VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

LOKALIZACE OBJEKTŮ V REÁLNÉM ČASE

REAL-TIME OBJECT LOCALIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠTĚPÁN RYDLO**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Dr. Ing. DUŠAN KOLÁŘ,**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20014/2016/xrydlo05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Rydlo Štěpán**
Obor: Informační technologie
Téma: **Lokalizace objektů v reálném čase**
Real-Time Object Localization
Kategorie: Modelování a simulace

Pokyny:

1. Seznamte se se zařízením RTL SDR, seznamte se s lokalizací objektů nejen na základě GPS/Galileo/Glonass, prostudujte literaturu ohledně korelace a paralelizace programů i na GPU.
2. Navrhněte způsob zpracování lokalizačního proudu dat ze specializovaného USB zařízení, prozkoumejte možnosti paralelizace a využití GPU pro akceleraci.
3. Po konzultaci s vedoucím implementujte programy pro získávání a korelaci dat z RTL SDR přes USB pomoci jednoho vlákna tak, aby zaznamenával data i správně lokalizoval objekt. Důkladně otestujte.
4. Paralelizujte program z bodu (3) zadání, vyzkoušejte možnosti akcelerace na GPU, otestujte nové varianty, změřte časové náročnosti jednotlivých variant implementace pomoci uložených dat.
5. Zhodnoťte svoji práci, její přínos.

Literatura:

- KAI Borre, DENIS M. Akos, A Software-Defined GPS and Galileo Receiver. c2007, ISBN 978-0-8176-4540-3.
- Dále dle doporučení vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kolář Dušan, doc. Dr. Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zaobírá zpracování radiového signálu a určení polohy přijímače na počítačích s využitím sofrwarově definovaného radia. Cílem této práce je provést paralelizaci programu tak, aby zpracování informací probíhalo v reálném čase. Pro dosažení zpracování v reálném čase bude paralelizace programu probíhat i na grafických procesorech. Obsahem práce je popis dvou na sobě nezávislých možnosti přenosu dat a jeho zpracování.

Abstract

This bachelor thesis purposes solution of a radio signal processing to determine actual receiver position on computers using software defined radio. The purpose of this thesis is to create and parallelize program to achieve real-time processing. To achieve real-time processing we will use GPU. This work contains a description of two independent possibilities of data transfer and processing.

Klíčová slova

lokalizace objektu, TDOA, OpenCL, paralelizmus, RTL SDR, reálný čas

Keywords

object localization, TDOA, OpenCL, paralelization, RTL SDR, real time

Citace

RYDLO, Štěpán. *Lokalizace objektů v reálném čase.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dušan Kolář.

Lokalizace objektů v reálném čase

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Dušana Koláře Další informace mi poskytl Ing. Josef Baier Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Štěpán Rydlo
9. května 2017

Poděkování

Tímto bych chtěl poděkovat panu doc. Dr. Ing. Dušanu Kolářovi za umožnění a pomoci při zvolení postupu pro vypracování této práce. Dále by jsem chtěl poděkovat dalším lidem, kteří na tomto problému se mnou pracovali, za vysvětlení a popis možností implementace.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Pro zjištění polohy objektu existuje řada metod a systémů. Rozvojem bezpilotních letadel – dronů pro využití např. střežení objektů, monitorování dopravní situace apod., vyvstává potřeba zjišťování jejich přesné polohy v reálném čase. Jedna z možností je využití systémů GNSS (GPS, GLONASS a v budoucnu GALILEO), kde je závislost na provozovateli těchto systémů, nebo použít systém zcela nezávislý.

Současný rozvoj elektroniky a informačních technologií dávají možnost tyto nezávislé systémy realizovat. Řada těchto principů je ověřena a ověřována simulací např. v programovacím prostředí Matlab, ale problém nastává při realizaci požadavku lokalizace objektu v reálném čase.

Cílem práce je navrhnout zpracování lokalizačního proudu dat z rádiového přijímače připojeného přes USB. Přijatá data slouží pro lokalizaci objektů v reálném čase. Tento systém pro lokalizaci objektů vychází z principu tří nebo více vysílačů, jejichž přesnou polohu známe a které budou ve stejných časových intervalech vysílat různý kód. Úkolem je pomocí korelace vypočítat časové rozdíly mezi příjmem signálů od jednotlivých vysílačů a na základě těchto informací vypočítat vlastní polohu. Existuje řada možností, jak přenášet potřebné informace pomoci rádiových vln a také řada možností implementace, jak data získávat. V práci bude popsána implementace dvou na sobě nezávislých systémů. Tyto systémy budeme rozdělovat podle typu synchronizace vstupního signálu s vytvořenými referenčními sekvencemi na statický a průběžný.

Pro úspěšné splnění cíle bakalářské práce bude primárním úkolem prostudovat literaturu ohledně korelace a paralelizace programů. Následně realizovat program pro korelaci pomocí jednoho vlákna, aby zaznamenával data i správně lokalizoval objekt. Při zpracování velkého množství dat, které je tvořeno rádiovým signálem, bude zapotřebí využít celé výpočetní kapacity, které nám zařízení umožňuje. Proto je zapotřebí program paralelizovat. Pro vykonání programu se ve výpočetních zařízení využívá procesorů CPU, na kterých se často provádí i paralelizace. Nicméně výkon CPU nemusí pro práci v reálném čase stačit, a tak můžeme pro paralelizaci využít také grafických procesorů GPU. Dříve byly využívány grafické procesory pouze pro výpočet zobrazení na monitor, projektor nebo jiná zařízení. V dnešní době však lze využít grafické procesory i pro zpracování jiných dat.

Výsledkem by mělo být ověření vytvořených programů na různých hardwarových platformách a stanovení požadavků pro realizaci systému. Výsledkem aplikace potom bude výpis lokalizačních údajů, tedy polohy na standardní výstup. Poloha bude počítána v reálném čase, proto výsledná poloha zařízení bude vypisována při každém výpočtu.

Kapitola 2

Technologie lokalizace objektů

Technologie pro lokalizaci objektů, co se týče přesnosti a dostupnosti, dosáhla velkého pokroku v první polovině minulého století. V této době se začalo využívat rádiových vln, tedy elektromagnetického záření pro určování polohy. Do té doby se pro určení polohy využívalo hvězd, uměle vytvořených orientačních bodů a magnetických kompasů. K určení polohy se používal astroláb (později sextant), který pomáhal určovat polohu podle hvězd a Slunce pomoci triangulace. Tato metoda ale nedosahovala tak vysockých přesností, byla časové náročná a závislá na tom jestli jsou hvězdy vidět.

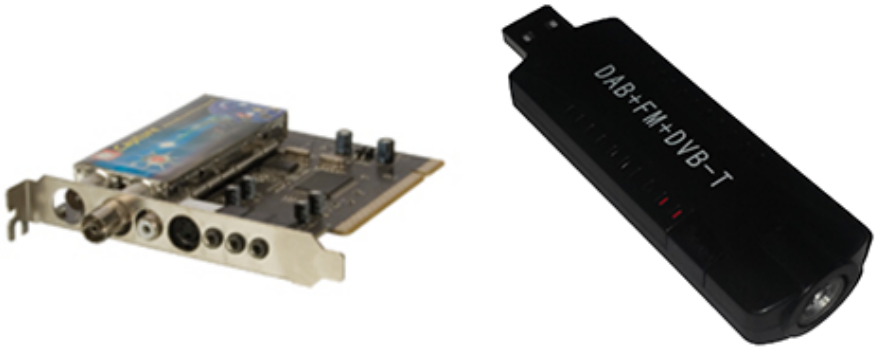
Jedním z prvních systémů pro lokalizaci objektů pomoci elektromagnetického záření byl systém RADAR (Radio Detecting And Ranging). Vysílá svazky vln a následně přijímá vlny odražené od objektů. Vzdálenost předmětu se určuje pomoci interference vysílaného a odraženého signálu. Dalším systémem, který pro navigaci vzniknul, byl systém LORAN. Tento systém využívá pozemních vysílačů na rozdíl od systému GPS, který využívá družic jako vysílače signálu z oběžné dráhy Země.

2.1 Princip činnosti softwarových rádií

Softwarové rádio, označováno jako SDR (Software Defined Radio), je rádiová technologie, která umožňuje přímé zpracování signálu v digitální podobě. Softwarové rádio je tedy program, který popisuje chování jednotlivých analogových nebo digitálních prvků rádiového přijímače [7]. Umožňuje nám tedy zpracovávat digitální, ale i analogově modulovaný signál. Program ale přijímá a zpracovává data pouze v digitální formě. Proto musíme použít přijímač, který nám převádí analogový signál do digitální podoby, tedy spojitý signál na diskrétní. Konstrukce přijímače musí obsahovat tuner, který se naladí na určitou frekvenci a přijímá signál. Tunery často následně vysílají analogová data pomoci svých rozhraní. Proto je zapotřebí mít v přijímači i A/D převodník, který nám signál vzorkuje podle stanovené vzorkovací frekvence a posílá digitální data [8]. Obrázek 2.1 zobrazuje různé typy přijímačů.

Jednotlivé přijímače se můžou lišit hardwarovým složením a dosahovat tak vyššího kmitočtového rozpětí, nebo potlačují více šumu. Dále mohou obsahovat různé hardwarové filtry, jiné digitální převodníky nebo jiné rozhraní (USB, PCI-E). Podle druhu přijímače je tedy potřebné nastavit rozhraní mezi softwarovým rádiem a přijímačem. Jedná se o nastavování parametrů pro přijímač a typ přijímaných údajů (I/Q vzorky, demodulovaná data, ...).

Softwarové rádio je tedy program, který zpracovává signál získaný přijímačem do námi požadované podoby. Velkou výhodou softwarových rádií je možnost upravovat zpracování signálu pouhou úpravou programu. Lze tedy pomoci jednoho přijímače demodulovat různé



Obrázek 2.1: Typy přijímačů rádiového signálu

typy signálů bez úpravy přijímače. Další výhodou je tedy i cena, jelikož nepotřebujeme hardwarové filtry, detektory a další prvky, protože všechny tyto prvky řeší právě software. Softwarová implementace těchto prvků nám také umožňuje tvořit různé kombinace, nebo prvky, které by hardwarově byly složitě řešitelné. Pro zpracovávání signálu v reálném čase je zapotřebí vyššího výpočetního výkonu. Jednou z možností zvýšení využití výkonů počítače je paralelizace. V 7dnešní době však počítače dosahují dostatečných výkonů, proto v některých případech není nutné ani provádět paralelizaci softwaru.

2.2 Rádiový přenos

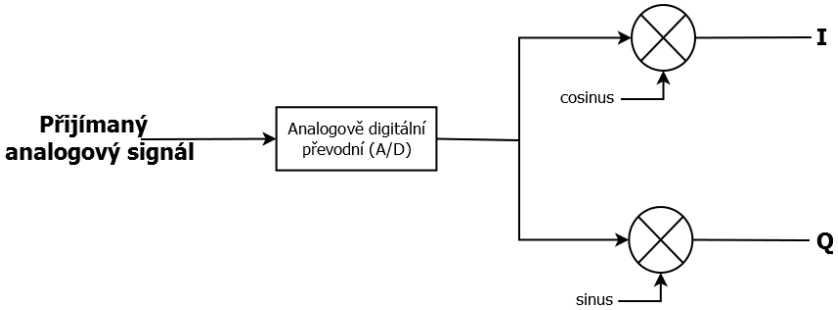
Jak již bylo uvedeno v úvodu této kapitoly, v dnešní době se pro lokalizaci využívá rádiových vln, jejichž přenosové médium je libovolné prostředí, které je schopné přenášet elektromagnetické vlny: vzduch, vakuum,... Důležitým parametrem rádiových vln je jejich frekvence. Rádiové vlny se tedy používají pro přenos informací mezi vysílači a přijímači. Rádiové vlny nepotřebují uměle vytvořené přenosové medium. Jednoduchost šíření signálu také umožňuje lepší dostupnost pro komunikaci.

2.2.1 Přenos dat

Součástí této práce bude popis a implementace dvou na sobě nezávislých systémů pro lokalizaci. Každý ze systémů využívá pro přenos dat změnu fáze vyslaného signálu. Data, která budeme zpracovávat z přijímače jsou I/Q vzorky [6]. Tyto vzorky vznikají vytvořením hodnoty cosinus pro I a sinus pro Q vzorky z přijímaného signálu. Obrázek 2.2 popisuje implementaci digitálního signálového procesoru společně s A/D převodníkem. Vstupem je analogový signál přijímaný tunerem přijímače.

