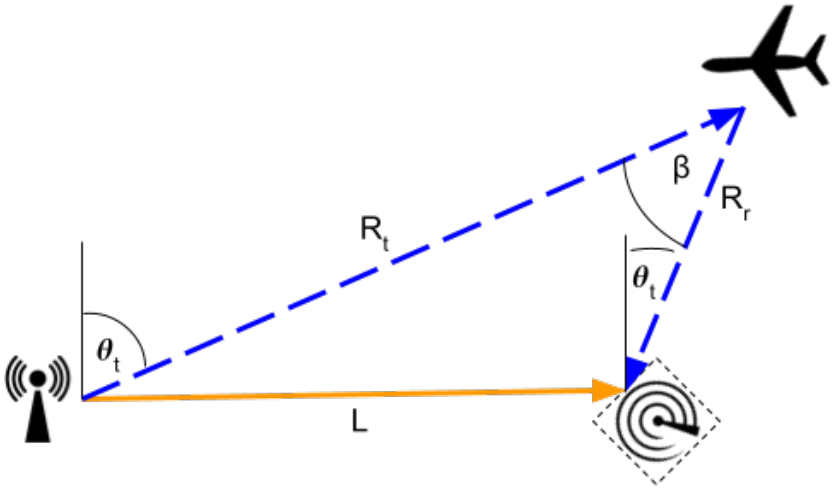


1.2. ábra. A bisztatikus radar térbeli sémája

A következőkben megmutatom a fontosabb elveket, melyek szerepet játszanak a radarrendszer jelfeldolgozásában és használatában.

1.1.1. A pozíció meghatározása

A céltárgy pozíciójának megismerése a radar felhasználás elemi célja. A elv alapvető lépéseit az 1.3. ábra felhasználásával mutatom be.



1.3. ábra. A geometriai rendszer

Jelen alkalmazásban a kétdimenziós esetet vizsgálom. Ezzel a megkötéssel a céltárgyaknak két szabadsági foka van, melyet legegyszerűbben az Rx pontba állított polárkoordináta rendszerben lehet megkeresni.

Távolság

Az adó, a vevő és a céltárgy általános esetben kifeszít egy háromszöget, melyet a szakirodalomban bisztatikus háromszögnek neveznek. Ennek a háromszögnek geometriai paramétereire tudunk következtetni, ezzel megismerve a céltárgy pozícióját.[3]

Az 1.2. ábra és 1.3. ábra együttes ismeretében a következő jelöléseket alkalmazom:

- L a Tx és Rx közötti távolság
- R_t a Tx és Target közötti távolság, R_x Rx és Target közötti távolság
- θ_t és θ_r a Tx-Rx tengely normálisa és a célpontra húzott egyenes szöge rendre Tx és Rx pontból
- β elemi geometria meglátások miatt $\theta_t - \theta_r$

A Tx pontban indulnak az EM hullámok. Mivel ezek mindig a közegbeli fénysebességgel terjednek, így az időtartamok és távolságok jó közelítést alkalmazva egyértelműen, konstans szorzótényezővel konvertálhatók egymásba.

Az Rx pontban a direkt (referencia) jelet τ_{ref} időpillanatban veszi a vevőegység, mivel a hullámnak meg kell tenni az L utat. ($\tau_{ref} = L \cdot c$) Az abszolút időskála jelen kérdésben nem szükséges, érdemes relatív időben dolgozni, az Rx pontból nézve.

A reflektált (megfigyelő, surveillance) jel viszont már τ_{sur} pillanatban érkezik meg, ahol $\tau_{sur} = (R_t + R_r) \cdot c >> \tau_{ref}$. Tehát a reflektált hullám késik, mivel nagyobb utat kell megtennie. Ez a feltétel meghatározza a síkban elfoglalt lehetséges reflexiós helyeket: az $R_t + R_r$ távolságösszeg ismert, ez egy a síkban ellipszis jellegű görbét ad meg, melynek gyújtópontjai Tx és Rx, ezek távolsága L, az azonos távolságú pontok távolságösszege R.

$$R = R_t + R_v = (\tau_{sur} - \tau_{ref}) \cdot c + L \tag{1.1}$$

Ezzel az egyik szabadsági fok kiesett, az *R (Range)* meghatározásra kerül.

Irány

A távolság ismeretében szükségünk van továbbá a céltárgy szögének, relatív irányának ismeretére. Ennek meghatározása legalább két módon történhet:

- forgó antenna: az antenna fizikai forgatásával változtatjuk az iránykarakterisztikát, így megmondhatjuk a jel milyen szög alatt látszik (benne van-e az adott pillanatban vett mintában, vagy nincs)
- antennarendszer és iránybecslő algoritmusok használata *

Előbbi mechanikai módszer körülményesebb, például a tömeggel rendelkező antenna fizikai időállandója miatt; vagy a technológia gépészeti és üzemeltetési bonyolultsága okán. Ennek megfelelően a második, szoftveres megoldásra koncentrálunk.

A *DOA (Direction of Arrival)* eljárások lényege, hogy egy beérkező hullám forrásának irányát az antennarendszerhez képest meghatározzák a mintavételezett *IQ* jel ismeretében.

A működés alapelve hogy az antennaelemekre a beeső hullám eltérő fázisban jut a térbeli szeparáció miatt. Az elemek távolságának és karakterisztikájának ismeretében a hullám érkezési iránya megbecsülhető. A pontosabb és hatékonyabb módszerek készítése jelenleg is aktívan kutatott téma [4] [5] [6], a projektem során a következő eljárásokat használom: [7]

- Fourier módszer
- CAPON
- MEM

Az iránymérés során az első feladat a $PAD(\theta)$ függvény meghatározása, mely a poláris szög függvényében adja a vett jelre vonatkozó teljesítmény jellegű mennyiséget. Második feladat ennek maximumának megkeresése, ez triviális.

A Fourier módszer arra a megállapításra épül, hogy az antennarendszer karakterisztikáját utólag is lehet matematikailag forgatni a megfelelő irányfüggő pásztázó súlyvektorral, tehát ha a pásztázással az elemek közötti fáziseltolódást ha kiegyenlítjük, akkor pontosan a beesési szöget forgattuk az antennán - ezzel megtaláltuk a keresett irányt

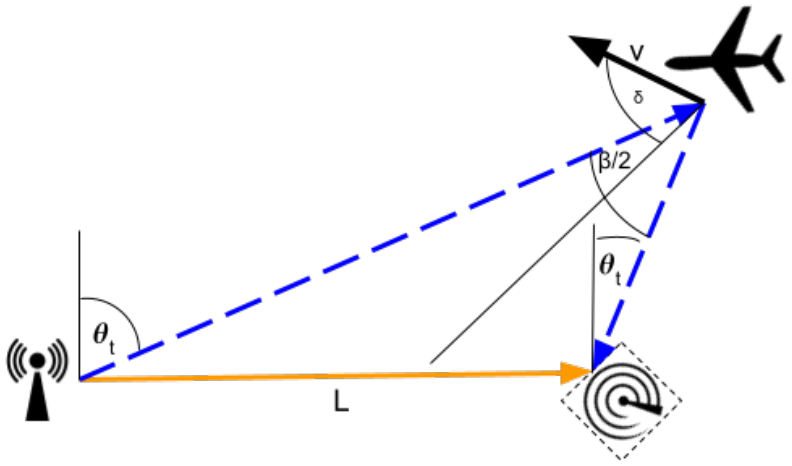
A CAPON módszer célja az abszolút teljesítmény helyett a jel-zaj viszony maximumának megkeresése, ehhez a Fourier módszerhez hasonló pásztázó forgatás mellett további MSINR nyalábformálást végez, ezzel elnyomva a megjelenő melléknyalábokat (zavaró teljesítményeket), és így pontosabbá téve a mérés eredményét. Ezt a módszert nevezzük még minimum variancia módszernek is.

A MEM (Maximum Entrópia Módszer) során igyekszünk minden információt felhasználni, ami a jelről rendelkezésre áll, és a nem elérhető adatokra nézve maximális bizonytalanságot kell kalkulálni. A Fourier becsléshez hasonlóan járunk el, csak figyelembe vesszük hogy az autokorrelációs mátrix nem nulla. A MEM módszer rendelkezik a legjobb felbontással ezen megoldások közül, de számítása körülményes és nehezen skálázható. [8] [9]

1.1.2. Sebességinformációk

A passzív radarral meghatározható a céltárgyak, reflektáló pontok sebessége is. Ennek elvi alapja az a tény, hogy a tárgy megfigyelőhöz képest vett relatív sebessége a reflektált hullám frekvenciáját befolyásolja a Doppler-hatás miatt.

A Doppler eltolódás mértéke a bisztatikus távolság (R) időbeli változásától függ.



1.4. ábra. A geometriai rendszer a Doppler hatás értelmezéséhez

Az abszolút sebesség függő f_D Doppler-frekvencia levezetése matematikailag körülményesebb, a szakirodalom alapján az 1.2 egyenlettel leírható: [10]

$$f_D = \frac{2 \cdot v}{\lambda} \cdot \cos\delta \cdot \cos\beta/2 \tag{1.2}$$

ahol λ a megfigyelő frekvenia hullámhossza, és az ábra szerint jelölöm a többi paramétert.

1.1.3. Jelszintek, megkötések

A bisztatikus radaregyenlet segítségével becslést tehetünk a rendszerben jelenlévő megfigyelési jelszintre (1.3 egyenlet), és a jel-zaj viszonyra (1.4 egyenlet):

$$P_r = \frac{P_t G_t}{4\pi R_t^2} \sigma_b \frac{1}{4\pi R_r^2} \frac{G_r \lambda^2}{4\pi} \tag{1.3}$$

ahol az újonnan definiált szimbólumok P_r a visszaverődött hullám teljesítménye, P_t a megvilágító forrás teljesítménye, G_t az sugárzó antenna nyeresége, σ_b a céltárgy radarkeresztmetszete (RCS), G_r a vevő antenna nyeresége.

A modell *LOS (Line-of-Sight)* elhelyezkedést feltételez, és pozíciófüggetlen RCS-t.

$$\frac{S}{N} = \frac{P_t G_t \sigma_b G_r \lambda^2}{(4\pi)^3 R_t^2 R_r^2 k T_0 B} \tag{1.4}$$

Érdemes még megjegyezni, hogy a referencia és a megfigyelő jelek között nagyságrendbeli eltérések vannak: míg a direkt jel inverz négyzetes csillapítást szenved, fizikai-geometriai elvek miatt a reflektált jel inverz negyedfokúlag csillapodik (mint minden radarnál). A passzív rendszereknél ez extra problémát okoz, mivel a megvilágító jelet nincs lehetőségünk lekapcsolni a reflexió megfigyelésének idejére. Ennek megfelelően elkerülhetetlen a direkt jel megjelenése a megfigyelési csatornában, főleg erősen szuboptimális elrendezésekben. Ez pedig rontja a számunkra érdekes reflektált jel-környezet szintviszonyt. Ezért a gondos fizikai orientáció megválasztás mellett (az antennarendszer főnyalábja lehetőleg ne a referenciasugárzó irányába mutasson) feltétlenül szükséges utólagos szűrés és referencia elnyomás a megfigyelő csatornákban.

1.2. A megvilágító jel paraméterei

A passzív radaros rendszer alapvető tétele, hogy csak a tőlünk független, amúgyis jelenlévő jel visszaverődését detektáljuk, magát a megvilágító jelet nem befolyásolhatjuk.

Ez magában hordoz nehézségeket, követelményeket a jellel kapcsolatban. A megoldás szükségessé teszi, hogy a jel számunkra jó korrelációs tulajdonságokkal rendelkezzen, egyértelműségi függvénye megfelelő legyen - az optimális jel véletlenszerűnek tekinthető, zaj jellegű.

Az egyértelműségi függvény megmutatja, hogy a jel mennyire korrelál jól saját maga időben-frekvenciában eltolt változataival. Kiszámítása lehetséges az 1.5 egyenleten látható módon:

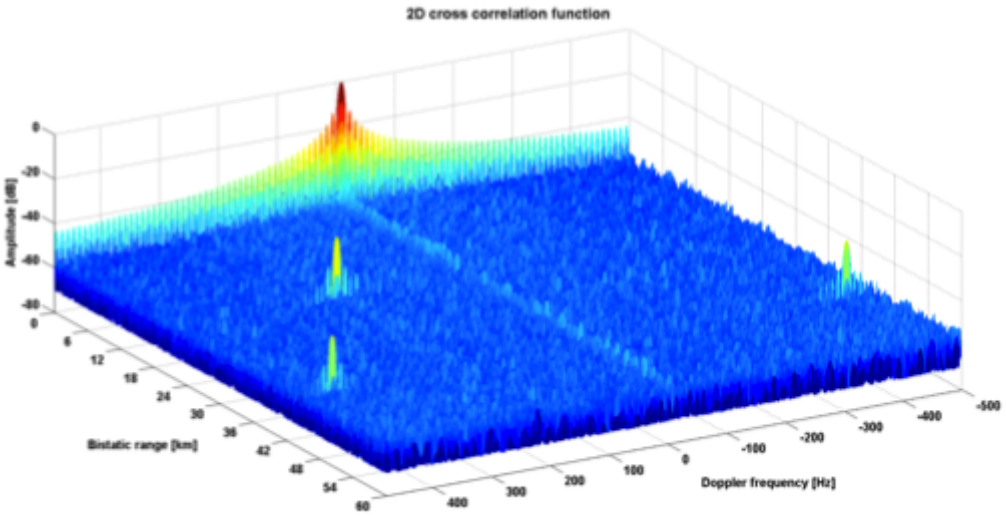
$$|\chi(\tau, f_D)| = |\int_{-\infty}^{\infty} s(t) s^*(t+\tau) e^{j2\pi f_D t}| \tag{1.5}$$

ahol τ az időmenzió, f_D a Doppler, s pedig a vizsgált időfüggő jel.

Ha egy jel jól korrelál a saját időben eltolt változataival, akkor a korrelációra alapuló minta és fázis felismerés nem fog működni a feldolgozási folyamat során, nem meghatározható a távolság. Ha a jel jól korrelál frekvenciában eltolt önmagával, akkor pedig a sebesség nem határozható meg, és a távolság detekcióban is gondot okoz a szivárgások miatt.

Ennek megfelelően a passzív radarozás szepontjából a megfelelő jel a zaj lenne, de jól használható például a DVB-T adás. A DVB-T jel egyértelműségi függvényében viszonylag kevés csúcs van, amelyek hamis detektálást jelenthetenek a passzív radar feldolgozás

szempontjából, valamint a dinamikatartománya közelítőleg 40dB. Ez alkalmas valós mérésekre. [14] Az 1.5 ábrán látható a függvény.



1.5. ábra. A DVB-T jel egyértelműségi függvénye (Pető Tamás ábrája, [14])

1.3. A működés

1.6. ábra. A jelfeldolgozás blokkvázlata

A program működésének megértéséhez áttekintem a passzív radaros felderítési rendszerünk átfogó lépéseit. Fontos látni, hogy a rendszer folytonos, jó korrelációs tulajdonságokkal rendelkező referenciajelre lett kifejlesztve, tehát a megvilágító jel folytonossága követelmény.

Előkészítő feladat a hardware megfelelő telepítése és a rendszer beindítása. A jelfeldolgozó program bekonfigurálása után kezdődik a mérés. Az antennákon vett jelet a nagysebességű szoftverrádiókkal digitalizáljuk. Ezekről optikai kapcsolaton egy Linux rendszert futtató számítógép letölti a mintákat. Egy ilyen elemi mintacsomagot dolgoz fel az általam fejlesztett program.

Első lépésként szükséges a megfigyelési csatornák szoftveres szűrése, hogy a beszivárgó referencia jel szintjét a szükséges szint alá csökkentsük. Ezt egy masszívan párhuzamosított adaptív (Wiener) szűrő végzi a referencia minták ismeretében.

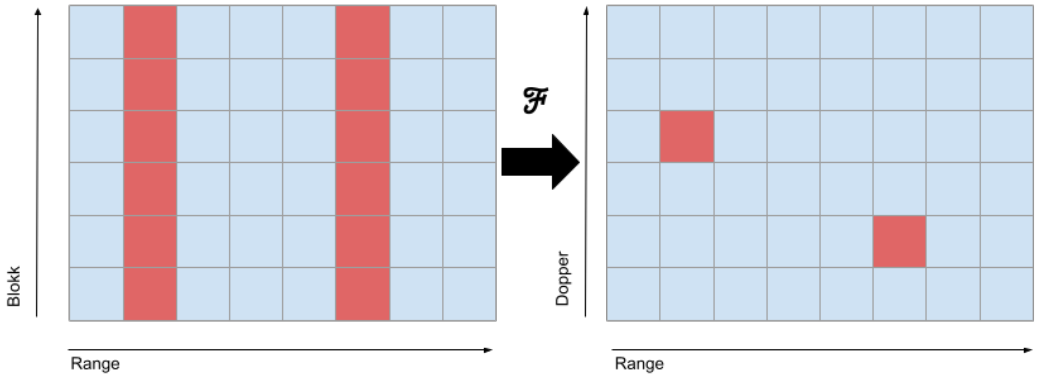
Második feladatként a elvégezzük a referencia és megfigyelési csatornák keresztkorrelációs 2D műveletét, hogy a céltárgyak helyzetét és sebességét megismerjük. Az elvi matematikai megoldás a következő 1.6. egyenlet lenne (a Doppler radar elkülönítési egyenlete), ha s_{sur} a megfigyelő jel, s_{ref} a referenciacsatorna:

$$|\chi(\tau, f_D)| = |\int_{-\infty}^{\infty} s_{sur}(t) s_{ref}^*(t-\tau) e^{-j2\pi f_d t}| \tag{1.6}$$

Szemléletesen mit jelent a radar elkülönítési egyenlete:

- τ dimenzió: Kihasználjuk, hogy az 1D keresztkorreláció ott vesz fel nagyobb értéket, ahol a vizsgált jel hasonlít a referenciajelre.
- f_D dimenzió: Kihasználjuk, hogy két komplex függvény skaláris szorzata nulla, amennyiben más frekvenciával rendelkeznek, ezzel elkülönítve a különböző sebességű céltárgyakról visszaverődő jeleket.

Ennek a operációnak kiszámítása például $1024 \cdot 512$ cellára elképesztő időbe telne. A sebesség meghatározásához naív esetben a referencia eltolt verzióival kéne korrelációt végezni, ám ez számítási szempontból igencsak lassú megoldás. Ezért részblokkokra bontva kétszintű impulzuskompressziós feldolgozási megoldást alkalmazunk. A keresztkorrelációs függvény az egyes blokkokban ott lesz a legnagyobb értékű, ahol a direkt és a reflektált jel között hasonlóság van. Először a távolságot határozzuk meg ilyen módon, gyakorlatilag idő dimenzióban végzett korrelációval - ezzel *range-compressed* impulzus vektorokat előállítva. A CW radaroknál is alkalmazott módon Fourier transzformációt végzünk az egymás alá írt blokkok mátrixán oszloponként, ezzel a fázisban tömörített sebességinformációt frekvencia jellegűvé konvertáljuk. Ezzel előáll az *RV (Range-Velocity)* mátrix, egyes irodalmak nómenklatúrájában RD. [12] [11] Ez úgy is felfogható, hogy a részblokkok között a céltárgy nem sokat mozdul, ellenben a fázisban megváltozik a reflektált hullám és mivel a felbontás fázisban jó, abból számítható a sebesség.



1.7. ábra. Az RV mátrix kialakulása

Az 1.7. ábrán láthatjuk a rendszer elvét, miszerint a két objektum más távolsága oszloprendben látszik, majd ebből emeli ki az FFT a pontos célcellát sebességet meghatározva.

Harmadik lépésben meg kell találni a lehetséges célpontokat, kiszámolni pozíciójukat és sebességüket a további követéshez. Ehhez fix számú maximumot keresük az RV mátrixban, kimaszkolva a frissen talált maximum környéki további értékeket (mivel a zaj és egyéb bizonytalanságok miatt nem diszkrét, egyetlen cellás maximumok keletkeznek az RV mátrixban). El kell végezni az iránybecslést és a célpont fizikai pozícióját kiszámolni. Fontos említést tenni a konfigurálható maszkolásról: statikus létesítmények, ismert *clutter* jelenség, a zavaróadó hatását ki lehet vágni az RV mátrixból a megfelelő mérési eredmények ismeretében.

Végezetül, megfelelő szűrés és validáció után térképen kell ábrázolni a megtalált célpontokat, további kiegészítő információval.

1.4. Korlátok

A passzív radar rendszerek használatának számos előnye mellett jelentős korlátai is vannak, melyek mind elméleti, mind gyakorlati síkon gátolják a tökéletes működést.

Egyrészt feltétlenül szükséges külső megvilágítási forrás, mely esetenként problémás lehet. Továbbá, amikor rendelkezésre áll a szükséges sugárzó, az jellemzően rendkívül nagy teljesítményű, mely óhatatlanul beszűrődik a vevő bemenetére, ezáltal rontva a reflektált-jel/környezet viszonyt, ezzel rontva a felbontást, érzékelést.

Lehetséges probléma még az elvakítás: ha a céltárgy a korábban mutatott L szakaszon van, bemérése lehetetlen. Hasonló problémát okoz a zérus relatív sebességgel rendelkező céltárgyak megfigyelése is: a zéró-doppler sávban jelentős a terepi visszaverődés és a beszivárgó referencia csatorna alapján adódó "zaj", így a megfigyelés szintén lehetetlen.

Egyes környezetekben a több adótorony is viselkedhet zavaró hatásként, vagy a komoly statikus reflexiók, melyek virtuális kvázi-adóként jelentkeznek.

Probléma még a komoly jelfeldolgozási kapacitás igény: gyengébb hardveren még 2015-ben is több perces futásidőt jelentett millós nagyságrendű minta feldolgozása. Ez messze van a valós idejű működéstől.

A dolgozat következő fejezetében bemutatok egy szoftveres megoldást, mely ezeket a problémákat megoldani hivatott.

2. fejezet

A clRadar program

2.1. Feladat

A kitűzött cél egy hatékonyabb, gyorsabb, videokártyán futó jelfeldolgozási lánc előállítása volt, mely pontosságban azonos vagy jobb eredményeket generál, mint a létező processzoron működő megoldás. Ennek érdekében kerestem a szükséges optimalizációs lehetőségeket, és megismerkedtem az OpenCL programozási környezettel.

2.1.1. Előzmények, segédanyagok

Létezik a problémára *proof-of-concept* bizonyítottan működő megoldás, melyet Dr. Dudás Levente jegyez. Ennek a programnak segítségével, és aktív használata során gyűjtött mérési adatokon, lehetséges az új megoldás tesztelése és verifikációja. Ez szolgáltatja a teljesítménymérés referenciáját is.

A programcsalád jellemzően C nyelven van megírva, és a méltán népszerű *FFTW3* [25] könyvtárra épül.

A kódbázis következő elemei teszik szükségessé a legnagyobb volumenű számításokat, így ezek optimalizációjával és pontos teljesítményanalízisével foglalkoztam:

- fastwiener.c: a referencia csatorna szűrése a megfigyelési csatornákból
- rv.c: mintacsomagból RV mátrix előállítása

fastwiener.c

A *fastwiener.out* bináris *stdin*-en várja a REF és az n-edik SUR csatorna mintáit, melyből előállítja *stdout*-ra a megszűrt kimenetet. A kód képes különféle megoldásokat használni az egyenletrendszer megoldására a súly vektor előállításához: jelenleg egy Gauss jellegű egyszerű egyenletmegoldó számolja ki a súlyvektort, de beépítetten hívható python szkript, ami háttérben optimalizált matematikai könyvtárakat használ.

rv.c

Az *rv.out* futtatás után *stdin*-en várja a REF és az n-edik SUR csatorna mintáit, melyből előállítja *stdout*-ra a kimenetet. Ez a verzió már alkalmaz valamiféle párhuzamosítást, mivel a futtatásokat különböző CPU magokra ütemezve gyorsulás érhető el az egymagos teljesítményhez képest. További gyorsulást lehetne elérni az FFTW3 könyvtár multithreading opcióinak kombinációjával, de erre a kód architektúrálisan nem alkalmas. Alapvető megközelítése, hogy két fázisú RV számítást használ, nem a naív algoritmust. Fontos

még a "függőleges FFT" elv is: az átmeneti tár mátrix oszlopát FFT-zve áll elő a végleges RV. Ilyen memória orientációban nem lenne triviális az FFT algoritmus bemenetparaméterezése, ezért átmeneti buffert használ.

2.1.2. Optimalizációs lehetőségek

A CPU-n futó verzión is lehetne további változtatásokat tenni, a hatékonyabb futásért, ezeket a következőkben összegzem.