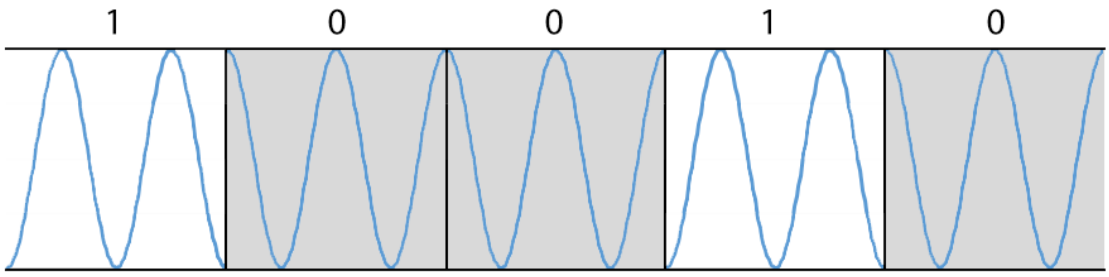
Přenos dat pro statický systém

Pro rádiový přenos k systému statického porovnávání jsme zvolili digitální modulaci BPSK. Tedy pomoci rádiového signálu jsou přenášené jednotlivé bity určitého datového toku. Bity jsou označené jedničkou nebo nulou v obrázku 2.3. Každý bit se skládá z více vzorků. Počet vzorků na jeden bit je určen dobou vysílání jednoho bitu a vzorkovací frekvencí přijímaného signálu. Existuje mnoho digitálních modulací, ale v navigaci se převážně používá modulace typu BSPK (Binary-Phase Shift Keying). Jedná se o jednoduchou modulaci, která přenáší



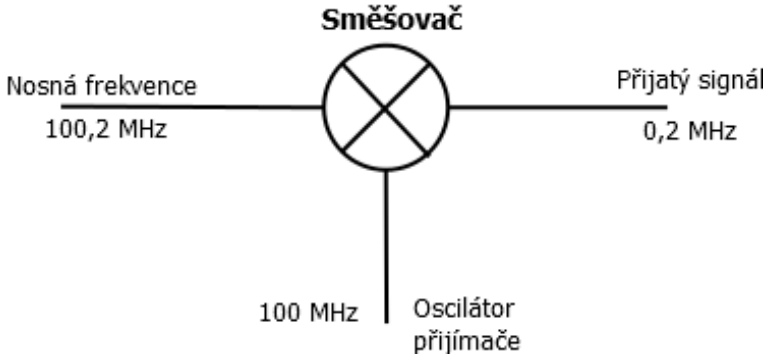
Obrázek 2.2: Získání digitálních vzorků I a Q z analogového signálu

jeden bit, 0 nebo 1. Pomoci změny fáze signálu dochází ke změně jednoho bitu. Přiklad BPSK je možné vidět v obrázku 2.3. Pro přesné demodulování, tedy převod signálu zpět na bity, musíme znát modulační frekvenci. Modulační frekvence určuje v tomto případě délku amplitudy digitální modulace typu BPSK [22].



Obrázek 2.3: Ukázka signálu pro modulaci BPSK

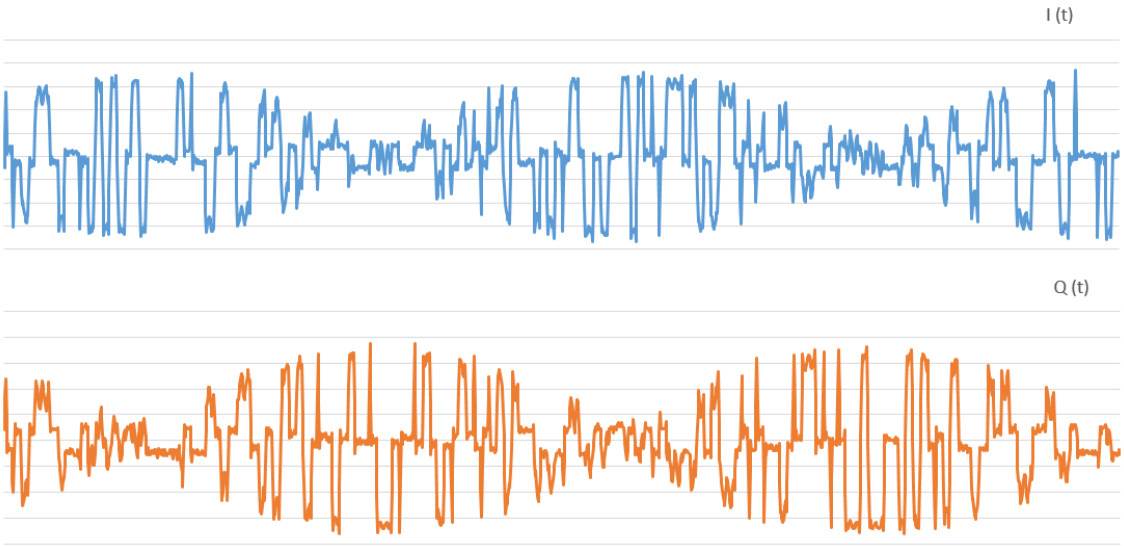
Data jsou přenášená pomoci změny fáze. Kdybychom nastavili přesnou frekvenci, na které jsou data vysílána, získávali bychom demodulovaná data. Přesnou frekvenci je však složité naladit z důvodu ovlivnění frekvence vlivem prostředí nebo Dopplerova jevu. Na obrázku 2.4 můžeme vidět směšovač, který se používá po přijímaní signálu. Nastavená frekvence na přijímači je frekvence lokálního oscilátoru. Při následném nastavení přijímače pro přijímaní těchto dat musí být frekvence nastavena na rozdíl nosné frekvence a modulační frekvence tak, aby docházelo k přijímání modulovaných dat.



Obrázek 2.4: Příklad směšovače, který je umístěný v RTL SDR zařízení

Přenos dat pro průběžný systém

Tento systém používá přenos stejný tomu předchozímu. Pro přenos dat využívá změnu fáze, nicméně jeden bit této zprávy je kratší. Jedná se tedy o stejný způsob přenosu, ale pro získání lokalizačních dat musíme zvolit jinou metodu. V tomto případě se snažíme naladit frekvenci přijímače, tak abychom přijímali demodulovaná data. Nicméně vlivem prostředí nebo Dopplerova jevu, vzniká odchylka mezi přijímaným signálem a nastavenou frekvencí lokálního oscilátoru ve směšovači, který je na obrázku 2.4. Tato frekvence je označována jako mezifrekvence (Intermediate Frequency). Vlivem tohoto rozdílů vzniká parazitní amplitudová modulace, které se musíme přizpůsobit. Grafický průběh přenosu lokalizačních sekvencí pomoci této modulace můžeme vidět na obrázku 2.5. Na obrázku jsou dva grafy, zobrazující průběh signálu ve vzorcích I a Q.



Obrázek 2.5: Graf přenášených dat pro průběžný systém

Zobrazený signál se skládá ze dvou signálů vysílaných dvěma vysílači, které pomocí změny fáze posílají hodnotu 1 nebo -1. Na obrázku je patrné, že signál dosahuje také hodnoty 0. Tato hodnota vzniká vlivem vzájemného rušení obou signálů. Každý z vysílačů posílá opačnou fázi a tak dojde k vzájemnému vyrušení.

Na obrázku 2.5 je zobrazen průběh jednoho přijímaného signálu, který tvoří hodnoty cosinus (I) a sinus (Q). Ve chvíli kdy jeden ze signálů dosahuje útlumu, druhý využívá celou šířku a nedochází tak ke ztrátě informací. Po každém útlumu dochází také ke změně fáze, vlivem průběhu křivky sinus a cosinus.

2.2.2 Multiplex

Pro správné určení polohy je zapotřebí, jak jsme si řekli, přijímat signál od více vysílačů. Vysílače musí tedy mít domluvený způsob posílání signálu tak, aby nedocházelo k vzájemnému rušení. K tomu slouží proces multiplexování, jehož cílem je nejefektivněji využít přenosové médium. Je několik druhů multiplexu a jedním z nich je technika TDMA (Time Division Multiple Access) [15]. Tento typ multiplexu se moc nepoužívá pro lokalizaci, protože je komplikovaný na realizaci přijímače, přesněji na přesné nastavení času. Každý vysílač v této metodě by vysílal stejné kódy na stejné frekvenci, ale v jiný čas.

Druhá možností pro multiplex je technika FDMA (Frequency Division Multiple Access) [15]. V této technice mají vysílače, které by se mohli vzájemně rušit, jinou frekvenci. Všechny vysílače tak mohou posílat stejné kódy. Jeden lokalizační systém tak používá více frekvencí. Pro každou z těchto frekvencí bychom museli mít zvlášť přijímač.

Posledním z technik multiplexování je CDMA (Code Division Multiple Access) [15]. Tato technika požívá pouze jednu frekvenci, na které vysílají všechna zařízení najednou. Každé zařízení vysílá jiný kód. Podle toho je následně možno rozeznat, který z vysílačů posílá jakou zprávu. Techniku CDMA používá systém GPS a připravující se systém Galileo.

2.2.3 Sekvence a korelace

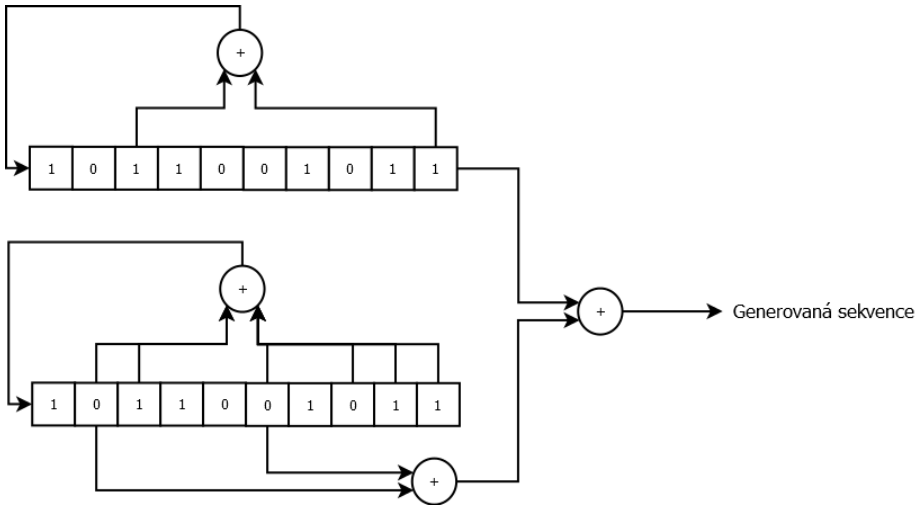
Každý z vysílačů vysílá vlastní pseudonáhodnou sekvenci PRN (Pseudo Random Noise). PRN se chová jako šum a využívá se k rozprostření spektra vysílaného signálu. Sekvenci lze však reprodukovat v přijímači, což je hlavní důvod využití těchto sekvencí. V systému CDMA každý z vysílačů vysílá rozdílné, ale předem stanovené sekvence na stejné frekvenci. Sekvence se v tomto projektu využívá jako dálkoměrný kód. Využívá rozdílné doby šíření signálu od zdroje v závislosti na vzdálenosti. Rozdíly mezi sekvencemi mohou být i v její periodě. Jednou z možných sekvencí je Barkerův kód [24], jehož 7bitovou formu jsme požívali v tomto projektu pouze pro testování.

Sekvence, která je použita pro tento projekt, se nazývá Goldův kód nebo také Goldova sekvence. Na obrázku 2.6 je zobrazen způsob generování této sekvence [20]. Délku sekvence lze vypočítat pomoci vztahu 2.1 [27], kde proměnná n velikost posuvného registru, a generují se pouze dva znaky kódu 1 nebo 0. Oba posuvné registry musí být stejné velikosti, do kterých vkládáme vypočítaný nový bit podle zadaných polynomů. Jedná se o bitové registry, takže výsledky dosahují hodnot 1 a 0.

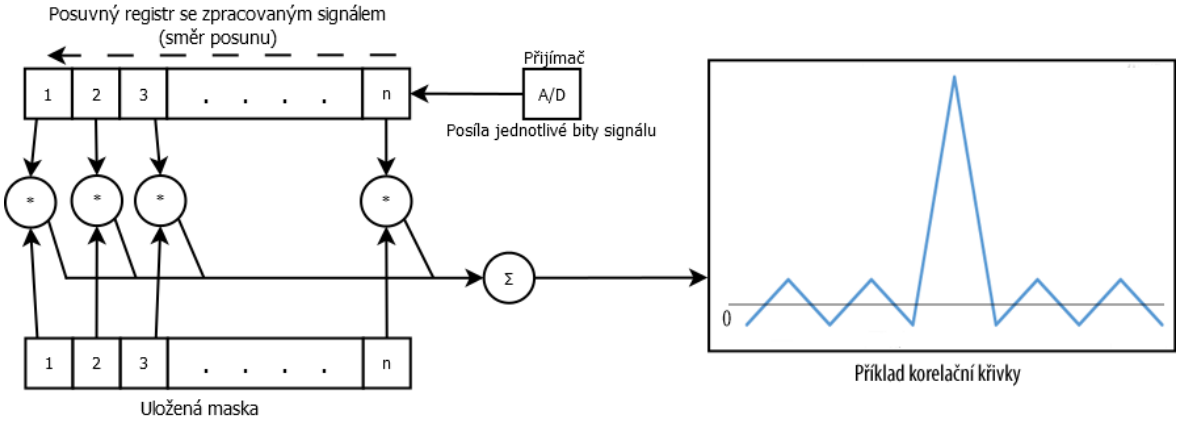
$$delkasekvence = 2^n - 1 \tag{2.1}$$

Generátor se tedy skládá ze dvou posuvných registrů. Sekvenci můžeme měnit nebo posouvat. Pro posunutí vygenerované sekvence změníme počáteční stav jednotlivých registrů. Posloupnost v sekvenci zůstane stejná, bude pouze posunuta. V případě, že dva vysílače vysílají stejnou posloupnost různě posunutou, v reálném případě by to pouze vytvořilo chybu při určení polohy. Pomoci posunu sekvence můžeme simulovat různé vzdálenosti vysílače od přijímače. Druhou z možností je změna pozic, ze kterých sčítáme jednotlivé bity, které dostáváme z druhého posuvného registru na obrázku 2.6. Jedná se o bity, jejichž součet pokračuje na sčítačku pro vytvoření znaku dané frekvence. Pří změně pozic bitů z druhého posuvného registru v generátoru dochází ke změně generované posloupnosti znaků. Tento typ sekvence používá například systém GPS.