- A szűrés rengeteg teljesítményt veszteget arra, hogy minden szálban újra előállítson egy bonyolultan számítható mátrixot, ezt ki lehetne szervezni.
- Az egyenletmegoldási algoritmus bár egyszerű és átlátható, de rengeteg felesleges munkát végez. Egyszeri faktorizáció után sokkal optimálisabb lenne a többi csatorna megoldása.
- Az FFT könyvtár gyorsabb lehetne, hogyha nagyobb mintaszámon dolgozhatna egyhuzamban. Igaz, kis pontszám esetén a szinkronizáció lassíthatja is a futást, de ha pl. $12 \cdot 2 \cdot 2^{20}$ pont lenne egyben betöltve a memóriába, az algortimusnak megfelelően beparaméterezve, miszerint 2^{10} pontos FFT-ből csináljon sorfolytonosan megfelelő darabszámút, erre pedig használjon annyi szálat, ahány processzor, akkor ez valószínűleg gyorsabb lenne.
- A memóriaműveletek ilyen adatmozgás esetében lassúnak tekinthetők. Bonyolultabb paraméterezéssel az FFT könyvtár megbízható azzal, hogy a memóriában lineárisan elhelyzett, virtuálisan 2D mátrixnak elképzelt tárból x-edik pontokból szedjen ki pontosan M darabot, és ezekkel dolgozzon, ráadásul helyben, ugyanoda tegye vissza. Ezzel a manuális memóriamozgatás megspórolható.
- A korreláció számításhoz spektrumtartományban szükséges komplex szorzás rendkívül párhuzamosítható lenne, ehhez lehetne szálakat nyitni a processzoron.

Összefoglalva: valószínűleg lehetne további gyorsulást elérni processzoron is, de jelen dolgozatban behatóbban nem vizsgáltam ezt a problémát. További forrás elérhető a többszálú FFTW3 paraméterezésről a könyvtár weboldalán. [26]

2.2. Felhasznált technológiák

Fontos irányelv, hogy fejlesztés során csak nyílt, ingyenes, és könnyen hozzáférhető techonlógiát lehet használni. Ennek megfelelően az OpenCL technológiát részesítettem előnyben, a programot C nyelven írtam, és a tesztelést Linuxon végeztem.

Emellett a rendszer stabilitásának érdekében külön figyelni kellett az erőforráshasználat előzetes felmérésére, mivel a túlfoglalt videokártya komoly problémákat okozhat egy számítógépes rendszer működésében.

2.2.1. Platform

A GPU-s program futtatására egy *NVIDIA GeForce GT 730 2GB* kártyát használunk éles környezetben. A fejlesztés egy *Intel HD 4000* valamint *Intel HD 620 rev2* laptop kártyán történt.

Előbbi kártya lényeges paraméterei:

```
$ clinfo
Number of devices                                 1
  DeviceName                                      GeForce GT 730
  Device Version                                  OpenCL 1.2 CUDA
  Driver Version                                  384.130
  Device OpenCL C Version                         OpenCL C 1.2
  Device Type                                     GPU
  Max compute units                               2
  Max clock frequency                             901MHz
  Compute Capability (NV)                         3.5
  Max work item dimensions                        3
  Max work item sizes                             1024x1024x64
  Max work group size                             1024
  Preferred work group size multiple              32
  Warp size (NV)                                  32
  Single-precision Floating-point support         (core)
    Support is emulated in software               No
  Global memory size                              2097741824 (1.954GiB)
  Max memory allocation                           524435456 (500.1MiB)
  Unified memory for Host and Device              No
  Alignment of base address                       4096 bits (512 bytes)
  Global Memory cache type                        Read/Write
  Global Memory cache size                        32768
  Global Memory cache line                        128 bytes
  Local memory size                               49152 (48KiB)
  Registers per block (NV)                        65536
  Max constant buffer size                        65536 (64KiB)
  Max size of kernel argument                     4352 (4.25KiB)
```

2.2.2. OpenCL

Az OpenCL egy nyílt, ingyenes sztenderd a cross-platform párhuzamos programozásra. [27]

Előnye, hogy a program képes minimális változtatással többmagos processzoron, valamint kompatibilis videokártyán, vagy egyéb eszközön is futni, így jól tesztelhető és portolható. Hátránya, hogy a gyártók saját megoldása egyes tesztek szerint gyorsabb lehet. Fő célom a projetkben egy OpenCL alapú implementáció előállítása volt, profilozási mérésekkel. Az OpenCL környzet és a programozási technológia megismerésében rengeteg segítséget nyújtottak *Prof. David Black-Schaffer* [22] ingyenesen elérhető YouTube videói a témában.

A hatékony FFT kódot a *clMathLibraries* projekt *clFFT* könyvtára biztosítja, mely ingyenes és nyílt forrású szoftver. [28] A kényelmes háromszögmátrixos egyenletmegoldást

és egyéb egyszerű lineáris algebrai módszereket a *clMathLibraries* projekt *clBLAS* könyvtárából vételeztem, bár ez sajnálatos módon NVidia architektúrán nem optimális.[29]

2.2.3. CUDA

A CUDA az NVidia saját párhuzamos számítási megoldása. Felhasználása a cég kártyáira korlátozott, azokra viszont magasan optimalizált. Egyes mérések szerint sebessége nagyobb lehet mint az OpenCL.

Sajnos a munka során nem jutott idő a CUDA megoldás kifejlesztésére, bár a technológiai különbségek vizsgálata valószínűleg izgalmas és meglepetéseket tartogató kutatási munka lenne, de az elérhető gyorsulás mértéke korlátozott, és a platform zártársága riasztó.

Változó neve	**Leírása**
system.channel_count	a rendszerbe bekötött referencia és megfigyelőcsatornák számának összege
system.block_size	az egy csomagban megkapott mintaméret (*complex float* típusok darabszáma)
wiener.tap_count	a generált adaptív (Wiener) FIR szűrő fokszáma (egyes platformokon max. 64)
rv.subblock_size	a részblokkok mérete (*complex float* típusok darabszáma) a kétfázisú számításban
rv.doppler_height	a visszaadott RV mátrix D irányú mérete, szűkíthető
hit.hit_count	a követett lehetséges célpontok maximum száma
hit.hitmask_R	a megtalált lehetséges célpont körül kimaszkolt terület R dimenziója (zaj és környezeti változók alapján)
hit.hitmask_D	a megtalált lehetséges célpont körül kimaszkolt terület D dimenziója (zaj és környezeti változók alapján)
hit.defmasks	előre definiált nullmaszkok az RV mátrixon, statikus zavaró jelenségek kiiktatására
dir.elem_offset	az antennaelemek távolsága (m)
dir.min_hits	ennyi találattól számolja ki az iránybecslést az adott maximumra
dir.enable_fourier	a Fourier iránybecslés engedélyezése
dir.enable_capon	a CAPON módszer engedélyezése
dir.enable_mem	a MEM módszer engedélyezése
dir.mem_col	a MEM módszerben kiválasztott oszlop
pos.frequency	a rendszer frekvenciája
pos.rx	a vevőantenna GPS pozíciója, heading-je
pos.tx	az adóantenna GPS pozíciója
track.threshold	a követés küszöbszintje

2.1. táblázat. A clRadar program konfigurációs változói

2.3. GPU jelfeldolgozás

2.3.1. Paraméterek

A könnyebb érthetőség érdekében a leírás elején bevezetem a lépésekben használt elvi szinten jelentést hordozó (konfigurációs) változókat, melyeket a 2.1. táblázatban összege-

zek.

További elméleti szinten lényegtelen, de üzemeltetési és tesztelési szinten fontos beállítások is lehetségesek a programban, és így aktuális futtatókörnyezetre könnyen átállítható. A konfigurációt a nagyszámú paraméter miatt már egyedi konfigurációs állományból töltöm be, melyre a felület szerkesztési lehetőséget fog adni.

2.3.2. Előkészítés

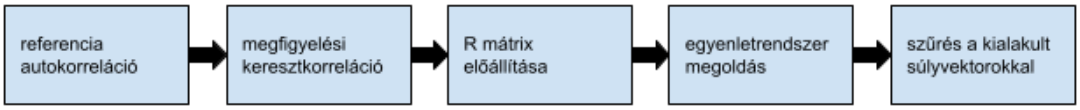
A program fájlból, vagy egyéb módon betölti a $system.block\_size \cdot system.channel\_count$ mennyiségű komplex mintát. A formátum 2x IEEE754 32-bit float, mely egyes rendszereken a *complex float* adattípusnak felel meg. A bemeneti fájlformátum lényegtelen, nyers IQ adatokat vár a program, mely általában *.cf32, .bin, .dat* kiterjesztéssel rendelkezik.

Rögtön itt történik az első memóriaművelet, a minták leírása a GPU memóriába. Ez a lépés jelentős, mivel a GPU-CPU busz sávszélessége korlátos, és más komponensek is jelen vannak a rendszerben.

Fejlesztési tapasztalat, hogy ez a folyamat integrált videokártyás (Intel) laptopokon gyorsabban megtörténik, mint dedikált GPU-val rendelkező platformokon. Talán ez az integráció az egyetlen előnye a RAM-ban allokált VRAM-nak. Fontos megjegyezni még, hogy ez a lépés hibás mutatók vagy a memóriatartományt védő flagek esetén végzetes lehet a kernelre nézve - azonnali kernel panic a figyelmetlen fejlesztő jutalma.

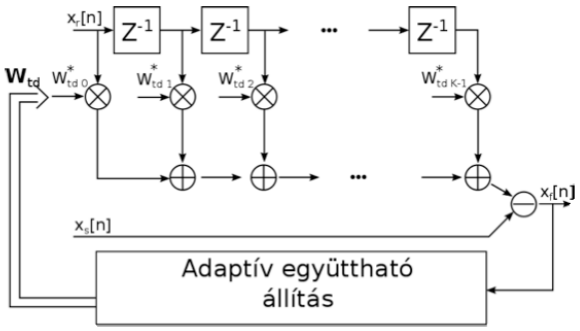
A továbbiakban minden memóriaművelet a videokártyán történik, így sebessége kielégítően magas.

2.3.3. Wiener-szűrés



2.1. ábra. A lépés átfogó sémája

A 2.2. ábra bemutatja a Wiener-szűrés általános blokkvázlatát:



2.2. ábra. A Wiener szűrés blokkvázlata (forrás: Pető & Seller [14])

A diszkrét Wiener szűrés matematikai alakja az 2.1. egyenlet szerint:

$$sur_{iclean}[n] = \sum_{j=0}^{N} a_i[j] sur_{iraw}[n-i] \tag{2.1}$$

ahol sur_{iclean} az i-edik megfigyelő csatorna szűrt jelvektora, a_i az i-edik csatorna Wiener együtthatóvektora, sur_{iraw} pedig a rádióról érkező nyers minta.

A szűrő együtthatóvektorához elő kell állítani, majd meg kell oldani csatornánként (az átláthatóság érdekében a csatornaindexet elhagyom) a 2.2. egyenletrendszert:

$$\underbrace{\begin{bmatrix} ref_{auto}[0] & ref_{auto}[1] & \cdots & ref_{auto}[N] \\ ref_{auto}[1] & ref_{auto}[0] & \cdots & ref_{auto}[N-1] \\ \vdots & \vdots & \ddots & \vdots \\ ref_{auto}[N] & ref_{auto}[N-1] & \cdots & ref_{auto}[0] \end{bmatrix}}_{\mathbf{R}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix}}_{\mathbf{a}} = \begin{bmatrix} sur_{cross}[0] \\ sur_{cross}[1] \\ \vdots \\ sur_{cross}[N] \end{bmatrix} \tag{2.2}$$

Az egyenletrendszerhez szükség van a referenciacsatorna autokorrelációs függvényére (ref_{auto}), és minden csatorna keresztkorrelációjára is a referenciával (sur_{cross}). [16]

A Wiener-Hopf egyenletrendszer előállítása

Az első lépése az autokorreláció előkészítése: ehhez a memória megfelelő helyeire kell másolni a referencia csatorna teljes adatmennyiségét, tehát $2 \cdot system.block\_size$ mintát. Az autokorrelációt spektrumban számítjuk ki a numerikus optimalitás miatt. Ennek megfelelően először FFT-t végez a program külön a két $system.block\_size$ méretű blokkon, majd a második vektorba beírja az elemek között készített a konjugált szorzatot,

és ezt a blokkot visszaszámolja a megfelelően paraméterezett IFFT-vel időtartományba. Így a memóriában megjelennek a referencia csatorna autokorrelációs függvényének értékei.