Ve vysílači je generován Goldův kód, podle obrázku 2.6, a je přenášen pomoci rádiového signálu. Tento signál je přijímačem přijat a demodulován na hodnoty 1 a -1. V přijímači není jasné, která fáze signálu nese jakou hodnotu což je problém, který bude řešit korelátor. Demodulovaný signál je posílán do korelátoru, obrázek 2.7, který porovnává tento signál s referenční uloženou sekvencí. Korelační maximum [20] nastane ve chvíli, kdy je posuvný registr se zpracovávaným signálem totožný s uloženou maskou. V případě, že korelační maximum dosahuje záporné hodnoty, byly přehozeny fáze pro signál BPSK a musí dojít k přepólování. Řešením je změnit znaménko referenční sekvence či přijímaného signálu, nebo vyhledávat i záporná maxima v korelační křivce. Je tedy nutné masku mít uloženou v přijímači, nebo jí generovat.



Obrázek 2.6: Příklad generátoru znaků Goldovy sekvence



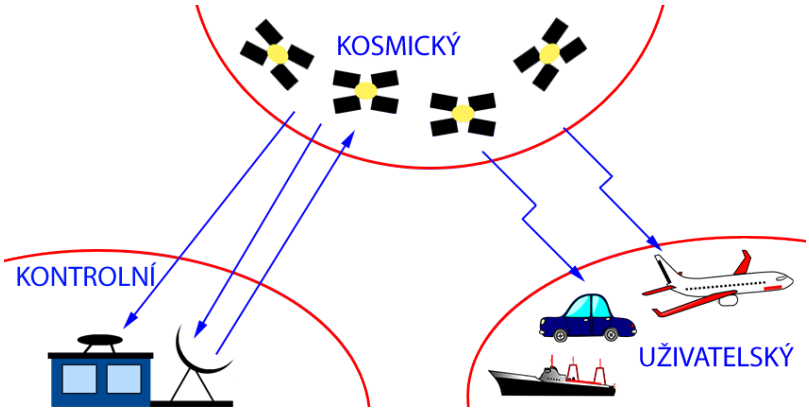
Obrázek 2.7: Příklad korelátoru Goldovy sekvence

2.3 Metody rádiové lokalizace objektů

Každá z metod pro zjištění polohy potřebuje orientační body. Body mohou být vytvořené přírodou, jako například hory, nebo jako v našem případě uměle vytvořené. Konkrétně se při rádiové lokalizaci jedná o vysílače vysílající určitý signál, který je popsán v předchozí kapitole, na předem dané frekvenci. Tyto vysílače se nazývají referenčními body, protože známe jejich přesnou polohu. Pro určení přesné polohy je zapotřebí získat signál minimálně ze tří vysílačů. Pokud chceme znát i výšku, ve které se nacházíme, potřebujeme signál minimálně ze čtyř vysílačů.

2.3.1 GNSS a jejich struktura

GNSS (Global Navigation Satellite System) je zkratka pro globální družicový systém pro určení polohy, které por určení polohy využívají družic obíhajících kolem země. Výhodou družicových systémů, proti pozemním systémům, je jejich dostupnost. Družice obíhají kolem země, takže signál z nich jsme schopni přijímat v místech, kde je viditelná obloha bez větších překážek. Družicové systémy jsou ale dražší na tvorbu, provoz a také signál z družic je ovlivňován při průchodu atmosférou planety. Všechny navigační systémy se dělí do tří částí neboli segmentů. Každý ze segmentů je zobrazen na obrázku 2.8.



Obrázek 2.8: Zobrazeni struktury navigačních systémů typu GNSS

První je řídící nebo také kontrolní segment. Tento segment se většinou skládá z více pozemních stanic. Tyto pozemní stanice se dělí na jednu hlavní řídicí a další monitorovací stanice. Hlavní řídící stanice má za úkol spravovat jednotlivé družice. Upravuje jejich pohyb na předem určených orbitách a také monitoruje signál, který vysílají družice a vyhodnocuje, případně opravuje časové odchylky. Další úlohou je sledování technického stavu družic a jejich údržba. Hlavní řídící stanice komunikuje s družicemi přes monitorovací stanice. Tyto stanice jsou rozmístěny na různých místech na zemi tak, aby bylo možné sledovat co nejvíce družic po co nejdelší dobu. Monitorovací stanice komunikují s jednotlivými družicemi a slouží k vytvoření spojení s hlavní řídicí stanicí.

Dalším segmentem je samostatný družicový systém, kterému se říká kosmický segment. Součástí tohoto segmentu jsou všechny údaje o pohybu družic, tedy popis jejich orbit. Každá družice se skládá z přijímače, aby mohla řídící stanice spravovat danou družici. Dalším prvkem družic je vysílač, na kterém družice vysílají předem stanovené signály. Družice obsahují také atomové hodiny pro určení přesného času, který je v některých systémech potřebný pro přesnou lokalizaci.

U pozemních navigačních systémů je struktura podobná jako u GNSS. Rozdíl mezi těmito systémy je, že pozemní systém nepoužívá družice, ale vysílače rozmístěné po zemi. Kontrolní segment v těchto systémech dále zůstává, jen nemusí být v tomto segmentu observatoř, ale pouze skupina lidí spravující daný systém.

2.3.2 Systémy GNSS

Jedním z globálních satelitních navigačních systémů je právě systém GPS (Global Positioning System). Tento systém je nástupce systému TRANSIT, někdy také NAVSTAR (Navy Navigation Satellite System), který byl spouštěn a používán od roku 1964 Spojenými státy Americkými. Byl používán převážně pro vojenské námořní účely. Tento systém byl nahrazen systémem GPS a je k dispozici i pro civilní potřeby, takže pro navigaci jej může používat každý. Systém používá pro multiplex techniku CDMA. Systém GPS se skládá minimálně z 24 družic, aby docházelo k maximálnímu pokrytí. Družice obíhají kolem země po 4 téměř kruhových drahách se sklonem 55°. Těmto drahám se také říká efemeridy.

Dalším ze systémů je ruský GLONASS (Global'naya Navigatsionnaya Sputnikovaya Sistema). Systém lze také používat pro civilní potřeby a, na rozdíl od systému GPS, používá pro multiplex techniku FDMA. Další z rozdílů mezi těmito systémy je v efemeridách. Družice systému GLONASS obíhají pouze po 3 drahách se skonem 64,8° od rovníku. Vyšší úhel sklonu umožňuje lepší dostupnost tohoto systému ve vyšších zeměpisných šířkách, tedy blíže k severnímu nebo jižnímu pólu. Systém má tedy lepší pokrytí. Družice dále také obíhají blíže k zemi, aby nedošlo ke střetu.

Galileo je dalším navigačním systémem, jehož výstavbu zajišťuje Evropská unie, prostřednictvím ESA (European Space Agency). V dnešní době ještě není navigační systém Galileo plně funkční, ale okolo země již obíhá několik družic tohoto systému. Tento evropský navigační systém bude nezávislý na dříve zmíněných systémech a měl by být primárně civilním systémem. Výhodou tedy je, že tento systém bude stále přístupný a nebude omezován pro vojenské užití. Oběžné dráhy družic systému Galileo jsou podobné předchozím dvou. Používají 3 oběžné dráhy se sklonem 56° od rovníku. Doba jednoho oběhu družice kolem země je u každého systému jiná. Hodnoty oběhu u všech těchto systémů jsou okolo 12 hodin.

2.3.3 Metody lokalizace objektů pomoci GNSS

První metodou pro určování polohy, kterou systémy GNSS používaly, byla Dopplerova metoda. Tato metoda počítá polohu způsobem, že každá z družic vysílá signál s určitým kmitočtem. Tento signál obsahuje časovou značku a to, kdy byl signál vyslán a vzdálenost mezi družicemi. Signál přijímaný uživatelem má v důsledku Dopplerova jevu jiný kmitočet. Pokud přijímaný signál má nižší kmitočet, družice se od uživatele vzdaluje. Pokud přijímaný kmitočet, je vyšší znamená to, že družice se přibližuje. Na základě těchto získaných údajů je možné vypočítat polohu. Dopplerova metoda nedosahovala vysoké přesnosti.

Metoda používaná dnes GNSS systémy je takzvaná dálkoměrná metoda. Poloha přijímače se určuje změřením vzdálenosti mezi přijímačem a jednotlivými družicemi. Vzdálenost se měří určením doby, která je potřebná k překonání vzdálenosti mezi přijímačem a družicí. Pokud známe tuto dobu a polohu jednotlivých družic, poloha se počítá jednoduše pomoci čtyř rovnic o čtyřech neznámých. Potřebujeme nejen znát přesnou polohu (x, y, z), ale i čas. Každá rovnice bude pro jednu družici a její obecný tvar popisuje rovnice 2.2. Tedy x, y, z jsou souřadnice družice, které jsou posílány na jiné frekvenci, a c je rychlost šíření rádiových vln ve vakuu, která je stejná jako rychlost světla.

$$\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = c \tag{2.2}$$

Pro každou družici nám vznikne jedna rovnice. Proto pro určení polohy je potřeba signál z minimálně tří družic. Dálkoměrná metoda se dále dělí na aktivní a pasivní. Měření pomoci aktivní dálkoměrné metody probíhá způsobem dotaz – odpověď, proto musí být uživatel vybaven vysílačem. Pozemní řídící stanice pošle dotaz konkrétnímu uživateli, který následně odpovídá zpět pomoci družic. Řídicí stanice potom vyhodnotí zpoždění odpovědi na jednotlivých družicích a vypočítá polohu uživatele, kterou mu následně posílá.

Při použití pasivní dálkoměrné metody výpočet polohy provádí uživatelské zařízení. Při pasivní metodě je tedy zapotřebí vyšší výpočetní výkon uživatelského zařízení, které zjišťuje svojí polohu. Uživatelské zařízení potřebuje pouze přijímač, protože v této metodě uživatelské zařízení nevysílá žádný signál. Jednou z výhod je neomezený počet uživatelských zařízení, protože polohu nevypočítává jedna řídící místnost s omezenou kapacitou. U pasivních systémů na rozdíl od aktivních, řídící stanice nezjistí polohu jednotlivých uživatelských zařízení. Družice tedy pouze posílají zprávu se svým časem a polohou. Uživatelské zařízení musí mít nastavený přesný čas. Čas je tedy další neznámou a proto potřebujeme signál z minimálně tří družic pro zjištění polohy. Pokud bychom chtěli znát i výšku (z) potřebujeme signál ze čtyř nebo více družic.

2.3.4 Lokalizace objektů použitím metody TDOA

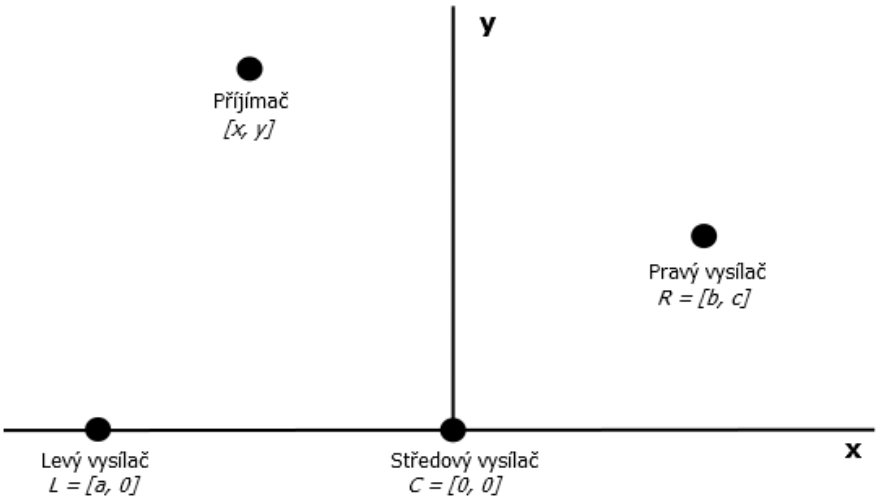
Pro lokalizaci objektu lze použít metodu TDOA (Time Different of Arrival). V této metodě musí každý z vysílačů vysílat sekvenci ve stejný čas. Časový rozdíl mezi přijatými sekvencemi nám určí rozdílnou vzdálenost mezi přijímačem a jednotlivými družicemi. Vysílače mohou používat pro multiplex technologii FDMA nebo CDMA. Technologii TDMA nelze použít pokud chceme vypočítávat polohu metodou TDOA, protože musí docházet k odesílání sekvence ze všech vysílačů ve stejný čas.

Výpočet polohy přijímače

Pro určení polohy požíváme v tomto projektu vysílače, které jsou stacionární. To znamená, že nemění svojí polohu a jejich poloha je předem známa, stejně jako kód, který vysílají. V případě zjištění rozdílu dvou časů lze zjistit hyperbolu, kde se přijímač vyskytuje a pokud přijímá signál ze tří vysílačů lze určit dva body. Pro určení polohy se používá časově-hyperbolická metoda [2]. Postup výpočtu polohy objektu v dvourozměrném prostoru si popíšeme na obecném příkladu. Na obrázku 2.9 je zobrazeno obecné zadání.