Hasonlóan végezzük el a keresztkorrelációkat is, minden csatornára, tömbösített FFT és IFFT operációkkal.

Fontos esemény még az R mátrix generálása az autokorrelációs függvényből. Mint azt az értékelési szekcióban majd még bővebben kifejtem, a lineáris memóriahozzáférés kritikus pont a videokártya sebesség kérdésében; így egyedi megoldást kellett fejlesztenem az R mátrix felépítésére.

Az R egy $wiener.tap\_count \cdot wiener.tap\_count$ méretű, Hermitikus és Töplitz tulajdonságú mátrix. Ezen tulajdonságok alapján kiindulva megmutatható, hogy egyetlen sora is generálja az egész mátrixot. Nem érdemes viszont az elemeket egyesével létrehozni/másolni, mert súlyosan hatékonytalan, nagyon nem szereti a GPU az ilyen memóriahozzáférést. Valós elemű példán keresztül mutatom be az általam fejlesztett megoldás működését, konjugált helyett ellentett operátort használva.
Legyen a generálni kívánt mátrix első sora:

$$R[1] := \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

És bevezetek egy *genvec* segédvektort, mely az R mátrix első sorának "tükörképe" a 2. elemtől kezdve:

$$genvec := \begin{bmatrix} -4 & -3 & -2 & 1 & 2 & 3 & 4 \end{bmatrix}$$

Vegyük észre, hogy a generátor vektor 4. oszlopától egy 4 méretű csúszóablakkal visszalépegetve mindig egy sornyi következő adatot látunk az R mátrixból, amelyet immár megfelelő memóriahozzáféréssel rögtön be is írhatunk a megfelelő helyre:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ -2 & 1 & 2 & 3 \\ -3 & -2 & 1 & 2 \\ -4 & -3 & -2 & 1 \end{bmatrix}$$

A Wiener-Hopf egyenletrendszer megoldása

A szakirodalom említi a Levinson-Durbin [17] algoritmust hasonló alakú egyenletrendszerek megoldására. Ezen algoritmus numerikus stabilitása kétséges, és optimalizálhatósága GPU futtatókörnyezetben nem hatékony a felhasznált rekurzív futási minta miatt. Ezért a hagyományos LU-faktorizációs, majd háromszögmátrixos egyenletmegoldó módszert választottam.

Az LU-faktorizációt saját készítésű és hangolású kernelek végzik Gauss eliminációval, mely meglepően gyors. A megoldás lépései:

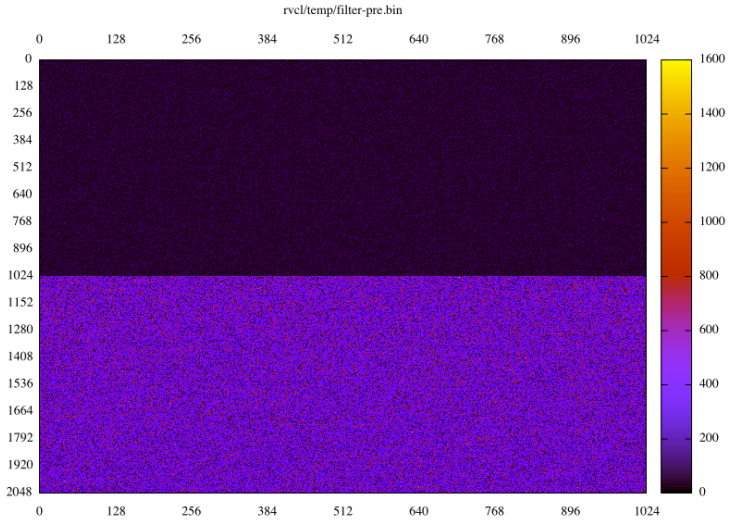
- A következő sorra vonatkozó eliminációs együttható(k) kiszámítása egy segédvektorba.
- A segédvektor alapján a következő célpont nullázása, átvezetés.

Fontos, hogy az egységmátrixot minden processzálás után újra kell építeni, mert a gyorsaság érdekében az elimináció után benne marad az egyik háromszög.

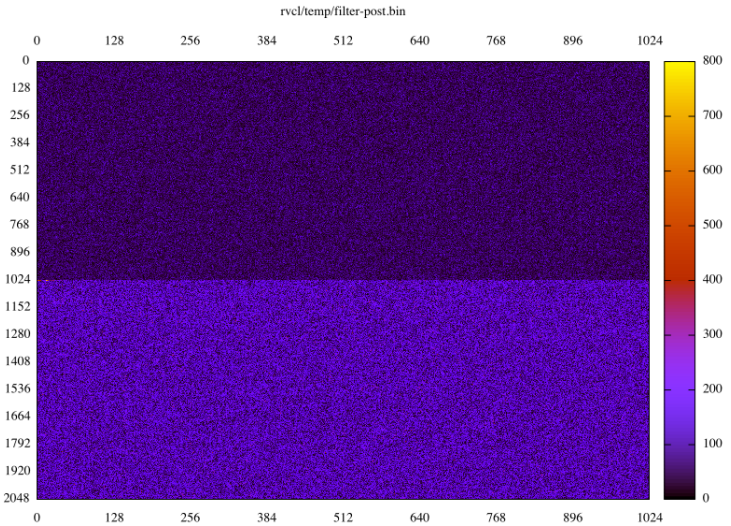
A háromszögmátrixos rendszer megoldását a clBLAS könyvtárra bíztam, mivel elemi feladat.

Szűrés

A bevezetőben ismertetett egyenlet alapján a szűrés már egyszerű lineáris feladat a súlyvektor ismeretében. A 2.3. ábrán látható a szűrés előtti és utáni állapot. Ránézésre észrevehető, hogy a jelszint csökkent, tehát valamit csinált a szűrés, helyességét a következő fejezetben mutatom be.



(a) Szűrő bemenet

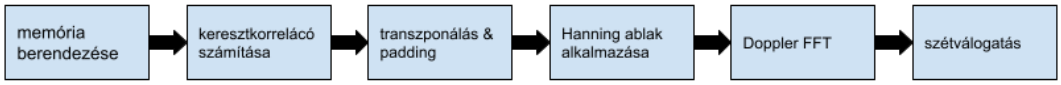


(b) Szűrő kimenet

2.3. ábra. Wiener szűrő hatása

A 2.1 ábrán látható feladatok elvégzésével rendelkezésre állnak a referencia csatornát kevésbé tartalmazó megfigyelési csatornák, melyet a következő programmodul felhasználhat. [16]

2.3.4. RV mátrix generálás



2.4. ábra. A lépés átfogó sémája

Az RV mátrixot a következő elv alapján számoljuk, mely a 2.4.-es ábrán folyamatában látható: A koheresen mintavételezett nagyméretű blokkból készítünk $rv.subblock\_size$ méretű részblokkot, ahol T_c egy részblokk mintáinak begyűjtéséhez szükséges idő. Ezek elemeinél az impulzus értéke nulla Doppler eltolódásnál a következő 2.3. egyenlet alapján lesz az i. blokkban meghatározható, melyet a $\chi^{'}(\tau, 0)$ segédvektorokkal jelölök, expliciten jelölve a még ismeretlen sebességinformáció hiányát:

$$\chi^{'}(\tau, 0) = \int_0^{T_c} r(t + iT_c)s^*(t + iT_c - \tau)dt \tag{2.3}$$

Ez a feladat alakját tekintve egy korreláció, amely műveletre pedig ismert, hogy FFT segítségével spektrumban elvégezve megoldása gyorsabb. Elméleti indoklás:

$$\mathcal{F}\{f \star g\} = \overline{\mathcal{F}\{f\}} \cdot \mathcal{F}\{g\}$$

A tétel bizonyítása hasonlít a Fourier transzformáció közismert konvolúciós tételére.

Ezután már csak a Doppler információ kibontása szükséges, melyet a következő módon tehetünk meg:

$$\chi(\tau, \mu) = \sum_{i=-N/2}^{N/2-1} \chi(\tau, 0) \exp \frac{-j2\pi}{N} \mu i \tag{2.4}$$

Ez egy FFT. (μ definiálásra kerül mint f_D becslője) [15]

Memóriaműveletek

A keresztkorrelációk kiszámításához különleges elrendezésű memóriára van szükség, ahol az $rv.subblock\_size$ méretű vektorokat további azonos mennyiségű nullvektorral egészítjük ki, hogy az FFT megfelelően paddingelve legyen. Ezt a másolást iteratív módon készítem, sajnos jelentősebb mennyiségű időt vesz igénybe, de tovább nem optimalizálható.

Keresztkorreláció

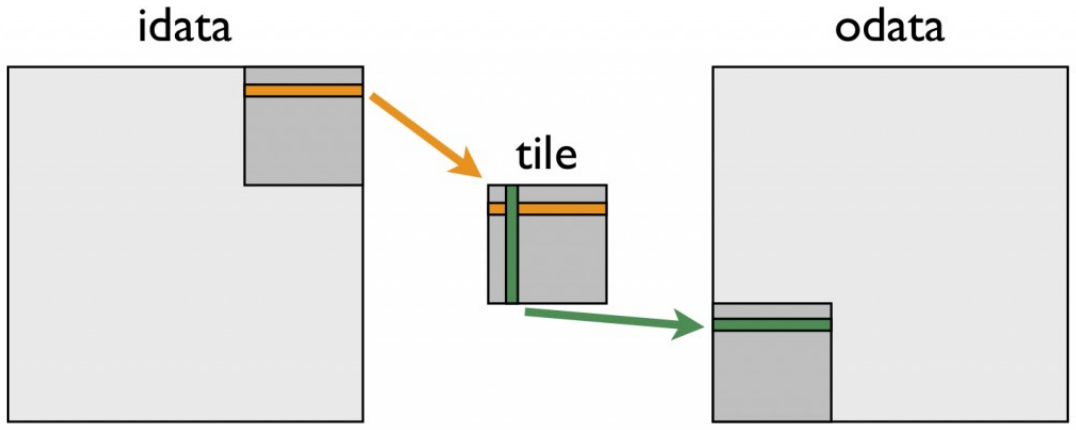
A program elvégzi a spektrumban a korrelációt: FFT-t készít blokkonként, a megfigyelési csatornákban helyben elkészíti a komplex konjugált szorzást a referenciacsatornával, majd a jelet visszahozza időtartományba.

Transzponálás

A D dimenzió meghatározáshoz szükséges a blokkokból képzett mátrixon az előző fejezetben ismeretett módon az oszloponkénti FFT. Ehhez szintén nullvektor paddig szükséges,

valamint ablakozás a jobb spektrum érdekében. További nehézség, hogya clFFT könyvtár nem igazán alkalmas oszloponkénti memóriahozzáférés optimalizálásra, ezért az egész mátrixot kénytelen vagyok transzponálni, melyre saját megoldást fejlesztettem.

A transzponálás GPU platformon a *coalesced* memóriahozzáférés kényszere miatt nehézkes. A hardver arra van optimalizálva hogy nagy, akár 256 bites blokkokban férjen hozzá a RAM memóriához, és ezeket cachelje rögtön tovább. A transzponálás klasszikusan nem lineáris hozzáférési probléma, és ez a naív algoritmus működésén azonnal meglátszik. Az NVidia fejlesztői blogján személetesen bemutatja a szerző a problémát, és a válaszul adott értelmes megoldást (2.5 ábra): [18]



2.5. ábra. Az optimális transzponálási algoritmus (forrás: NVIDIA)

Lokális memóriába kell kikérni egy egész sor adatot, helyben megforgatni, mivel a lokális memória hozzáférés sokkal gyorsabb, majd lineárisan beírni a célmátrixba, ilyen megoldással akár 5x-10x gyorsulás realizálható a naiv kernelhez képest. Ez a módszer tökéltesen átültethető OpenCL-re, a nyilvánvaló hátulütőivel:

- A lokális memóriában is kerülhet *bank-conflict* állapotba a memória, ami lassítja a feldolgozást - lehetséges hogy emiatt csökkenteni kell a *tile* méretet.
- Minél nagyobb egy *tile* mérete, annál jobb a nyereség - viszont a transzponálni kívánt mátrix mérete korlátozza.
- Nehézkes átláthatóság.