Při použití metody TDOA, známe rozdílný čas příjetí signálu z jednotlivých vysílačů. Tento čas pro jednotlivé vysílače nám popisuje rovnice 2.3, 2.4, kde rozdíl časů levého a pravého vysílače jsou odvozené od středového vysílače. Proměnná cl označuje rychlost světla.

$$\begin{aligned} \tau_L &= \frac{1}{cl}(\sqrt{(x-a)^2 + (y-0)^2} - \sqrt{(x-0)^2 + (y-0)^2} \\ \tau_L &= \frac{1}{cl}(\sqrt{(x-a)^2 + y^2} - \sqrt{x^2 + y^2}) \end{aligned} \tag{2.3}$$



Obrázek 2.9: Konfigurace vysílačů pro určení polohy metodou TDOA

$$\begin{aligned} \tau_R &= \frac{1}{cl}(\sqrt{(x-b)^2+(y-c)^2} - \sqrt{(x-0)^2+(y-0)^2} \\ \tau_R &= \frac{1}{cl}(\sqrt{(x-a)^2+(y-c)^2} - \sqrt{x^2+y^2}) \end{aligned} \tag{2.4}$$

Jedná se o nelineární rovnice, pro jejichž řešení si nejdříve zavedeme tyto proměnné 2.5.

$$L = \tau_L * cl \qquad R = \tau_R * cl \qquad k^2 = x^2 + y^2 \tag{2.5}$$

L a R nám přestavují vzdálenost, která je určena časovými rozdíly. Proměnná k určuje vzdálenost cíle od středního stanoviště, hodnoty x a y jsou tedy souřadnicemi cíle. Následně tedy určíme hodnotu x.

$$\begin{aligned} L &= \sqrt{(x-a)^2+y^2} - \sqrt{x^2+y^2} && \Rightarrow \\ L+k &= \sqrt{(x-a)^2+y^2} && \Rightarrow \\ (L+k)^2 &= (x-a)^2+y^2 && \Rightarrow \\ L^2+2Lk+k^2 &= x^2-2xa+a^2+y^2 && \Rightarrow \\ L^2+2Lk+k^2 &= k^2-2xa+a^2 && \Rightarrow \\ L^2+2Lk &= -2xa+a^2 && \Rightarrow \\ x &= \frac{a^2-2Lk-L^2}{2a} \end{aligned} \tag{2.6}$$

Jelikož neznáme proměnné k a x je dobré pro další použití upravit rovnici do následující podoby:

$$x = A + Bk, \text{ kde} \qquad B = -\frac{L}{a}, \qquad A = \frac{a^2-L^2}{2a} = \frac{a+BL}{2} \tag{2.7}$$

Nyní využijeme stejným způsobe proměnnou R, kde při řešení za x dosadíme vztah 2.7, tak aby výsledkem byla rovnice pro hodnotu y:

$$\begin{aligned}
R &= \sqrt{(x-b)^2+(y-c)^2} - \sqrt{x^2+y^2} && \Rightarrow \\
R+k &= \sqrt{(x-b)^2+(y-c)^2} && \Rightarrow \\
(R+k)^2 &= (x-b)^2+(y-c)^2 && \Rightarrow \\
R^2+2Rk+k^2 &= x^2-2xb+b^2+y^2-2yc+c^2 && \Rightarrow \\
R^2+2Rk+k^2 &= k^2-2xb+b^2-2yc+c^2 && \Rightarrow \\
R^2+2Rk &= -2xb+b^2-2yc+c^2 && \Rightarrow \\
R^2+2Rk &= -2b*(A+Bk)+b^2-2yc+c^2 && \Rightarrow \\
R^2+2Rk &= -2Ab-2Bkb+b^2-2yc+c^2 && \Rightarrow \\
y &= \frac{c^2+b^2-2Ab-2Bkb-R^2-2Rk}{2c} && (2.8)
\end{aligned}$$

Upravíme vzorec 2.8 do stejné podoby jako je rovnice 2.7:

$$y = C + Dk, \text{ kde} \qquad C = -\frac{c^2+b^2-2Ab-R^2}{2c}, \qquad D = \frac{-Bb-R}{c} \tag{2.9}$$

Pro určení polohy potřebujeme znát hodnotu x a y, které popisují souřadnice. Výpočet těchto hodnot určují rovnice 2.6 a 2.8. Každá z těchto rovnic ale obsahuje proměnnou k, kterou vypočítáme následujícím způsobem:

$$\begin{aligned}
k^2 &= x^2+y^2 && \Rightarrow \\
k^2 &= (A+Bk)^2+(C+Dk)^2 && \Rightarrow \\
k^2 &= A^2+2ABk+B^2k^2+C^2+2CDk+D^2k^2 && \Rightarrow \\
0 &= A^2+2ABk+B^2k^2+C^2+2CDk+D^2k^2-k^2 && \Rightarrow \\
0 &= k^2(B^2+D^2-1)+k(2AB+2CD)+(A^2+C^2) && (2.10)
\end{aligned}$$

Z rovnice 2.10 je patrné, že se jedná o kvadratickou rovnici. Výsledkem budou tedy dvě hodnoty k. V případě, že diskriminant bude záporný, naměřené časy neodpovídají skutečnosti, a výsledné hyperboly se nikde neprotnou. Výsledkem tedy budou dvě hyperboly, kde jejich průsečík značí polohu. Nicméně při použití tří signálů získáme dva body, kde se hyperboly protínají. Tento problém můžeme řešit dvěma způsoby.

Jedním z těchto způsobů je použití signálu ze čtvrtého vysílače, a vytvořit tak více výsledných hyperbol. Díky této metodě následně můžeme zvýšit přesnost určení polohy. Tomuto způsobu pro získání přesnějšího výsledku polohy se říká multilaterace, a můžeme se o něm dočíst v [17]. Druhou z možností určení, který z těchto dvou bodů je správný, je provést výpočet vzdálenosti mezi bodem a jednotlivými vysílači. Výsledné vzdálenosti převedeme na čas, a všechny tyto časové hodnoty musí souhlasit s naměřenými časy. V případě, že budeme takto porovnávat bod, který není ten správný, jeden z vypočítaných časů bude mít opačné znaménko.

Posun lokalizačního systému

Pro použití tohoto postupu k výpočtu polohy objektu musí být vysílače rozmístěné tak jak je tomu v příkladě zobrazeném na obrázku 2.9. Podmínkami tedy je aby souřadnice

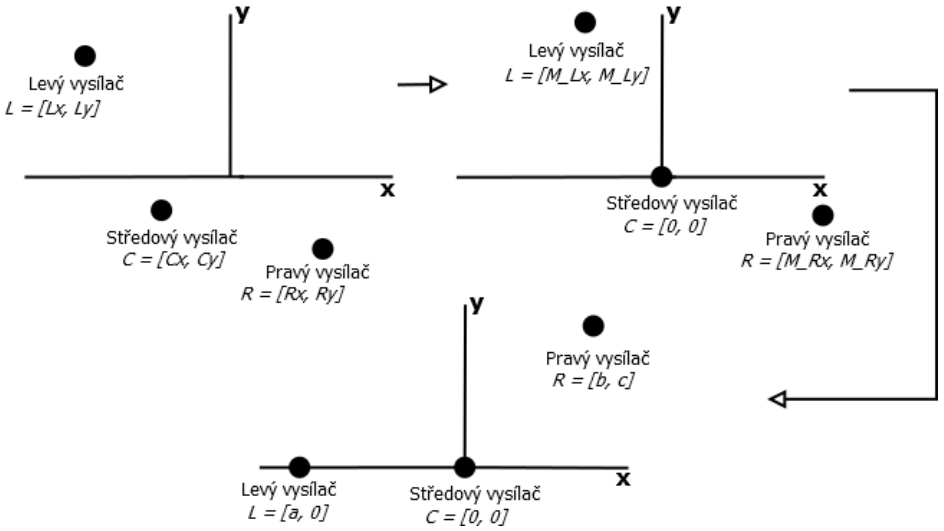
středového vysílače byly $[0, 0]$ a levý vysílač byl ve stejné rovině jako středový vysílač, tedy jeho hodnota y byla 0. V případě systému o více vysílačích není možné tohoto rozložení vždy dosáhnout. Proto musíme systém při výpočtu posunout a vypočítanou polohu následně posunout stejným způsobem zpět. Princip posunu je zobrazen na obrázku 2.10, kde pro první posun použijeme následující rovnice.

$$\begin{aligned} M\_Lx &= Lx - Cx \qquad M\_Ly = Ly - Cy \\ M\_Rx &= Rx - Cx \qquad M\_Ry = Ry - Cy \end{aligned} \tag{2.11}$$

Pomoci rovnic 2.11 dojde posunu systému, tak že středový vysílač má polohu $[0, 0]$. Nyní musí dojít k pootočení vysílačů, tak aby hodnota y pro levý vysílač dosahovala hodnoty 0. K tomu slouží následující rovnice.

$$\begin{aligned} DLL &= \sqrt{(M\_Lx)^2 + (M\_Ly)^2} \\ ca &= -\frac{(M\_Lx)}{DLL} \\ sa &= -\frac{(M\_Ly)}{DLL} \\ a &= -DLL \\ b &= (M\_Rx) * ca + (M\_Ry) * sa \\ c &= -(M\_Rx) * sa + (M\_Ry) * ca \end{aligned} \tag{2.12}$$

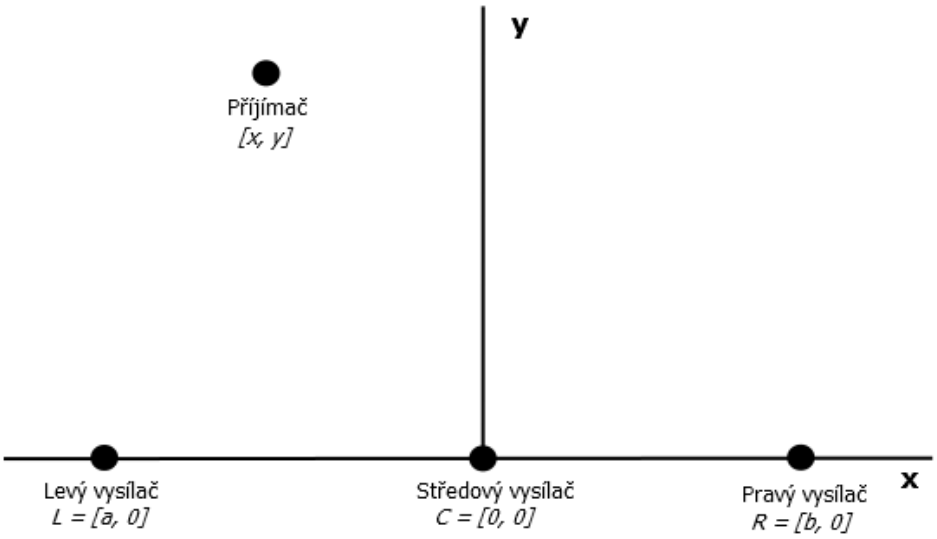
Rovnice 2.12 popisuje otočení systému, kde sa obsahuje hodnotu sinus pro otočení a ca obsahuje hodnoty cosinus pro otočení. Po otočení máme systém v rozložení, ve kterém můžeme vypočítat polohu přijímače. Postup otáčení můžeme vidět na obrázku 2.10.



Obrázek 2.10: Posuv vysílačů pro výpočet polohy přijímače

Jeho poloha bude $[xx, yy]$. Polohu můžeme vypočítat podle rovnic 2.6, 2.8 a 2.10. Výsledkem budou tedy dva body, které budou mít souřadnice v posunutém a pootočeném systému. Získané hodnoty xx a yy musíme posunout zpět, aby poloha přijímače odpovídala skutečné pozici v lokalizačním systému. Výsledná poloha přijímače bude mít tvar $[x, y]$:

$$\begin{aligned} x &= (xx * ca - yy * sa) + Cx \\ y &= (xx * sa + yy * ca) + Cy \end{aligned} \tag{2.13}$$



Obrázek 2.11: Zobrazení umístění vysílačů v jedné přímce

Tímto postupem dokážeme vypočítat polohu vysílače pokud upravíme rozložení vysílačů podle obrázku 2.9. Pokud vysílače jsou v jedné přímce musí být pozměněn výpočet, nicméně je stále potřeba provádět posuv a rotaci systému, tak jak je popsána v této části.

Vysílače tvoří přímku

Vysílače mohou svým umístěním tvořit přímku ve 2D prostoru. Takový příklad je zobrazen na obrázku 2.11. Potom pro výpočet můžeme použít hodnoty A a B ze rovnice 2.7. Dále také můžeme vytvořenou kvadratickou rovnici 2.10, nicméně musí být změněna rovnice pro výpočet hodnoty y z rovnice 2.9. Jinak by docházelo k dělení nulou. Když tedy jsou vysílače v jedné rovině, lze vyjádřit vzorec pro proměnnou y takto:

$$
\begin{aligned}
R &= \sqrt{(x-b)^2+y^2} - \sqrt{x^2+y^2} && \Rightarrow \\
R+k &= \sqrt{(x-b)^2+y^2} && \Rightarrow \\
(R+k)^2 &= (x-b)^2+y^2 && \Rightarrow \\
R^2+2Rk+k^2 &= x^2-2xb+b^2+y^2 && \Rightarrow \\
R^2+2Rk+k^2 &= (A+Bk)^2-2b(A+Bk)+b^2+y^2 && \Rightarrow \\
R^2+2Rk+k^2 &= A^2+2ABk+B^2k^2-2Ab-2Bbk+b^2+y^2 && \Rightarrow \\
y^2 &= R^2+2Rk+k^2-A^2-2ABk-B^2k^2-b^2+2Ab+2Bbk && \Rightarrow \\
y^2 &= (-B^2+1)k^2+(2R-2AB+2Bb)k+(R^2-A^2-b^2+2Ab) && \Rightarrow \\
y &= \sqrt{(B^2+1)k^2+(2R-2AB+2Bb)k+(R^2-A^2-b^2+2Ab)} && (2.14)
\end{aligned}
$$

Následně opět dosadíme do rovnice pro Pythagorovu větu tak, abychom zjistili vzdálenost k, která určuje vzdálenost přijímače od středového vysílače.

$$
\begin{aligned}
k^2 &= x^2 + y^2 && \Rightarrow \\
k^2 &= (A + Bk)^2 + (-B^2 + 1)k^2 + (2R - 2AB + 2Bb)k + (R^2 - A^2 - b^2 + 2Ab) && \Rightarrow \\
k^2 &= A^2 + 2ABk + B^2k^2 + (-B^2 + 1)k^2 + (2R - 2AB + 2Bb)k + (R^2 - A^2 - b^2 + 2Ab) && \Rightarrow \\
k^2 &= (1)k^2 + (2R + 2Bb)k + (R^2 - b^2 + 2Ab) && \Rightarrow \\
0 &= (2R + 2Bb)k + (R^2 - b^2 + 2Ab) && \Rightarrow \\
k &= \frac{R^2 - b^2 + 2Ab}{-2R - 2Bb} && (2.15)
\end{aligned}
$$

Z rovnice 2.15 získáme vzdálenost k, kterou následně dosadíme do rovnice 2.7, kde získáme hodnotu x. Následně hodnotu y získáme podle vztahu odvozeného z Pythagorovy věty.

$$y = \sqrt{k^2 - x^2} \tag{2.16}$$

Umístění vysílačů v přímce se pro lokalizaci nepoužívá, protože dochází ke snížení dosahu a přesnosti systému.