Hanning ablak

Az ablakozó függvényt előre legenerálom a program indulásakor, $rv.subblock\_size$ méretben, ezzel csökkentve a terhet feldolgozási időben. Ezt a tárat nem kell minden újabb mintacsomag fogadásakor újraírni. A következő függvény generálja az N elemű diszkrét Hanning ablakot, még a CPU-n:

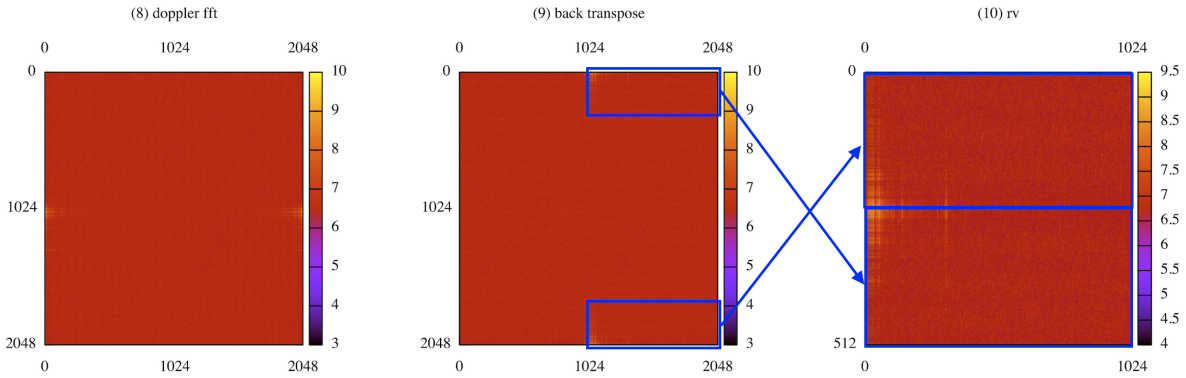
$$w[n] = w_0\left(\tfrac{L}{N}(n - N/2)\right) = \left.\begin{array}{r} \tfrac{1}{2}\left[1 - \cos\left(\tfrac{2\pi n}{N}\right)\right] \\ = \sin^2\left(\tfrac{\pi n}{N}\right) \end{array}\right\}, \quad 0 \le n \le N,$$

A Hanning ablak alkalmazásához saját kernelt fejlesztettem, mely kielégítő sebességgel képes a fél mátrixot felszorozni. Elmond a teljesítményről valamit, hogyha (feleslegesen) felszoroztatom a nullás padding vektorokat is, az se változtat a teljes feldolgozásra vetített időszázalékában.

Ezután a lépés már meglepően gyorsan lefut.

Leválogatás

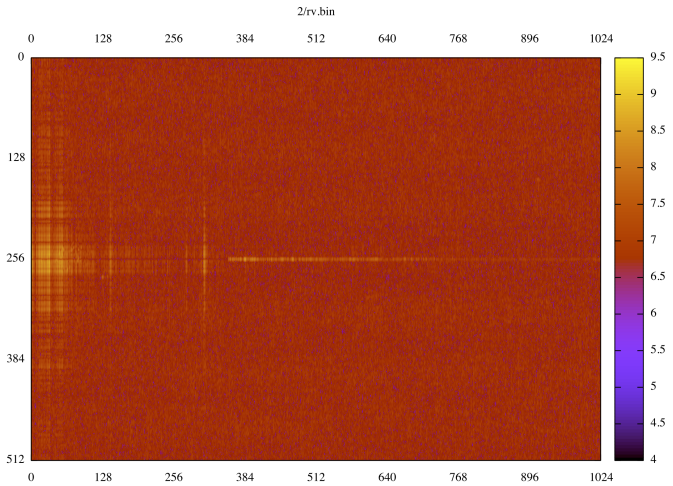
Az eredménymátrixban már jelen vannak az RV mátrixok, ezt egy bonyolult saját kernel transzponálja vissza és rendezi átmeneti tárba a kész RV mátrixokat, ahonnan a következő lépés átveszi azokat. [15]



2.6. ábra. Az RV mátrix rendező kód futása

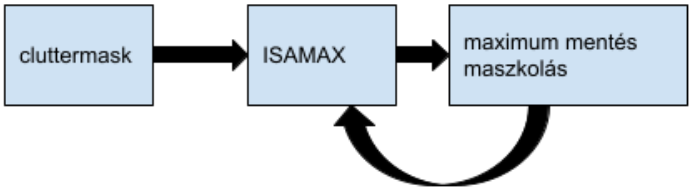
A 2.6 ábra illusztrálja az RV mátrixok rendezésének nehézségét. Először mátrixonként transzponálni kell a Doppler FFT-k oszlopából, majd kiválasztani a pont fordított sorrendben lévő RV szekciókat, és helyesen bemásolni a célterületre. Mindezt maximális lokális memória felhasználás mellet kell végezni a cache minél jobb érvényesítéséhez.

A következő 2.7. ábrán illusztrálom a folyamat végermékét. Amikor elkészült ez a lépés, tovább lehet haladni a hit térkép meghatározására.



2.7. ábra. Az RV GPU-n generálva

2.3.5. Hit keresési megoldás



2.8. ábra. A lépés átfogó sémája

Az RV mátrixban a lehetséges céltárgyakról történt visszaverődést jelenleg egyszerű maximumkereséssel detektálom komplex CFAR algoritmusok helyett. A feldolgozás sémáját a 2.10. ábra szemlélteti.

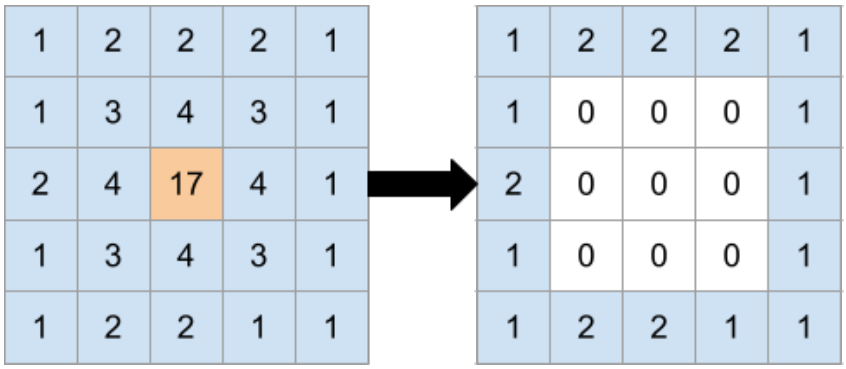
Cluttermask

A clutter és az egyéb ismert hibás pontok kiszűréséhez először az alapértelmezett $hit.defmask$ maszkokat alkalmazom az RD mátrixra. Ez a következő módon történik:

- A program indulásakor inicializáltam a $defmask$ mátrixot. Ez a maszkolandó helyeken (0, 0) elemet tartalmaz, a megmaradó pontokban (NAN, NAN) értékeket, dimenziói $(rv.subblock\_size, rv.doppler\_height)$.
- A fejlesztett kernel soronként, megfelelő tiling felhasználásával kiveszi az RV és MASK elemeket a globális memóriából, elvégzi a tesztet a MASK állapotára, és ha 0, beírja az RV megfelelő helyére.

Itt is fontos az egyesített írási és olvasási műveletek kiemelése, mivel nem minden RV elem változik, és a fragmentált memóriahozzáférés jelentősen lassíthatja a futást.

Maszkolt maximumkeresés



2.9. ábra. A maskedmax lépés szemléltetése ($hit.hitmask\_R = hit.hitmask\_D = 3$)

Ez a lépés iteratív, és annyiszor fut mint a $hit.hit\_count$ rendszerváltozóban definiálva van, tehát ennyi lehetséges célpontot keres mátrixonként.

A maximumkeresés közismert *BLAS Level1* rutin, tehát a *clBLAS* ISAMAX függvényét érdemes használni a feladatra, jobb megoldás fejlesztése kétséges. Érdekesség, hogy

a rutin használatához további átmeneti buffert igényel a könyvtár, melynek foglalását fejlesztőre bízza. Ez a függvény a legnagyobb abszolútértékű elem indexét adja vissza.

Az általam fejlesztett maszkoló kernel ezután a következő fő funkciót valósítja meg az összes RV mátrixban:

1. kimaszkolja a megtalált maximumot és annak közvetlen környezetét, a definiált ($hit.hitmask\_R, hit.hitmask\_D$) méretben, hasonlóan a cluttermask lépéshez, nullával.

2. kimásolja a maximumponton található IQ értéket az adott sorszámú maximumok $system.tap\_count$ hosszú sorvektorának megfelelő oszlopába.

3. a maximumhely pozícióját menti az adott sorszámú maximum pozíciókat tartalmazó $system.tap\_count$ hosszú sorvektorba

Ezután visszatérünk a következő maximumok kereséséhez.

2.3.6. Iránymérési megoldások

Előellenőrzés

Rendelkezésre áll a *hit*-ek értékmátrixa a következő alakban, valamint hasonló módon a pozíciók mátrixa:

$$\begin{bmatrix} M_{1,0} & \cdots & M_{C,0} \\ \vdots & \ddots & \vdots \\ M_{1,X} & \cdots & M_{C,X} \end{bmatrix}$$

Ahol $M_{c,x}$ a x-rendű maximum c-edik csatornán vett komplex (IQ) értéke.

Amennyiben legalább $dir.min\_hit$ csatornán rendelkezésre áll egy adott maximum, a *hit* pozíció mátrix alapján, csak akkor számolja ki a program a bonyolult algoritmusokat.

Az iránymérési algoritmust minden megfelelő sorra le kell futtatni, ezzel soronként keletkezik egy irányszög az adott rendű maximumhoz. Hibás futás vagy értelmezhetetlen vétel esetén NAN értéket jegyzek be a megfelelő oszlopvektor cellába.

Algoritmusok

Az iránymérést kapcsolható módon a következő három módon valósítom meg: Fourier módszer, CAPON módszer, MEM módszer. Minden módszerben kiértékelem a $PAD(\theta)$ (Power Anguar Density) függvényt a beállított minimális szögfelbontásban vett lépésenként. Segédváltozók és megnevezések:

- S_θ: a pásztázó vektor, mely a következő alakú: $\begin{bmatrix} 1 & e^{j1\beta dcos(\theta)} & \cdots & e^{jC-1\beta dcos(\theta)} \end{bmatrix}$
- sur_{corr}: a vett jelből alkotott azonos alakú vektor (a *hit* mátrix egy sora) korrelációs mátrixa

A sur_{corr} mátrix a következő alakban áll elő, ha v a *hit* értékmátrix adott sora:

$$\begin{bmatrix} v(0) \cdot v^*(0) & v(0) \cdot v^*(1) & \cdots & v(0) \cdot v^*(C-1) \\ \vdots & \vdots & \ddots & \vdots \\ v(C-1) \cdot v^*(0) & v(C-1) \cdot v^*(1) & \cdots & v(C-1) \cdot v^*(C-1) \end{bmatrix}$$

A Fourier becslés a következő 2.5 egyenlet alapján számítható:

$$PAD(\theta)_{Fourier} = S_\theta^H sur_{corr} S_\theta \tag{2.5}$$

A Capon becslés matematikai alakja (2.6 egyenlet):

$$PAD(\theta)_{Capon} = \frac{1}{S_\theta^H sur_{corr}^{-1} S_\theta} \tag{2.6}$$

A MEM becslés matematikai alakja (2.7 egyenlet):

$$PAD(\theta)_{MEM} = \frac{1}{|S_\theta^H sc_j^{-1} sc_j^{-1H} S_\theta|^2} \tag{2.7}$$

Ahol sc_j^{-1} a korrelációs mátrix inverzének egy j-edik kiválasztott oszlopa. Az oszlop megválasztása határozza meg a felbontás minőségét. [21] [20]

Ezen függvények megoldása programozástechnikailag nem bonyolult feladat. A közismert BLAS könyvtár funkcióival kiszámítható. Az optimalitás kérdése nem jelentős, mivel relatíve kis méretű mátrixokon dolgozik a megoldás. Felvetődik a kérdés, hogy ezen a ponton érdemes lenne CPU feldolgozásra váltani.

A tényleges szög meghatározása

A korábban említett ISAMAX BLAS függvény felhasználása adódik ebben az esetben is, a generált PAD vektorban futtatva. A szögfelbontást visszaszorozva a maximum helyével megkapható a tényleges szög. Ezt is fejlegyzem a maximumok pozícióit tartalmazó mátrixban, oszlopként helyesen.

Ezzel rendelkezésre áll a folyamatban a pozícióinformáció (ellipszis szinten), és a sebesség, mely további feldoglozást igényel.

Relatív pozíció meghatározása

Az ellipszis és a vevőből húzott megadott irányszögű egyenes metszete az első fejezetben ismeretett módon megadja a vevőhöz képesti relatív pozíciót poláris koordinátarendszerben. Ez a számítást már iteratív módon processzoron végzem, mivel a memóriakorlát miatt a GPU megvalósítás sebessége aránytalanul gyenge.

2.3.7. Követés és vizualizáció

2.10. ábra. A lépés átfogó sémája

GPS számítás

A relatív 2D pozíció és az Rx állomás pontos térbeli elhelyezkedésének ismeretében a céltárgy síkbeli pozíciója egyszerűen számítható - a föld görbületét a vizsgált tartományon elhanyagolva:

$$\begin{pmatrix} lat \\ lon \end{pmatrix} = \begin{pmatrix} rx.lat + rel\_dst\_lat \\ rx.lon + rel\_dst\_lon \end{pmatrix} \tag{2.8}$$

Ahol a relatív távolság koordináták a valós poláris távolság vevőantenna heading és hullám beesési szögre vett vetületeként képezhetők.

A céltárgy sebessége az 1.2 egyenletből számítható.

Követés

A pontos pozíciók és sebességek ismeretében a következő megállapításokat tehetjük:

- bár a fizikai objektumok sebessége és pozíciója ugrásszerűen nem változhat, a megfigyelés tökéletlenségei miatt mégis láthatunk olyan valid célpontot, ami mégis ilyet tesz
- a megfigyelés esetlensége miatt nem érdemes a megszokott képfeldolgozási eljárásokat alkalmazni, mivel például az esetleges korrelációs problémák esetén a kieső frame-ek teljesen felborítják az ilyen algoritmusokat (például ha levegőt vesz FM esetén a bemondó...)

Ezért jelen pillanatban egyszerű véges impulzusválaszú szűrést végzek: amennyiben a célpont két frame óta folyamatosan látszik, és elmozdulása a térkép szerint reális, összekötöm az észleléseit és feltüntetem a céllistában.

Ábrázolás

Az ábrázolást egy lokálisan hostolt webalkalmazásban végzem, melynek térkép rétege az ingyenes OpenLayers technológián alapul.

2.4. Programszervezési kérdések

2.4.1. Optimalizációs elvek

- **minta be, adat ki**: a GPU-ra csak pontosan egyszer másolódik le a minta csomag futásidőben, és csak a feldolgozott információk jönnek ki, semmiféle átmeneti adat, mivel a sávszélesség véges
- **minden csatornát egyszerre kell feldolgozni**: az adatmozgatás GPU kontextusban különösen drága, érdemes egyszer letölteni a kódot a célprocesszorra, és azt hagyni futni a teljes adatmennyiségen
- **az R mátrix újrahasznosítása**: a referencia csatorna autokorrelációját elég egyszer elvégezni, majd egy optimális kóddal legenerálni a viszonylag nagy mátrixot
- **egyesített egyenletmegoldás**: egyszer kell csak faktorizálni az R mátrixot, utána az igen optimális háromszög megoldó kód végzi csatornánként a feldolgozást
- **minden logikailag egybetartozó FFT-t egy hívás végezzen**: a célprocesszoron futtatott kód paraméterezése is sok idő, egy hívásban kell indítani amit csak lehet
- **csakis inplace FFT végezhető**: a clFFT átmeneti buffert használ *out-of-place* FFT esetén, ami nem fér bele a memóriába, valamint így az erőforrásigény is csökkenthető
- **minden vektorszámítást** (konjugált szorzás, hanning ablakozás) **párhuzamosan** kell végezni, ez a videokártya tervezett működési tartománya

2.4.2. Rendszerarchitektúra

A labor előző passzív radar feldolgozó megoldása BASH szkriptek, python és C programok laza egymás után pipeolt kollekciójaként működött. Ez az összeállítás rendkívül moduláris, azonnal cserélhető egy-egy eleme, ezzel könnyítve a tesztelést vagy a fejlesztést.

Ezzel szemben én egy monolitikus alkalmazás elkészítését tűztem ki célul, mely egyetlen futtatható állományból áll, és további kiegészítő munkaállományokból (OpenCL kernelek, konfigurációs fájl, logfájlok). Ez a felépítés biztosítja az átláthatóságot és az egyszerű mozgathatóságot.

A RADAR összes paraméterét egyetlen konfigurációs fájlban teszem kezelhetővé, melyet a webböngészőben futó grafikus felület is képes befolyásolni, így akár menet közben is változtatva-újraindítva a jelfeldolgozási láncot, ezzel segítve a gyors terepi méréseket. Követelmény a program autonóm működése, így az egész rendszer egyetlen paranccsal felhozható kell hogy legyen. A hibatűrés érdekében supervisor szál figyeli a program futását, és ha eltűnik a feldolgozó szál, megkísérli az újraindítást.

A webes működés érdekében *Joris Vink* [19] Kore.io nevű környezetét használom, mely egy egyszerűen használható platform skálázható WebAPI-k és webalkalmazások készítésére. Írásba foglalt célja a biztonság, skálázhatóság és a gyors fejlesztés támogatása. Mindemellett rendkívül minimalista és puritán C kód hajtja meg, mely moduláris és könnyen bővíthető saját igények szerint. Megelőző projektekben is dolgoztam ebben a környezetben, rendkívül jó tapasztalataim miatt is esett erre a környezetre a választásom. [23] A szoftver a rendkívül permisszív ISC licensz alatt kerül kiadásra, így jogszerűen felhasználható ebben az alkalmazásban.

3. fejezet

Teljesítmény mérés, verifikáció

3.1. Tesztrendszerek, állapot

A tesztelés a következő 3 rendszeren történt:

- Intel "Ivy Bridge" i5 3210M + Intel HD Graphics 4000 + 8G ram (2012) // BSD
- Intel i7 7600U + Intel HD 620 rev2 ultrabook (2017) // Manjaro Linux
- AMD Ryzen 7 1700X + NVIDIA GeForce GT 730 asztali számítógép (2015) // Ubuntu Linux

A processzorra vonatkozó teszteket csak az AMD Ryzen konfiguráción végeztem el, mivel annak volt esélye belátható idő alatt elvégezni. A GPU folyamatot minden platformon futtattam, mely meglepő eredményt hozott.

A jelfeldolgozási rendszer profilozásával az RV mátrix legyártásáig foglalkoztam, a további lépések elhanyagolható időigényűek a nagyságrendekkel kisebb adatmennyiségek miatt.

3.2. Mérési módszerek

3.2.1. CPU

A CPU program sebességét a kétcsatornás futásesetben mértem, a *hyperfine* [30] benchmark eszközzel. A futásidő független a bemeneti binárisok tartalmától, így nyugodtan indíthattam a szűrőt és a mátrix generátort is azonos bemeneti fájlokkal. A mérést az egyetemi passzívradar gépen (Ryzen) végeztem, 10 bemelegítő, és 100 mért futtatással. Ennek eredménye:

```
$ hyperfine -w 10 -m 100 -i './fastwiener.out 20 384 < rxin1.bin > x.bin'
Benchmark #1: ./fastwiener.out 20 384 < ../sample/2/rxin1.bin > temp.bin
  Time (mean ± dev):      1.196 s ±  0.024 s    [User: 1.132 s, System: 0.036 s]
  Range (min ... max):    1.087 s ...  1.230 s    100 runs
$ hyperfine -w 10 -m 100 -i './rv.out 20 10 256 < rxin1.bin > rv.bin'
Benchmark #1: ./rv.out 20 10 256 < ../sample/2/rxin1.bin > rv.bin
  Time (mean ± dev):      88.0 ms ±   1.3 ms    [User: 61.3 ms, System: 20.0 ms]
  Range (min ... max):    84.2 ms ...  90.8 ms    100 runs
```

Tehát a teljes átlagos futásidő ezen a szűrőfokszámon és csatornaszámon $1196 + 88 = 1284$ms.

3.2.2. GPU

A GPU megoldás mérése több utánajárást igényelt. Az OpenCL (és a CUDA) programok indulása alapvetően lassú, mivel a GPU-n futtatott kód előállítása valós időben történik, az első futtatás előtt. Hasonlóan lassú a memóriák kifoglalása, és a clFFT tervek generálása. Ezt az időtartamot nem veszem bele a szigorúan értelmezett futásidőbe, mivel csak egyszer, a program indulásakor okoz késést.

További nehézséget okoz a GPU futtatás aszinkron jellege. A CPU-ról nézve nem igazán látható, mit csinál a kártya. Emiatt építettek az OpenCL és CUDA technológiákba profilozó függvényeket, és hardware támogatást: a kártya menti a kernel indítási idejét, befejeződését, és további metrikákat - ez viszont belső támogatást, eseménykezelést igényel a kliensprogram, tehát a clRadar alkalmazás részéről, valamint a futást is lassíthatja.
Megjegyzés. Az OpenCL profilozás és hívási struktúra kódképre is bonyolult, hosszú sorokat igényel, így a szöveges kódmellékletek alkalmazását tudatosan elvetem.

A totál bruttó futásidőt továbbra is processzoron mérem, mivel ez egyszerűbb, és ott már úgyis szinkronba kerülnek a végrehajtási szálak.

3.3. Memória

A CPU esetén a memóriafoglalás lényegtelen, 2020-ban a RAM bőven elégséges ilyen mennyiségű adat feldolgozására. A GPU megoldás memóriahasználata elfogadható keretek között alakul. Alapvetően a következő képletek szabályozzák a foglalt memóriát, egyéb bufferek mellett:

```
sample     = (B * C) * 2  sizeof(complex float);
correl     = (2 * B * (C-1)) * sizeof(complex float);
doppler    = 2 * correl;
rv          = (SB * D * 2) * (C-1) * sizeof(complex float);
R           = taps * taps * sizeof(complex float);

total_gpu = 2*sample + 3*R + correl + 2 * doppler + 2 * rv;
```

Jelentős számú egyéb kártyás memóriafoglalás van, de ezek mérete az itt jelölt tárolókhoz képest eltörpül, így nem szignifikáns. Ez a vizsgált csatornaszámokra a következőt jelenti: 3.1. táblázat

	2	**4**	**12**
memóriafoglalás GPU-ban	122 MB	282 MB	922 MB

3.1. ábra. GPU memóriafogalás csatornaszámra vetítve

Fontos, hogy egy GPU-ban nem lehet a rendelkezésreálló memóriából tetszőleges méretet foglani, van maximum az egyben foglalható blokkra, ebbe mindenképp bele kell férni. Sajnos az FFT implementáció egyes architektúrákon további buffereket is próbál foglalni, emiatt a $2GB$ videomemóriával rendelkező tanszéki kártyákon már jelentkezett probléma.

3.4. Sebesség

3.4.1. Globális összehasonlítás

A következő táblázatban összegzem a futtatások és mérések eredményeit: 3.2 táblázat (B=20, SB=10, D=256, taps=64)

	CPU2	**GPU2**	**GPU4**	**GPU12**
első futás	–	$0.327sec$	$0.660sec$	$1.997sec$
melegedés	–	-3.05%	-1.51%	-0.60%
bejáratva	$1.045sec$	$0.317sec$	$0.650sec$	$1.985sec$
csatornaidő	$1045ms$	$317ms$	$216ms$	$180ms$
gyorsulás	-	-69.67%	-79.33%	-82.77%

3.2. ábra. Sebességek összehasonlítása és értékelése

Ez az összehasonlítás félrevezető lehet! A CPU pszeudo-párhuzamos megoldása miatt 12 csatorna feldolgozási ideje is csak 1045msec, mivel 16 szálon futhat összesen a program. Így a GPU hátrányba kerül 4 csatorna fölött.

A táblázatból kiolvasható, hogy a 12 csatornás GPU megoldás 82%-kal gyorsabban állítja elő az RV mátrixokat, mint a 2 csatornán, egy magon futtatott CPU megoldás.

Kifejezetten érdekes, hogy a 12 csatornás futtatás mennyivel lassabb a radargépen, mint a fejlesztésre használt 2017-es Intel beépített videokártyán: ott a futtatás csak 850 ms (taps=384-re 890ms).