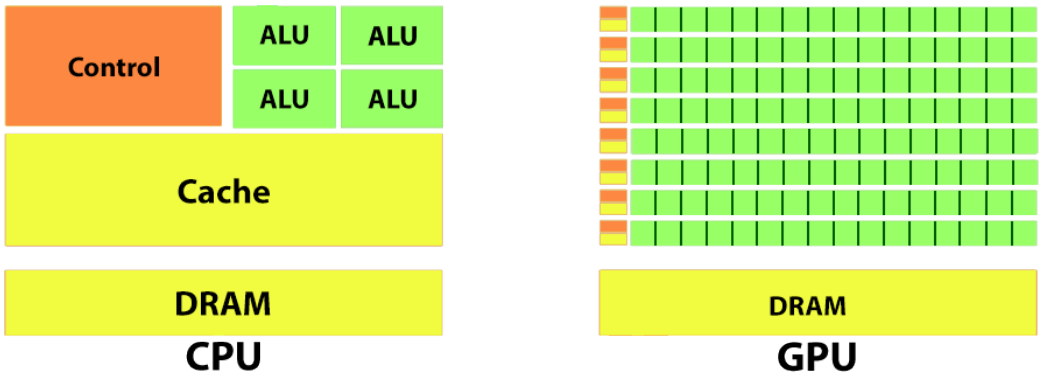
2.4 Možnosti paralelizace

Softwarové rádio je program, který může simulovat chování hardwarových prvků určitého rádiového přijímače. Je to program, který běží na výpočetních zařízeních jako počítač, notebook, atd., které mají připojený rádiový přijímač. Program tedy následně běží na zařízení pod nainstalovaným operačním systémem. Zpracování přijímaného signálu může být výpočetně náročné. Když tedy má program pracovat v reálném čase a provádí mnoho operací se signálem, musíme program vhodně paralelizovat. Tak dosáhneme vyššího využití výkonu zařízení. Výpočty v procesorech provádí ALU (Arithmetic Logic Unit). V této jednotce se provádějí všechny aritmetické i logické výpočty.

Hlavní výpočetní jednotkou v počítači je procesor CPU (Central Processing Unit). Tyto výpočetní jednotky jsou navržené s ohledem na minimální odezvu s nízkým objemem práce za čas. CPU také disponuje velkou mezipamětí (*cache*) a instrukční jednotkou, která umožňuje dobře optimalizovat vykonávání instrukcí. Hlavní paměť počítače tvoří DRAM, která dosahuje kapacity v řádu gigabytů. Tato paměť má vyšší dobu přístupu a zapisování, než mezipaměť procesoru. Paměť DRAM slouží pro ukládání větších objemů dat se stále rychlým přístupem oproti pevným diskům.

Dalším procesorem, který se v počítačích nachází je grafický procesor neboli GPU (Graphic Processing Unit). Tyto grafické jednotky mohou být samostatný komponent, jako je grafická karta. Grafický procesor lze také nalézt na některých základní deskách, nebo jsou součástí chipu CPU. Grafické procesory neobsahují velikou mezipaměť, ale mají více jednotek ALU, jak je vidět na obrázku 2.12. Každá grafická karta má svojí paměť DRAM nezávislou na hlavní pamětí, kterou obsluhuje procesor CPU [28].

Seznam typů pamětí DRAM, které jsou součásti grafických karet jsou zobrazeny v tabulce 2.1. Tyto grafické paměti jsou označované jako GDRAM. Procesor CPU je s grafickou kartou propojen pomoci sběrnice. Přenosová rychlost sběrnice je ovlivněna typem a šířkou



Obrázek 2.12: Zobrazeni struktury CPU a GPU

Typ paměti	Frekvence (MHz)	Šířka pásma (GB/s)
DDR2	2000–2500	128–230
DDR3	2500–3500	160–224
GDDR4	3000—4000	160—256
GDDR5	1000—2000	288—336,5
GDDR5X	1000—1750	160—673
HBM	250—1000	512—1024

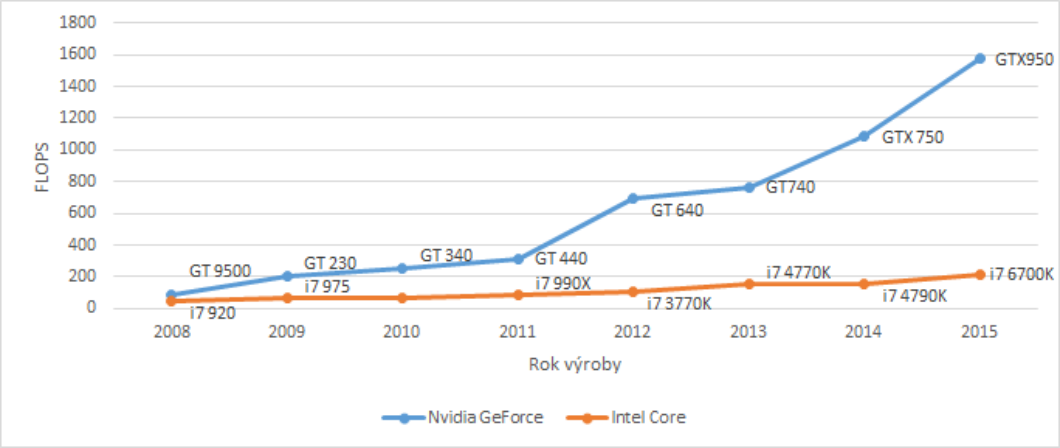
Tabulka 2.1: Typy paměti DRAM, frekvence a šířky pásma [26]

dané sběrnice. V dnešní době je používána pro přenos mezi CPU a GPU sběrnice PCI Expres verze 3.0. Přenosová rychlost této sběrnice při šířce pásma 128 bitů je 15,8 GB/s [25]. Přenosová rychlosti sběrnice mezi CPU a GPU je nižší než šířka pásma pamětí DRAM 2.1.

Výsledný rozdíl mezi GPU a CPU tedy tvoří počet výpočetních jednotek. Nižší kapacita mezipaměti a počet instrukčních jednotek téměř vylučuje jakoukoli optimalizaci vykonávaných instrukcí při překladu programu. Grafický procesor dále také obsahuje několik řídících jednotek, které pracují s určitým počtem ALU (nazývaných *stream* procesory). GPU jsou tedy navrženy na maximální objem práce za čas i s horší odezvou. V případě CPU nedochází k velkému zvyšování výpočetního výkonu. Důvodem je, že při zvyšování frekvence procesorů roste potřeba na chlazení a na energetický příkon. Struktura GPU tyto problémy nemá, proto můžeme vidět v grafu 2.13 vyšší nárůst výpočetní rychlosti procesorů CPU než GPU. GPU ale má menší instrukční sadu oproti CPU, a tak nemá vysokou optimalizaci instrukcí. Rychlost výpočtu se udává v jednotkách FLOPS (Floating-point Operations per Second). Tato jednotka vyjadřuje počet operací v plovoucí desetinné čárce za sekundu. Graf nám nezobrazuje maximální dosažené výkony v daném roce, ale přibližný vývoj výpočetní rychlosti.

2.4.1 CPU

Paralelizovat program při práci s CPU lze pomoci vláken nebo procesů. Nový proces je kopie původního procesu a jeho dat, proto vzniká značná režie dat. Procesy nemohou navzájem přistupovat do jedné paměti. Jedinou výjimku tvoří sdílená paměť (shared memory). Je to paměť, která je poskytována systémem pro komunikaci mezi procesy. Dochází k vytváření



Obrázek 2.13: Porovnání a vývoj výkonu GPU a CPU

tzv. rour, pro komunikaci mezi procesy. Pro každý přístup pro čtení a zapisování do této paměti musí proces žádat systém. Jednotlivé procesy jsou na sobě nezávislé, protože dochází ke kompletní kopii. V případě, že jeden z procesů bude přerušen nebo nastane chyba, tato chyba neovlivní ostatní procesy. Procesy je tedy vhodné používat v místech, kde není zapotřebí komunikace mezi jednotlivými procesy. Jedním z takových příkladů může být webový server, který vytváří pro každou komunikaci jednotlivé procesy.

Pro paralelizaci programů lze použít také vlákna. Vlákna jsou podobná procesům, jenom v případě tvorby nového vlákna nedochází ke kopii původního. Proto vytvoření nového procesu trvá déle než vytvoření nového vlákna. Všechna vlákna jsou tvořená v jednom procesu a proto je významný rozdíl ve sdílené paměti. Vlákna sdílí svojí paměť pomoci globálních proměnných v rámci jednoho programu. Přistup do paměti zůstává v rámci programu. Nevýhodou vlákna je jeho odolnost vůči chybám. Pokud v jednom vláknu nastane chyba, může dojít k pádu celého programu. V tabulce 2.2 můžeme vidět pro jednotlivé procesory dobu pro vytvoření procesu nebo vlákna a rychlost přenosu mezi nimi [13].

CPU	Čas pro vytvoření nového (ms)		Přenosová rychlost (GB/s)	
	Proces	*Vlákno*	*Proces*	*Vlákno*
Intel 2.6 GHz Xeon E5-2670	8,1	0,9	4,5	51,3
Intel 2.8 GHz Xeon 5660	4,4	0,7	5	32
AMD 2.3 GHz Opteron	12,6	1,2	1,8	5,3
AMD 2.4 GHz Opteron	17,6	1,4	1,2	5,3

Tabulka 2.2: Čas pro vytvoření a přenosové rychlosti mezi vlákny a procesy [13]

Práce se sdílenou pamětí u procesů a vláken je dost podobná procesům, co se týče přístupu. Do sdílené paměti může zapisovat pouze jeden proces či vlákno, aby nedošlo k poškození uložených dat. Přesněji tedy, aby byla jistota, že v paměti jsou zapsána data pouze jedním vláknem. Pro synchronizaci procesů a vláken se používají mutexy, podmínkové proměnné a semafory. Mutex se často používá pro synchronizaci přístupu ke sdílené paměti. Dosahuje pouze dvou stavů odemknuto/zamknuto, aby došlo k vzájemnému vyloučení přístupu k paměti. Používá se spíše pro vlákna. Druhou možností k omezenému přístupu k paměti je použití semaforů. Ty se používají v procesech, a jejich výhodou oproti

mutexu je, že dosahují více stavů. Proto jejich použití není pouze pro přístup k paměti, ale také synchronizaci jednotlivých procesů.

Posledním nástrojem pro synchronizaci vláken jsou podmínkové proměnné. Ty slouží k synchronizaci vláken na základě signálů. Vlákno vyčkává na signál, který pošle předchozí vlákno, po dokončení určité části kódu [3]. Na jeden signál může čekat více navazujících vláken a tak dojde k mnohonásobnému uvolnění z čekání. Tuto možnost mutexy neumožňují. Nevýhodou signálů je, že nepřetrvávají. To znamená, pokud vlákno, které se synchronizuje pomoci tohoto signálu, nečeká, potom vytvořený signál nebude využit a ztrácí se. Proto pokud vlákno zmešká signál, může uváznout [23].

2.4.2 GPGPU

GPGPU (General-Purpose Graphic Processing Unit) jedná se způsob využití paralelizace na grafické kartě k výpočtu obecných algoritmů. Dříve grafické procesory byly schopny zpracovávat pouze specializované výpočty. Specializovanými výpočty jsou myšlené výpočty, pro které jsou GPU určeny, tedy vykreslování grafiky. GPGPU využívá grafické jednotky GPU a vlastní programovací jednotku, kterou lze použít právě pro paralelizaci. První standardy, které podporovali GPGPU, byly představeny koncem roku 2000.

Všechny výpočty jsou tedy prováděny na GPU. Pro možnosti paralelizace na GPU je potřeba popsat jeho strukturu. Přibližnou strukturu grafického procesoru můžeme vidět na obrázku 2.12. GPU se skládá z několika řídicích jednotek, které obsahují více ALU, na kterých dochází k výpočtům. Každá řídicí jednotka má vlastní instrukční sadu. Instrukční sada je menší než u CPU, proto některé z grafických procesorů nejsou schopné provádět operace s čísly v plovoucí desetinné čárce. Grafické procesory oproti CPU dosahují nižší pracovní frekvence, a pro jednu operaci potřebují GPU více času. Proto grafické procesory jsou vhodnější pro manipulaci s daty [18].