3.4.2. Részfeladatok

A részfeladatok elemzését 12 csatornára mellékelem a következő táblázatban:

```
WRTE            writedown  30.210 ms   26.66 Gbit/s   96 MB
READ               readup  13.978 ms   26.41 Gbit/s   44 MB
COPY             copy ref   0.523 ms  128.28 Gbit/s    8 MB
COPY             copy ref   0.518 ms  129.51 Gbit/s    8 MB
COPY         surv arrange   5.755 ms  128.27 Gbit/s   88 MB
COPY            w arrange   0.089 ms    0.51 Gbit/s    0 MB
COPY            copy fref   0.520 ms  129.01 Gbit/s    8 MB
FFT       refautocorr_fft   7.090 ms   37.86 Gbit/s   32 MB
FFT         crosscorr_fft  38.947 ms   41.35 Gbit/s  192 MB
FFT          fltrconv_fft  42.484 ms   34.75 Gbit/s  176 MB
FFT             range_fft  76.299 ms   21.11 Gbit/s  192 MB
FFT              velo_fft  67.260 ms   21.95 Gbit/s  176 MB
IFFT     refautocorr_ifft   3.550 ms   75.61 Gbit/s   32 MB
IFFT       crosscorr_ifft  38.947 ms   41.35 Gbit/s  192 MB
IFFT        fltrconv_ifft  38.942 ms   37.91 Gbit/s  176 MB
IFFT           range_ifft  68.207 ms   23.61 Gbit/s  192 MB
KERN     refautocorr_corr   1.868 ms   71.86 Gbit/s   16 MB
KERN       crosscorr_corr  22.695 ms   65.05 Gbit/s  176 MB
KERN        fltrconv_conv  22.703 ms   65.03 Gbit/s  176 MB
KERN    hermitoep_genbase   0.007 ms    1.14 Gbit/s    0 MB
KERN      hermitoep_genmx   0.010 ms   25.60 Gbit/s    0 MB
```

```
KERN            lu factor   0.697 ms    0.75 Gbit/s    0 MB
KERN               filter  11.261 ms  131.11 Gbit/s  176 MB
KERN         blockarrange  11.709 ms  137.55 Gbit/s  192 MB
KERN           range_corr  17.919 ms   82.39 Gbit/s  176 MB
KERN             velo_pad   6.372 ms  252.75 Gbit/s  192 MB
KERN         velo_hanning  17.137 ms   93.99 Gbit/s  192 MB
BLAS        triangular sv   1.767 ms    0.03 Gbit/s    0 MB
FILL           fillsample  16.409 ms   98.16 Gbit/s  192 MB
FILL             fillfref  16.409 ms   98.16 Gbit/s  192 MB
FILL             fillzero  15.045 ms   98.13 Gbit/s  176 MB
*ALL       rv calculation 1986.199 ms    0.19 Gbit/s   44 MB
--------------------------------------------------------------------------------
memops:   44.19 ms   8.07%
memcpy:    7.41 ms   1.35%
fft:     232.08 ms  42.39%
ifft:    149.65 ms  27.33%
blas:      1.77 ms   0.32%
kern:    112.38 ms  20.53%
```

A CPU-GPU memóriaműveletek sebessége elfogadható tartományba esik. A GPU-GPU transzfer sebessége várható módon sokkal jobb, tovább nem javítható: az idő nagyjából 10%-a memóriamozgatással telik. Furcsa, hogy a FILL címkével megjelölt operációk, melyek konstans töltést jeleznek, relatíve milyen lassúak a másoláshoz képest - lehetséges hogy a memóriafogalás számlájára ezeket is statikus másolássá kéne alakítani.

Az FFT/IFFT sebességét befolyásolni tovább nem tudom, de impresszív sebességeket mutat - ennek ellenére a teljes futás 74%-a FFTzéssel telik.

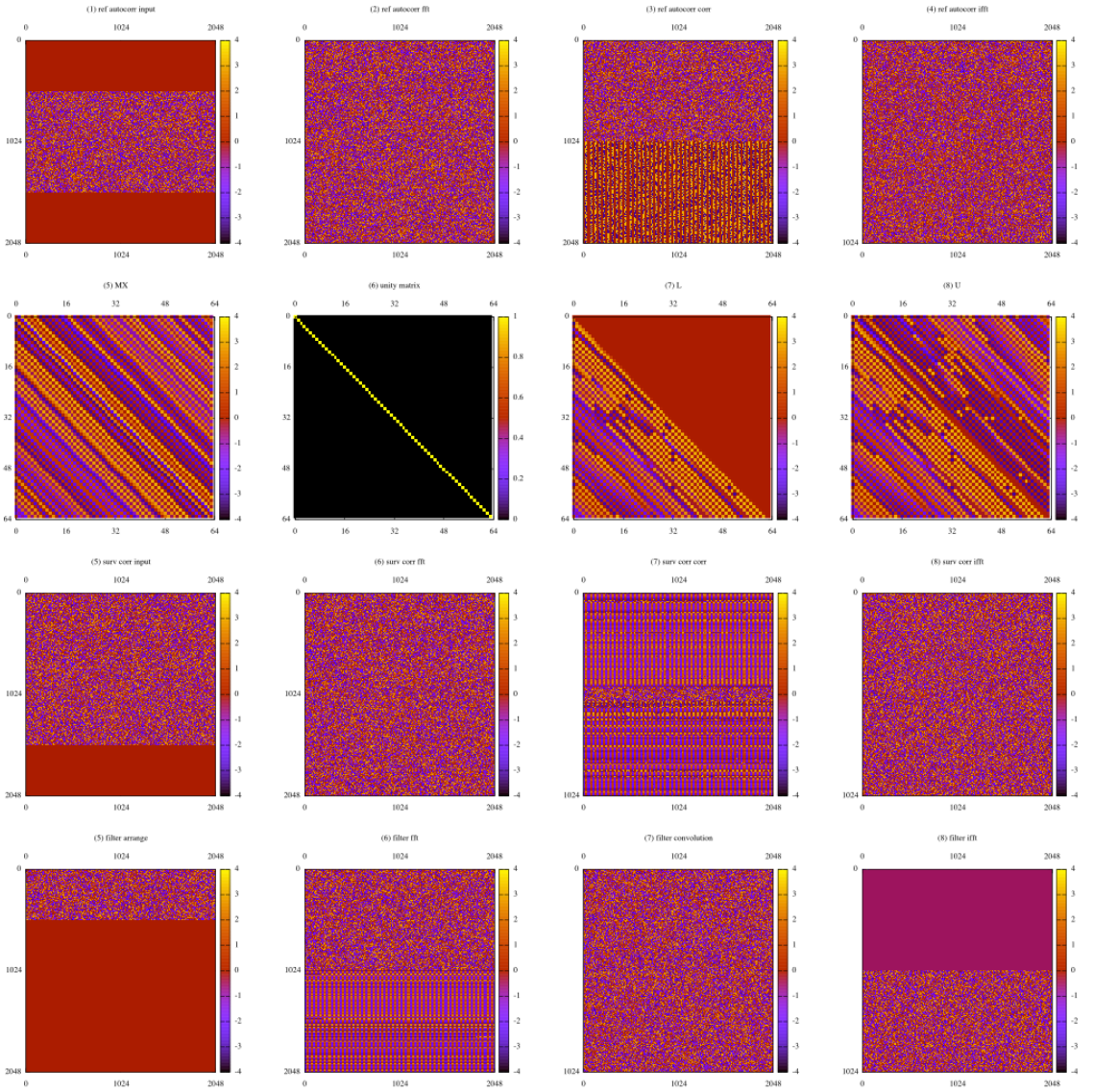
A saját kernelek sebessége igencsak szór, valószínűleg ezek még gyorsíthatók - de a 20% teljes időarány javítása nem biztos hogy összességileg megéri.

3.5. Helyesség és pontosság

3.5.1. Folyamatlogika

Saját fejlesztésű eszközeimmel a transzformációs lépések helyessége és sikeressége vizuálisan jól követhető - főleg a fejlesztés folyamata során. Itt a 2 csatornás feldolgozás ábráit mellékelem. Rendelkezésre áll akár 12 csatornáig is a legenerált vizualizáció, de ezek igen nagy ábrák, darabonként akár 100MB méretű PDF-ek. A szemléltetésre egy *gnuplot* alapú saját szkriptet használtam, mely a komplex elemek abszolútértékének 10-es alapú logaritmusát hőtérképen jelzi.

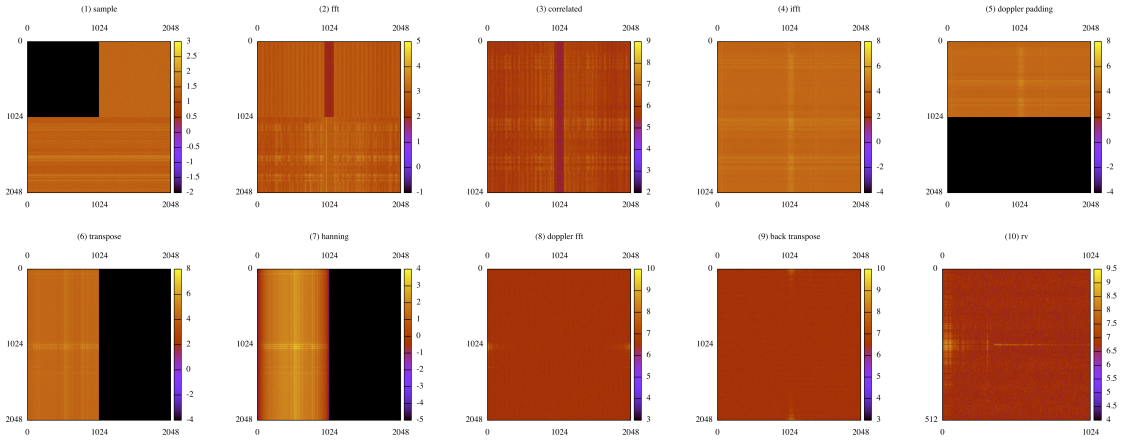
Wiener



3.3. ábra. 2-csatornás feldolgozás folyamata

Az első sor a referencia csatorna autokorrelációjának számítása. A második sorban az R mátrix faktorizációs folyamata van vizualizálva. Érdekes az U mátrix formája: az alsó háromszög nem tökéletesen nulla. Ez a float operációk torzítása miatt van, és nem okoz problémát, mivel később az egyenletmegoldó úgyis csak a megfelelő háromszögeket veszi figyelembe. A harmadik sor a megfigyelési csatornák keresztkorrelációját mutatja. A negyedik sorban a konvolvált referenciát állítom elő.

RV



3.4. ábra. 2-csatornás feldolgozás folyamata

Magyarázat

1. Látszik a referenciacsatorna nullás paddingjének hatása, valamint a megfigyelési csatorna csúsztatott kétszerezése.
2. Látszik az FFT hatása, szimmetria.
3. A szorzás levitte az FFT mély pontjait a megfigyelési csatornába, ez is működik.
4. Az IFFT is működik, kisimult az ábra.
5. A kép fele fekete, ráadásul az alsó, tehát a 0 paddig folyamat működik.
6. A 90 fokos forgás bizonyítja a transzponáló rutin helyességét.
7. Lekékültek az N széles mátrix részsorok végei, ez szép reprezentációja a hann ablak működésének.
8. Az FFT szépen kisimítja a mátrixot.
9. A két oldalsó csúcs szépen elfordult, tehát a transzponálás megint működik.
10. A rendező lépés vizuálisan hasonló megoldást ad ki, mint a C referenciakód, így ez is valószínűleg jó.

3.5.2. Számszerű egyezés

A kimeneti eredmény ellenőrzésére használtam egy másik saját programot, mely PASS-FAIL jelzést szolgáltat két cf32 fájl között, abszolútérték eltérés alapján.

A folyamat maximum abszolút eltérése ezrelékes nagyságrendben alakul, amely megfelelő.