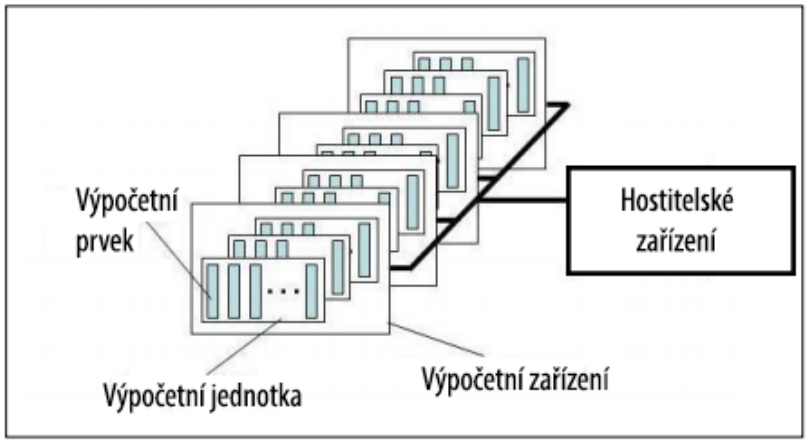
Paralelizovat program pomoci GPGPU nám umožňuje mnoho standardů. Jedním z těchto standardů je například CUDA [9], se kterým lze provádět paralelizaci pouze na grafických procesorech od firmy Nvidia. Druhou možností je standard Stream [10]. Tento standard podporuje jenom grafické procesory od firmy AMD. Třetím nejpoužívanějším standardem pro paralelizaci pomoci GPGPU je standard OpenCL, jehož podrobnější popis najdeme v následující kapitole.

2.4.3 OpenCL

OpenCL je průmyslový standard, který umožňuje provádět paralelizaci nejenom na CPU, ale i GPU (pomoci GPGPU), nebo na dalších samostatných výpočetních zařízeních. První návrh standardu vznikl v průběhu roku 2008 výrobci grafických karet a procesorů. Pro následné zkvalitnění vývoje, byla vytvořena pracovní skupina v rámci tzv. *Khronos Group*. Poslední vydanou verzí standardu je OpenCL 2.2 z roku 2016. Nicméně od verze 2.0 nastává problém s podporou grafických karet nVidia. Jejich podpora skončila u verze 1.2 a zatím nepodporují novější verze tohoto stadnardu. Standard nezahrnuje pouze programovací jazyk, ale i knihovy, API a runtime systémy pro podporu vývoje softwaru. Programovací jazyk pro OpenCL je rozšířením jazyků C a C++. OpenCL umožňuje programátorovi napsat univerzální programy pro grafické karty, které jsou přenositelné. Standard OpenCL se skládá ze 3 hlavních modelů. Jednotlivé modely si popíšeme v následujících kapitolách.

Platformní model

Tento model se skládá ze dvou hlavních části. První částí je hostitelské zařizení (*host device*), který implementuje v počítači CPU. K tomuto zařízení je připojeno jedno nebo více OpenCL zařízení. Každé z těchto zařízení obsahuje výpočetní jednotky (*compute units*), které jsou následně rozdělené do výpočetní prvky (*processing elements*). Všechny výpočty na OpenCL zařízení probíhají právě v výpočetních prvcích. Složení platformního modelu můžeme vidět na obrázku 2.14.



Obrázek 2.14: Platformní model

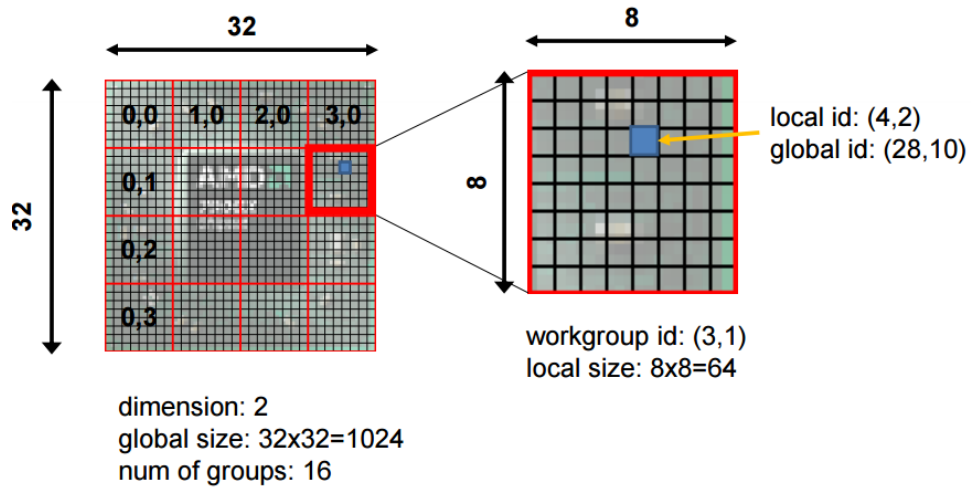
OpenCL aplikace se tedy skládá ze dvou částí. První částí je hostitelský program, který je vykonáván hostitelským procesorem. Druhou částí je potom kód, který je vykonáván a OpenCL zařízení. Tento kód odesílá hostitelské zařízení jako příkazy na zařízení OpenCL. Kód posílaný pro zařízení OpenCL může být buď v binární podobě, nebo v podobě zdrojového kódu. Jednotlivé výpočetní prvky v OpenCL zařízení následně vykonávají kód.

Vykonávací model

Vykonávací model se skládá ze dvou částí: program vykonávaný OpenCL zařízením (*kernel*) a hostitelský program (*host program*). Kernel je spustitelný kód, který se vykonává na OpenCL zařízeních. Kernely mohou být buď datově paralelní, nebo úlohově paralelní. Úlohově paralelní jsou kernely, které na sebe při výpočtech navazují. Datově paralelní je případ, kdy jednotlivé kernely provádějí stejné úlohy, ale s jinými daty.

Každý z kernelů je spuštěn na jednom výpočetním prvku. Tento prvek se nazývá pracovní položka (*work-item*), která má vlastní jednoznačné ID v rámci indexového prostoru. Pracovní položky jsou dále organizované do pracovních skupin (*work-group*). Skupiny mají také své jednoznačné ID jak je tomu u pracovních položek. Každý z kernelů má v rámci jedné pracovní skupiny svůj identifikátor. Slouží pro komunikaci mezi kernely za použití sdílené paměti. Celý tento prostor se nazývá NDRange. N označuje počet dimenzí a nabývá hodnot 1, 2 a 3. D označuje jednotlivé pracovní položky v OpenCL zařízení bez ohledu na skupiny. Příklad rozdělení výpočetního prostoru můžeme vidět na obrázku 2.15, který popisuje jednotlivá ID pro jednu výpočetní položku.

Každá pracovní položka obsahuje svojí paměť, dále má přístup k paměti skupiny a ke globální paměti. To ale popíšeme v další kapitole, nicméně nám umožňuje provádět jak datovou paralelizaci, tak úlohovou paralelizaci. Druhou částí vykonávacího modelu je



Obrázek 2.15: Popis identifikace pracovní položky v rámci zařízení [4]

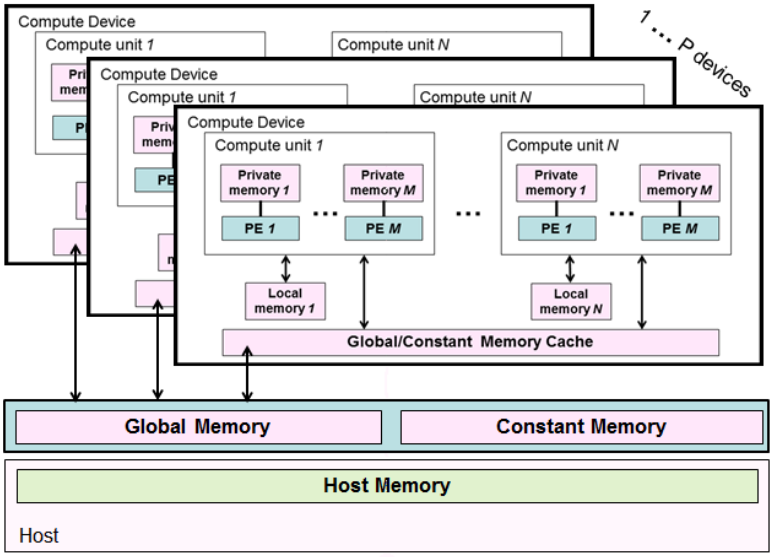
hostitelský program, tedy program, který běží na hostitelském zařízení a spravuje zdroje OpenCL. Tento program definuje kontext, který se skládá z:

- **Devices**: Seznam všech OpenCL zařízení, která jsou použitelná
- **Program objects**: Programový a spustitelný kód, který je implementován do kernelu
- **Kernels**: Funkce, která běží na OpenCL zařízeních.
- **Memory objects**: Paměťové objekty, které slouží pro přenos informací mezi hostitelem a OpenCL zařízení. Každý z kernelů na zařízení má k těmto objektům přístup.

Další část, kterou kontext obsahuje je fronta příkazů(*Command Queue*). Slouží pro správu vykonávání jednotlivých kernelů. Jeden kontext může obsahovat několik na sobě nezávislých front. Tyto fronty obsahují příkazy pro vykonání jednotlivých kernelů s jejich parametry, příkazy pro správu paměti (přesouvání dat z, do nebo mezi jednotlivými paměťovými objekty) a příkazy pro synchronizaci (zachování pořadí prováděných příkazů). Fronta tedy tvoří plán příkazů, které se mají na OpenCL zařízení provést. Tyto příkazy jsou spouštěny asynchronně mezi hostitelem a zařízením. Příkazy se mohou vykonávat mimo pořadí. V obou z těchto možností jsou příkazy vykonávaný v pořadí, ve kterém byly do fronty vloženy. Nicméně při vykonávaní mimo pořadí se příkazy spouštějí bez čekání na dokončení předchozího příkazu.

Paměťový model

Tento model popisuje různé typy pamětí a možnosti oprávnění přístupu z hlediska hostitelského programu nebo programu OpenCL zařízení. Pro přenos mezi zařízeními se používá globální paměť. Do této paměti mohou zapisovat jak jednotlivé výpočetní prvky, tak hostitel. Součástí paměti je konstantní paměť, do které může zapisovat pouze hostitel a výpočetní prvky mohou pouze číst data. Další dvě paměti už nejsou přístupné hostiteli, a to ani pro čtení. Jedná se o lokální paměť a privátní paměť. Privátní paměť je nezávislá pro každou výpočetní prvek a může do ní přistupovat pouze jeden výpočetní prvek. Pro komunikaci ve skupině se používá lokální paměť. Na obrázku 2.16 jsou zobrazené jednotlivé paměti a možnosti přístupů [19].



Obrázek 2.16: Zobrazení druhů pamětí a možnosti přístupu k nim [14]

2.5 Možnosti ukládání informací o lokalizačním systému

Při použití aplikace pro lokalizaci na více na sobě nezávislých navigačních systémech je potřeba ukládat všechny důležité informace, které nejsou přenášené daným systémem. Každý systém má různé polohy vysílačů, různé kódy vysílané vysílači a další údaje, které musíme měnit pro jednotlivé systémy. Úprava programu pro každý ze systémů by nebyla vhodná a neumožňovala by případnou změnu v lokalizačním systému bez zásahu do zdrojového kódu programu. Proto potřebujeme nějaký datový soubor, ve kterém tyto informace budou uloženy a kde lze měnit údaje o systému bez zasahování do kódu programu. Pro jednoduchost přenosu jednotlivých systémů budou systémy rozdělené do jednotlivých souborů.

Pro zvolení vhodného formátu musíme definovat údaje, které potřebujeme k jednotlivým systémům ukládat. Musíme znát frekvencí, na které vysílače v daném systému vysílají. Jelikož se jedná o multiplex typu CDMA, bude zadána pouze jedna nosná frekvence a jedna modulační frekvence. Další údaje, které jsou stejné pro všechny vysílače, jsou délka vysílání jednoho bitu a počáteční stav posuvných registrů pro generování kódu. Počáteční stav posuvných registrů bude u všech systémů stejný. Jejich hodnotu tedy nemusíme pro každý systém měnit a proto zůstane staticky nastavena ve zdrojovém kódu programu. Posledními údaji jsou informace o jednotlivých vysílačích. Přesněji poloha vysílače a umístění bitů z posuvného registru, které generují na sobě nezávislé kódy.

Na základě těchto informací můžeme vytvořit vlastní formát pro jejich ukládání. Musíme zajistit kontrolu validity souboru, aby nedocházelo k chybám způsobeným špatným formátem vstupních informací. Proto použijeme již existující formáty, které mají definovanou syntaxi a existují volně dostupné knihovny, které zajišťují kontrolu validity a umožňují přístup k informacím uloženým v daném souboru. Pro ukládání potřebných informací lze zvolit formát typu JSON, XML,... V tomto případě pro ukládání informací použijeme formát XML.

XML (eXtensible Markup Language) je standart pro řazení a přenos dat [12]. Tento standard je podobný jazyku HTML, který se používá pro tvorbu internetových stránek. Data v souboru lze upravovat v jakémkoliv textovém editoru.

2.6 Dílčí závěr

Lokalizaci objektů pomoci rádiových vln je používána hlavně díky způsobu šíření a přesnosti, které dosahuje. Lokalizační systémy používají dálkoměrné metody, které využívají kódového rozdělení jednotlivých signálů. Kód neboli sekvence bitů je vysílána rádiovými vlnami za použití digitální modulace typu BPSK, která mění fázi signálu podle přenášených dat. Každý z vysílačů vysílý jinou pseudonáhodnou sekvenci a je proto možné použit metodu multiplexu CDMA, kde nemusíme měnit frekvenci přijímače nebo mít více přijímačů. Jednou z možností jak zpracovat přijatý signál je pomoci počítače. Softwarově definované rádio je program, který zpracovává určitý signál. Zpracování signálu může být výpočetně náročné.

Uvedli jsme si na jakých komponentech v počítači a jakým způsobem můžeme paralelizovat. Jednotlivé komponenty mají různé výhody a dosahují jiných výpočetních rychlostí. Pokud potřebujeme provést mnoho různých operací, vhodnějším prvkem pro paralelizaci je CPU. Standard OpenCL umožňuje paralelizaci na samostatných výpočetních zařízeních a zároveň není závislé na jednom hardwaru nebo operačním systému. To nám umožňuje paralelizovat například na GPU, které dosahuje vyššího výpočetního výkonu při správně zvolené paralelizaci. Dále jsme uvedli, že potřebujeme ukládat data o lokalizačních systémech. Určili jsme, které informace potřebujeme ukládat a na základě těchto specifikací jsme zvolili datový typ XML.

Kapitola 3

Zpracování lokalizačního proudu dat

Lokalizační proud dat je signál, elektromagnetické záření, vysílaným vysílači. Jednotlivé vysílače mají svojí polohu a vysílají předem stanovený signál, tak aby bylo možno použít metodu TDOA pro určení polohy.

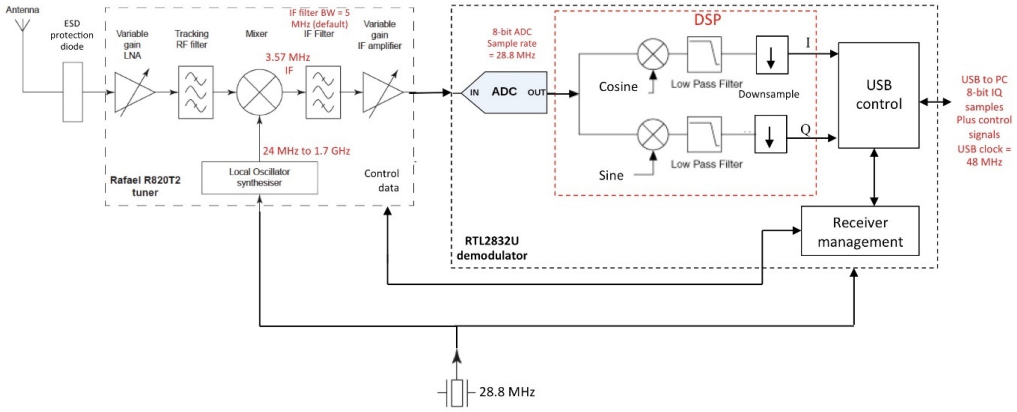
3.1 Možnosti zpracování lokalizačního proudu dat z USB zařízení

Pro přesné zpracování lokalizačního proudu dat musíme znát, jaká data dostáváme od přijímače. Typ dat se může lišit podle jednotlivých přijímačů. Od přijímače můžeme získávat I/Q signál nebo již zpracovaný signál pomoci komponent v přijímači. Přijímač, který budeme používat pro příjem signálu je připojený přes USB. Tento přijímač obsahuje tuner Rafael Micro R820T [5] a digitální demodulátor RTL2832U od firmy Realtek [11]. Tuner toho přijímače přijímá signál na frekvenci od 42 do 1002 MHz. Signál získaný přijímačem je přenášen v analogové podobě do digitálního demodulátoru. Demodulátor následně přes USB rozhraní do počítače posílá hodnoty i a Q, které jsou typu `uint8_t` a nabývají tak hodnot 0 - 255. Blokové schéma přijímače je zobrazené na obrázku 3.1. Veškeré další zpracování signálu následně provádí softwarové rádio.

3.1.1 Ovládání přijímače

Komunikace mezi USB přijímačem a softwarovým rádiem je celkem jednoduchá. Všechny funkce potřebné pro práci s přijímačem pro jazyk C a C++ obsahuje hlavičkový soubor rtl-sdr.h, kde lze najít názvy funkcí, parametry a stručný popis funkce. Při použití tohoto hlavičkového souboru vyžaduje překlad programu použití statické knihovny, která obsahuje implementaci funkcí z hlavičkového souboru. Při následném spouštění programu musí být programu k dispozici dynamická knihovna. Dynamické knihovny je třeba přenášet s přeloženým programem. Knihovny se umisťují do systémové složky, nebo do stejné složky jako je spustitelný soubor programu. Následně funkce přidané knihovny umožňují ovládat USB přijímač a číst přijatý signál.

Knihovna tedy obsahuje funkce pro nastavení nosné frekvence, vzorkovacího kmitočtu. Tyto hodnoty se udávají v parametrech funkcí v jednotkách Hz a lze je i zpětně získat ze zařízení. Další funkce nám umožňují nastavit zesilovač tuneru a krystalický oscilátor. Zesílení



Obrázek 3.1: Blokové schéma použitého rádiového přijímače [1]

signálu pomoci tuneru se udává v decibelech (dB). V případě že tuto hodnotu nenastavíme, pak určování hladiny zesílení zajišťuje digitální demodulátor, stejně jako nastavení krystalického oscilátoru.

Po připojení přijímače a nastavení všech parametrů, můžeme přejít ke čtení signálu ze zařízení. Toto čtení lze provádět synchronně či asynchronně. Pro každý typ tohoto čtení je vytvořena implementace v knihovně rtl-sdr.h. Parametry funkce dále určují velikost pole a ukazatel na pole kam budou data uložena. Pole se musí skládat z bezznaménkových číslic, která musí být minimálně 8-bitová. V případě asynchronního čtení funkce blokuje další akci, dokud není ukončena z jiného vlákna. Při použití asynchronního čtení musíme musíme provádět paralelizaci, jinak by došlo k úplnému zablokování programu.

3.1.2 Lokalizační proud dat

Jak je možné vidět na blokovém schématu (obrázek 3.1), zařízení je připojené a ovládá se přes rozhraní USB. Pomoci tohoto rozhraní jsou zasílány vzorky signálu podle nastavené vzorkovací frekvence. Vzorkovací frekvence nastavuje frekvenci A/D převodníku (demodulátoru), tedy přesněji počet vzorků přijatého analogového signálu za sekundu. Vzorky jsou následně přenášeny pomoci rozhraní USB do počítače, kde dochází k dalšímu zpracování.

Program, který pracuje jako softwarové rádio, po nastavení přijímače čte vzorky přijatého signálu. Vzorky jsou čteny a ukládány do jednorozměrného pole (vektoru), jehož jednotlivé buňky jsou typu bezznaménkového 8-bit celého čísla (*uint8_t*). Do vektoru jsou uloženy vzorky i a Q přijatého signálu. Na každé sudé pozici vektoru je uložen vzorek i a na následující pozici je vzorek Q, který byl vytvořen demodulátorem ve stejnou chvíli jako vzorek I.

3.2 Návrh zpracování lokalizačního proudu dat

Jak jsme zmínili v předchozí kapitole, budeme pracovat s přijímačem, který zasílá vzorky i a Q přes rozhraní USB. Vzorky jsou vytvořené pomoci demodulátoru, který převádí přijatý analogový signál do digitální podoby. Další informací, kterou potřebujeme znát, je typ signálu, který je přenášen na dané frekvenci. V našem projektu budou lokalizační data přenášena pomoci změny fáze rádiového signálu. Data jsou v obou systémech přenášena

stejným způsobem, který je popsán v podkapitole 2.2.1. Obě tyto metody potřebují vytvořit referenční sekvenci, se kterou budeme porovnávat přijaté informace.

3.2.1 Generování referenční sekvence a korelace

Sekvenci, kterou vysílá vysílač, můžeme mít uloženou, nebo jí můžeme generovat stejným způsoben a na základě stejných parametrů jako vysílač. V podkapitole 2.2.3 jsme popsali co je to sekvence a k čemu při určení polohy slouží. V projektu budeme používat Goldův kód, konkrétně tedy délku posuvných registrů 10-ti bitů. Dojde tedy ke generování sekvence s délkou 1023 znaků. Počáteční stav obou posuvných registrů může být stejný u všech vysílačů v jednom nebo více lokalizačních systémech. Generování rozdílných sekvencí určují pozice buněk, ze kterých získáváme bity pro generování (obrázek 2.6).

Pro generování všech potřebných sekvencí nám stačí pouze dva posuvné registry. Známe tedy sekvenci znaků, které jsou posílány. Potřebujeme ještě ale vědět délku jednoho bitu, který přenášíme. Přesněji kolik vzorků je jeden bit. To se vypočítá pomoci vztahu 3.1, kde je potřeba znát jakou dobu je vysílán jeden bit. Pro generování sekvence budeme potřebovat, dva posuvné registry a jednoduché sčítání. Generování může proběhnout pouze jednou a výslednou sekvenci si uložíme do vektoru nebo můžeme sekvenci generovat při každé korelaci.

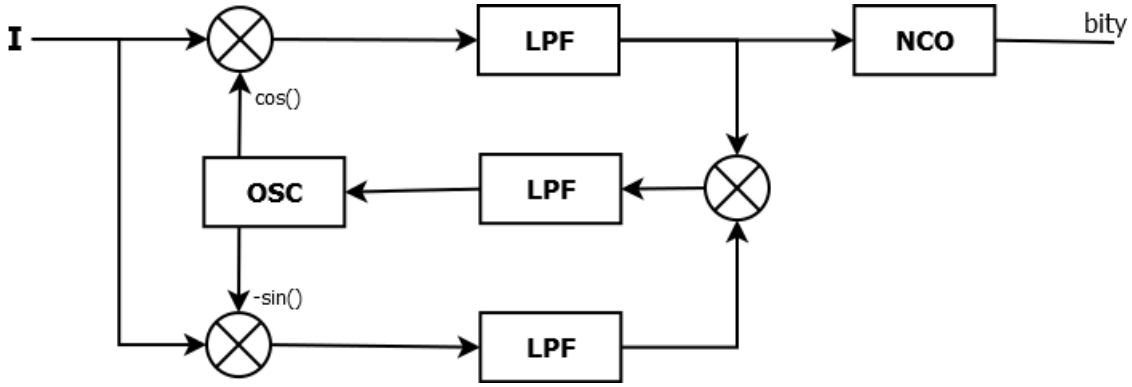
$$pocet\ vzorku\ v\ jednom\ bitu = \frac{doba\ vysilani\ jednoho\ bitu}{vzorkovaci\ frekvence} \tag{3.1}$$

Korelace je proces, při kterém dochází k porovnání generované sekvence s přijatým signálem. Princip korelace je popsán v kapitole 2.2.3. Na obrázku 2.7 je dále zobrazené, jak korelace probíhá a její výsledek. Implementace korelace je tedy velice jednoduchá a lze jí implementovat několika způsoby. Jedním ze způsobů korelace je pomoci násobení, kde demodulovaný signál a generovaná sekvence se skládá z čísel 1 nebo -1. Následně dojde k sečtení všech výsledků násobení a máme bod pro korelační křivku. Druhým ze způsobů, jak provádět korelaci je porovnávání. Tento způsob implementace by snížil potřebnou velikost jednotlivých buněk v poli na jeden bit.

3.2.2 Zpracování signálu pro statický systém

Signál je přenášen pomoci digitální modulace a poslán pomoci rádiového signálu. Demodulace je proces získání poslaných informací přes rádiový signál. Jak již bylo zmíněno pro přenos lokalizačních dat, používáme digitální modulaci typu BPSK. V BPSK pomoci otočení fáze signálu dojde ke změně přenášené hodnoty. Jedná se o binární modulaci, kdy jsou přenášeny pouze dva stavy. V případě lokalizačních dat jsou tyto stavy -1 a 1. Pro digitální demodulaci typu BPSK se používá Costasova smyčka (*costas loop*). Blokové schéma této smyčky můžeme vidět na obrázku 3.2. Costasova smyčka zpracovává přijaté i a Q hodnoty. Následná hodnota přijatého znaku je určena podle fáze na výstupu tohoto prvku. Pokud fáze na výstupu je záporná, jedná se o znak -1, pokud kladná tak 1. Přijímač posílá vzorky I/Q ve formátu bezznaménkového 8-bit celého čísla. Vzorky přijatého signálu dosahují hodnot 0 až 255, v tomto případě by nebylo možné vytvořit zápornou fázi. Musí tak dojít k posunutí signálu, aby bylo dosaženo i záporné fáze. Od každého vzorku odečteme hodnotu 127,5. Tak dojde k posunu přijatého signálu. Takto posunuté vzorky před Costasovou smyčkou jsou uloženy jako hodnoty v plovoucí desetinné čárce (double, float).

Další prvky, které Costasova smyčka obsahuje, jsou filtry dolní propusti označené jako LPF. Tyto filtry slouží ke snížení velikosti kmitů signálu. Filtry lze implementovat jako



Obrázek 3.2: Blokové schéma Costasovy smyčky

posuvné registry, ze kterých je počítán průměr při vložení nového prvku. Na výstupu je umístěný prvek NCO, tedy fázový závěs. Tento fázový závěs nám určuje, jestli výstup z filtru je kladný a jedná se o hodnotu 1 nebo záporný pro hodnotu 0.