3.6. Terepi mérés

Az elkészült rendszer alkalmas terepi mérések végzésére, valamint mentett régebbi mérések utólagos feldolgozására. A bemutatott mérés a *Liszt Ferenc Nemzetközi Repülőtér*en készült, a következő jelentős konfigurációs paraméterekkel (3.1):

Változó neve	**Értéke**
system.channel_count	4
system.block_size	2^{20}
wiener.tap_count	384
rv.subblock_size	1024
rv.doppler_height	512
hit.hit_count	20
dir.min_hits	3
dir.enable_capon	true
pos.frequency	DVB-T
track.threshold	2

3.1. táblázat. A clRadar program konfigurációs változói

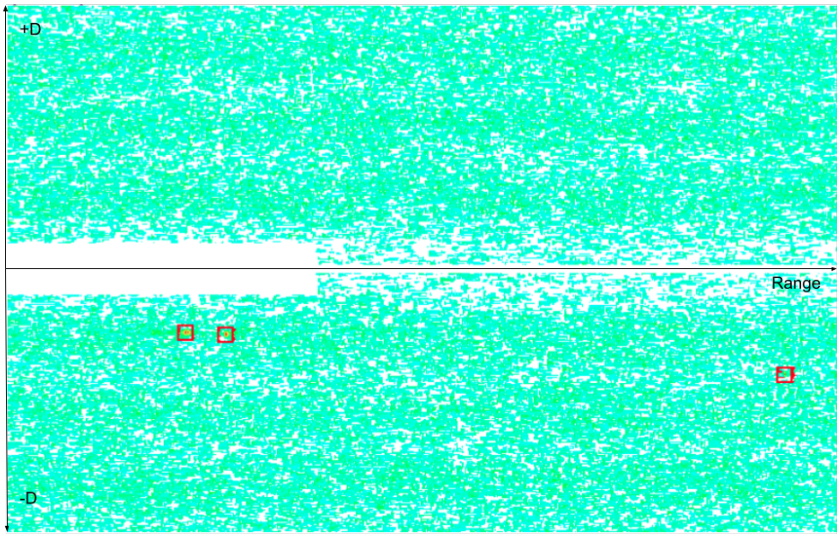
Kiemelem, hogy a mérés paramétereinek beállítása továbbra is jelentős helyismeretet és szakértelmet igényel a fizikai, terepi mérési lokális sajátosságai miatt.

A fizikai konfiguráció 4 logper volt, MRA3/4 elrendezésben, a kimaradó csatorna volt referencia mérésre fenntartva. Az megfigyelési frekvencia a "C-multiplex" sávra került ($610Mhz$), az antennák távolsága $0.75 \cdot \lambda$, az adó pozíciója $(47.491691, 18.979128, 300)$, a radar pozíciója $(47.416431, 19.304206, 120)$, főnyaláb iránya 130 deg.

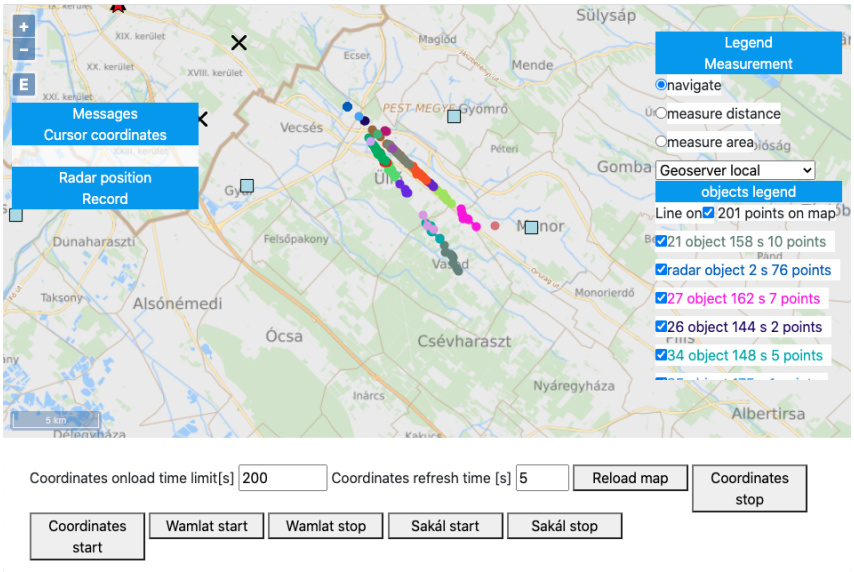
3.5. ábra. Az antennarendszer, és a céltárgy

A 3.6. ábrán látható RV mátrixon jól látszanak az aktuális célpont R-D síkon elfoglalt pozíciói.

A 3.7. ábra illusztrálja a radar kezelői felületét. Megyjegyzés: A két vizuálisan látható elkülönülő vonal igazából azonos céltárgyakat tartalmaz, csak a Budapesti TV adás jellemzői - több adótorony közvetlen közelben, időben eltolva - okozza a célpontok duplázódását.



3.6. ábra. RV mátrix kimaszkolt régiókkal



3.7. ábra. Passzív radarrendszer működés közben

4. fejezet

Összefoglalás

4.1. Eredmények értékelése

A GPU-n futó megoldás működni látszik, és megfelelő hardware mellett nyilvánvalóan jobb megoldás mint CPU párja. Használható mintafeldolgozásra, és megfelelően skálázható nagyobb csatornaszámra.

- Az elért gyorsulás bár impresszív nagyságú, de erősen félrevezető. Feljebb az 1 magos CPU feldolgozást hasonlítottam össze a videokártya teljes erejével. A valóságban a CPU gyorsabb, mivel 16 magon történik a futtatás.
- A labor jelenlegi (2021) feleszereltsége videokártya szempontból nem elégséges igazán hatékony GPU jelfeldolgozáshoz.
- További fejlesztések szükségesek az interfész és a tracker algoritmus irányában, a jobb átláthatóság érdekében.

4.2. Távlati tervek

4.2.1. Optimális CFAR algoritmusok

A RADAR hit-ek detekciója a feltalálással egyidejű probléma. Nem szerencsés, amikor a clutterben elveszik az ellenséges repülő vagy egyéb objektum. Ezért a szakirodalomban rendkívül sokféle *CFAR (Constant False Alarm Rate)* algoritmusról értekeznek, melyek lényege a hasznos hit-ek leválogatása az RD mátrixról.

Ezen algoritmusok optimalizációja GPU futtatásra napjainkban is aktív kutatás részét képezi.

Továbbfejlesztési irányként megfontolandó lenne egy komplexebb CFAR algoritmus beépítése a programba a megfelelő szinten.

4.2.2. Re-moduláció

Szakmai szinten izgalmas kérdés lenne a referencia csatorna szükségességét megszüntetni. Felvetődik a kérdés, ha amúgyis minden csatornában igen magas jelszinttel megjelenik, miért "pazarol el" a felhasználó egy értékes csatornát egy olyan jelért, mely amúgyis rendelkezésére áll.

Érdemes lehet megvizsgálni annak lehetőségét, hogy a megvilágító jelet a megfigyelési antennák segítségével állítsuk elő dekódolással, majd demodulációval.

Sajnos Dr. Dudás Levente kísérletei szerint ez FM rádiós megvilágítással gyakorlatilag lehetetlen, a *cos* függvény erősen nemlineáris tartományán a legkisebb hiba is óriási torzításhoz vezet, mely a korrelációt ellehetetleníti.

Viszont a DVB-T adások vizsgálata eredménnyel kecsegtető. Mivel ez egy OFDM digitális adás, mely egy MP4 streamet tartalmaz, így ha sikerül egy frame-et megfelelően dekódolni a beépített hibajavító kódolás segítségével, a minta eredeti adattartalma tökéletesen ismert, mely újramodulálható: ezzel tökéletes referencia jelünk keletkezne. Erre vonatkozó további ismeretgyűjtést folytatok jelenleg.

4.2.3. Egyéb geometriák támogatása

Jelen pillanatban a program ekvidisztáns, lineáris antennasor támogatására van felkészülve. A tanszéki fejlesztés irányvonalának megfelelően szükség lehet a közeljövőben egyéb antennaelrendezések és csatornakonfigurációk támogatására is, ezesetben a clRadar programon is fejlesztés szükséges.

4.2.4. Soft-reset

Az alapparaméterek módosítása jelenleg csak az egész jelfeldolgozó rendszer leürítésével és újraindításával lehetséges. Érdemes lehet megvizsgálni a teljesen valós idejű, leállást nem igénylő változtatási mód fejlesztésének lehetőségét a gyorsabb terepi hangolás érdekében.

Irodalomjegyzék

[1] NATURE. Radio Detection and Ranging. Nature 152, 391–392 (1943). https://doi.org/10.1038/152391b0

[2] H. D. Griffiths, "From a different perspective: principles, practice and potential of bistatic radar," 2003 Proceedings of the International Conference on Radar (IEEE Cat. No.03EX695), 2003, pp. 1-7, doi: 10.1109/RADAR.2003.1278701.

[3] Al-Sharabi, Khalil. (2012). A SimpleMethod to Derive the Bistatic Tracking Radar System Formulas. AL-Rafdain Engineering Journal (AREJ). 20. 140-149. 10.33899/rengj.2012.47291.

[4] Han, Hui & Wan, Liangtian & Si, Wei. (2014). Performance Optimization of DOA Estimation in Non-Uniform Linear Antenna Array by PSO Algorithm. Applied Mechanics and Materials. 490-491. 451-455. 10.4028/www.scientific.net/AMM.490-491.451.

[5] Eray, Hamza & Temizel, Alptekin. (2020). Performance Analysis of Noise Subspace-based Narrowband Direction-of-Arrival (DOA) Estimation Algorithms on CPU and GPU.

[6] Chen, Zhong & Wang, Jiegui & Tang, Xiwen. (2021). An Improved Estimation Method of Direction of Arrival. 65-69. 10.1109/ICEIEC51955.2021.9463844.

[7] Dr. Seller Rudolf, Nagyfrekvenciás rendszerek - előadások, BME.

[8] Burg, J.P. Maximum Entropy Spectral Analysis. Proceedings of 37th Meeting, Society of Exploration Geophysics, Oklahoma City.

[9] Cadzow J A. Maximum Entropy Spectral Analysis. Encyclopedia of Statistical Sciences. John Wiley & Sons, Inc. 1975:1519–1533.

[10] Nicholas J. Willis. Bistatic Radar. Artech House, 1991

[11] Milovanovic, Vladimir. (2018). On fundamental operating principles and range-doppler estimation in monolithic frequency-modulated continuous-wave radar sensors. Facta universitatis - series: Electronics and Energetics. 31. 547-570. 10.2298/FUEE1804547M.

[12] Don Koks. How to Create and Manipulate Radar Range–Doppler Plots. DSTO–TN–1386

[13] Q. Chen, B. Tan, K. Woodbridge and K. Chetty, "Doppler Based Detection of Multiple Targets in Passive Wi-Fi Radar Using Underdetermined Blind Source Separation," 2018 International Conference on Radar (RADAR), 2018, pp. 1-6, doi: 10.1109/RADAR.2018.8557324.

[14] Pető Tamás. Több csatornás DVB-T alapú passzív radar. TDK dolgozat. https://tdk.bme.hu/VIK/HWBeagy/Tobb-csatornas-DVBT-alapu-passziv-radar [Hozzáférés: 2021]

[15] Palmer, James E.; Harms, H. Andrew; Searle, Stephen J.; Davis, LindaM. (2013). DVB-T Passive Radar Signal Processing. IEEE Transactions on Signal Processing, 61(8), 2116–2126. doi:10.1109/TSP.2012.2236324

[16] Wiener, Norbert (1949). Extrapolation, Interpolation, and Smoothing of Stationary Time Series. New York: Wiley. ISBN 978-0-262-73005-1.

[17] Levinson, N. (1947). "The Wiener RMS error criterion in filter design and prediction." J. Math. Phys., v. 25, pp. 261–278. Durbin, J. (1960). "The fitting of time series models." Rev. Inst. Int. Stat., v. 28, pp. 233–243.

[18] Mark Harris. An Efficient Matrix Transpose in CUDA C/C++. Blog post. Hozzáférés: 2021 https://developer.nvidia.com/blog/efficient-matrix-transpose-cuda-cc/

[19] Joris Vink. Developer. https://github.com/jorisvink

[20] Y. Khmou, S. Safi, M. Frikel. Comparative Study between Several Direction of Arrival Estimation Methods

[21] N. András. Iránymérés adaptív antennarendszerrel. HIRADÁSTECHNIKA 2004/3.: pp. 49-54. (2004)

[22] David Black-Schaffer. https://www.youtube.com/watch?v=Ffn4ebXkViQ&list=PLiwt1iVUib9sUM

[23] https://kore.io/ https://docs.kore.io/4.1.0/

[24] `https://ha7wen.hu`

[25] `http://www.fftw.org`

[26] `http://www.fftw.org/fftw3_doc/Usage-of-Multi_002dthreaded-FFTW.html`, `http://www.fftw.org/fftw3_doc/Advanced-Complex-DFTs.html`

[27] `https://www.khronos.org/opencl/`

[28] `https://github.com/clMathLibraries/clFFT`

[29] `https://github.com/clMathLibraries/clBLAS`

[30] `https://github.com/sharkdp/hyperfine`

Ábrák jegyzéke