Poslední prvek má zkratku OSC. Jedná se o referenční oscilátor [16], který můžeme také nahradit prvkem NCO. Oscilátor se skládá hlavně z fázového závěsu. Důležité pro tento prvek je správné nastavení modulační frekvence. Pokud frekvence nebude správně nastavena, nemusí dojít ke správné demodulaci. Jelikož budeme určovat polohu pohybujícího se objektu vůči vysílačům, je nutné tuto frekvenci upravovat. Ke změně frekvence dochází vlivem Dopplerova jevu. Odchylku modulační frekvence lze určit několika způsoby. Jedním ze způsobů je použití rovnice 3.2. Výsledkem této rovnice bude amplituda s dvou násobnou frekvencí než je frekvence modulačního kmitočtu.

$$sample = -atan(\frac{I}{Q}) \tag{3.2}$$

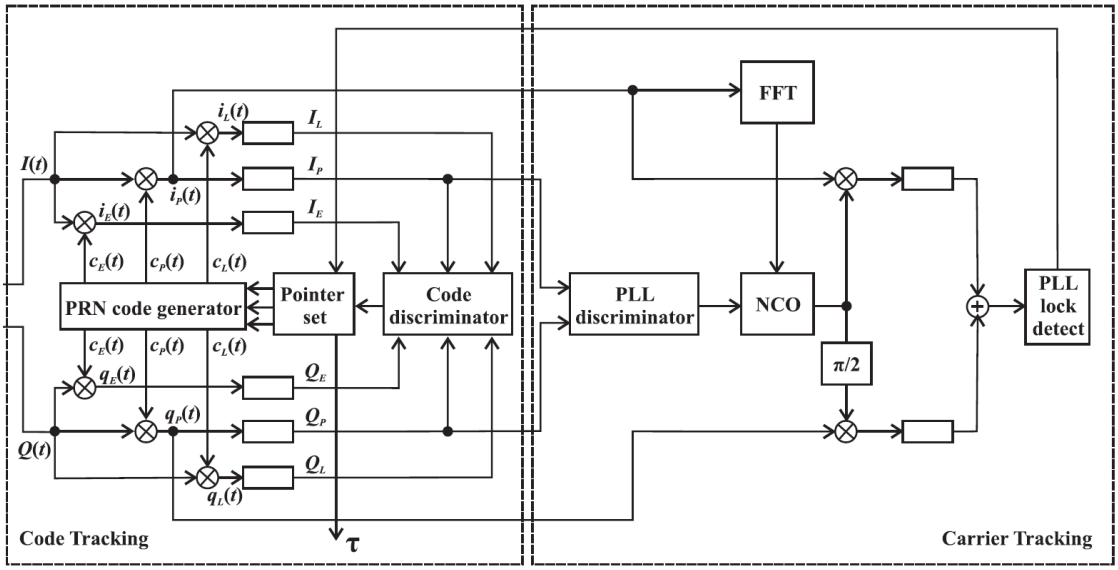
Druhou z možností je ta, že budeme násobit vzorek i a Q. Výsledek tohoto násobení budeme zpracovávat pomoci FFT (*Fast Fourier Transformation*), díky které dostaneme přesnou modulační frekvenci. Výsledkem této demodulace jsou jednotlivé znaky, které byly vysílány. Tato data musíme porovnat s referenčními sekvencemi. Tomuto procesu se říká korelace a je popsán společně s generováním sekvencí v kapitole 2.2.3.

3.2.3 Zpracování signálu pro průběžný systém

Pro tuto možnost přenášení dat byl zvolen obvod, který neprovádí pouze demodulaci, ale zároveň porovnává data s referenčními sekvencemi. Výsledkem tohoto obvodu je rozdílný čas mezi příjmem signálu z jednotlivých vysílačů. Na obrázku 3.3 můžeme vidět blokové schéma tohoto zpracování.

Jak je vidět toto schéma se skládá ze dvou částí. První z částí je *Code Tracking*. Porovnávání přijatého signálu s referenční vygenerovanou sekvencí. Porovnává pouze jednu sekvenci. Proto pro každou sekvenci musíme vytvořit tuto porovnávací část. V *Code Tracking* je rovněž zřejmé, že dochází k přelévání energie mezi i a Q větvemi.

Tato implementace provádí synchronizaci referenční sekvence s přijatým signálem. Postup se nazývá *early late correlation*. K porovnávání používá tři repliky brzký(*early*), okamžitý(*prompt*) a opožděný(*late*) referenční sekvence, které jsou pro větve i a Q stejné. Tyto repliky (c_E, c_P, c_L) jsou vynásobeny se vstupním i a Q signálem a vznikne nám tak šest



Obrázek 3.3: Blokové schéma pro zpracování signálu průběžného systému

vstupních signálů pro korelátor. Každý z těchto signálů je přiveden do registrů, kde je následně vytvořen pro každých N vzorků. Tyto výsledky jsou označené I_E, I_P, I_L, Q_E, Q_P, Q_L. Výsledky jsou následně použity v DLL diskriminátoru označeného jako `Code discriminator`. DLL (*Delay Lock Loop*) se zaměřuje na sledování zpoždění příchozího signálu [21]. Výpočet, který diskriminátor provádí, popisuje rovnice 3.3.

$$D_{DLL} = \frac{I_E - I_L}{I_P} + \frac{Q_E - Q_L}{Q_P} \tag{3.3}$$

V případě, že hodnota I_P nebo Q_P, je rovna 0, výpočet D_{DLL} není proveden. Výsledná korekční hodnota D_{DLL} je použita pro časový posun generovaných replik. Tento proces se opakuje, do té doby dokud nedojde k synchronizaci vygenerované pseudonáhodné sekvence a přijímaného signálu. Výsledný posuv určuje zpoždění signálu, které použijeme pro výpočet polohy.

Druhou část blokového schématu 3.3 tvoří *Carrier Traking*, jejíž úkolem je zjistit, zda-li je referenční sekvence synchronizována se signálem. Ve chvíli kdy je dosaženo synchronizace však stále dochází k posunu referenční sekvence z důvodu nepřesné frekvence amplitudy přijímaného signálu. K určení frekvence je použito vzorků I, které jsou vloženy do prvku FFT. Druhou z možností je vynásobení vzorku i a Q. Vznikne tak signál, který se podobá sinusoidě a frekvence je stejná jako frekvence odchylky referenčního oscilátoru a přijímaného signálu.

3.2.4 Určení polohy

Pro určení polohy je důležité určit vrcholy korelačních křivek. Po zjištění vrcholů určíme dobu mezi vrcholy jednotlivých korelačních křivek. Dobu mezi vrcholy určíme pomoci počtu vzorků, které vrcholy mezi jednotlivými křivkami dělí, a vzorkovací frekvencí. To nám popisuje vztah 3.4, kde indexy proměnných určují jednotlivé vysílače. Proměnná vrchol určuje pozici vzorku vrcholu v korelační křivce.

$$t_{12} = \frac{vrchol_1 - vrchol_2}{vzorkovacifrekvence} \tag{3.4}$$

Jelikož všechny vysílače vysílají sekvenci ve stejný čas, pomoci tohoto vzorce můžeme zjistit časový rozdíl mezi přijmutím signálu z jednotlivých vysílačů. Následně pomoci vztahu 3.5, zjistíme rozdílnou vzdálenost přijímače od jednotlivých vysílačů, na základě času. Konstanta c označuje hodnotu rychlosti světla.

$$a_{12} = t_{12} * c \tag{3.5}$$

Použitím vzorce 3.5 získáme hodnoty L a R. Tyto hodnoty dále použijeme společně s informacemi o pozicích vysílačů pro výpočet polohy, tak jak je popsán v kapitole 2.3.4. Musíme provést také posun, abychom mohli použít výše zmíněné vzorce.

3.3 Návrh paralelizace zpracování lokalizačního proudu dat

Jak jsme si uvedli v kapitole 2.1, softwarové rádio je program, který může modelovat jednotlivé hardwarové prvky pro zpracování signálu. Jednotlivé prvky pracují jako jeden systém. Paralelizace je důležitým aspektem pro zpracování lokalizačního signálu a výpočet polohy v reálném čase. Postup pro výpočet polohy rozdělíme na bloky. Tyto bloky budou zpracovávat vstupní data nezávisle a výsledky předávat dalšímu bloku. Jednotlivé bloky tedy implementujeme pomoci funkcí, které budou rozděleny do hlavičkových souborů podle bloků. Tento způsob implementace umožňuje jednoduchost použití stejného bloku pro programy, které budou zpracovávat data jiného lokalizačního systému.

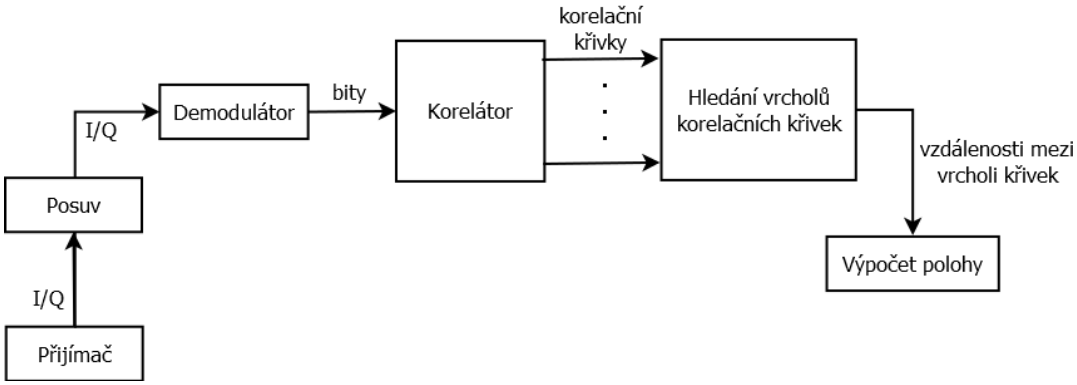
Pro tento projekt je výhodnější použití vláken z důvodu nutnosti přenášet informace mezi jednotlivými bloky. Dalším z důvodů, proč použít vlákna oproti procesům, je redundance dat u procesů. Jednotlivé bloky nepotřebují znát chování ostatních bloků.

3.3.1 Možnosti paralelizace pro statický systém

Pro statickou paralelizaci jsme zvolili následné rozložení do bloků (obrázek 3.4). Důležité pro vhodnou akceleraci programu je zvolení správného počtu bloků tak abychom využili co nejvíce z výpočetní kapacity jednoho vlákna. Všechny z bloků nejdříve implementujeme a otestujeme na procesoru CPU. Na základě výsledků testování potom určíme, které z bloků spojíme a které budeme paralelizovat na GPU. První částí systému je blok přijímače, který pracuje se zařízením USB a potřebuje rychlý přístup do paměti pro přenos dat. Toto vlákno neprovádí žádné matematické operace ale pouze kopíruje přijatá data do globální paměti. Další paralelizace tohoto bloku není možná.

Navazující blok provádí posun signálu jak je definováno v podkapitole 3.2.2. Tento blok provádí jednoduchou matematickou operaci, kdy od každé přijaté hodnoty odečte polovinu rozsahu přijímaných hodnot. Tato část může zároveň rozdělovat i a Q vzorky, které bude ukládat do globální paměti pro další zpracování. Data jsou zpracovávána pomoci třetího bloku – demodulátoru. Provádí demodulaci signálu pomoci Costasovy smyčky popsané v podkapitole 3.2.2. U tohoto bloku je možné provést další paralelizaci. Konkrétně by se jednalo o rozdělení větve I, Q a zpětné smyčky. Tímto rozdělením bychom dosáhli maximálního rozložení na jednotlivá vlákna.

Důležitým blokem je korelátor, který porovnává demodulovaný signál s referenčními sekvencemi a jehož výsledkem jsou korelační křivky. Počet korelačních křivek se odvíjí od počtu vysílačů v lokalizačním systému. Pro každý vysílač vznikne jedna korelační křivka.



Obrázek 3.4: Model rozdělení programu do bloků, pro zpracování signálu BPSK

Jednou z možností paralelizace korelátoru je rozdělení porovnání signálu s jednotlivými referenčními sekvencemi. Každé vlákno by tak vytvářelo jednu korelační křivku. Jednotlivé body korelační křivky jsou na sobě vypočítávány nezávisle a zde existuje další možnost paralelizace. Žádnou z těchto paralelizací však nebudeme muset provést, pokud bude dosaženo zpracování v reálném čase.

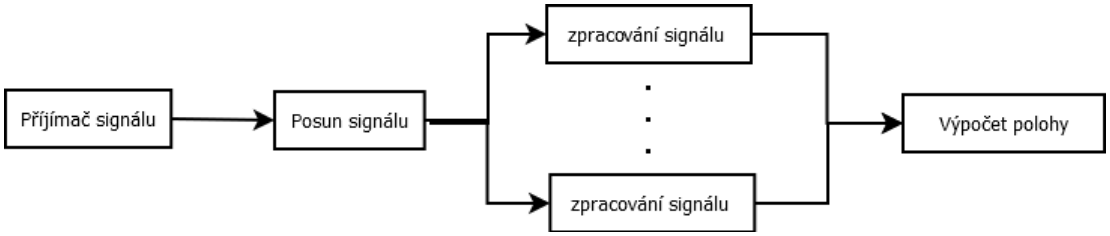
Předposledním blokem je vyhledávání vrcholů v jednotlivých korelačních křivkách. Tento blok určuje počet vzorků mezi vrcholy jednotlivých korelačních křivek. Počet korelačních křivek bude stejný jako počet vysílačů v jednom lokalizačním systému. V případě GNSS jsme si uvedli, že tyto systémy se skládají z přibližně 24 družic. U našeho systému nebude vysílačů více, a proto paralelizaci bude lepší provést pomoci vláken, kde by jedno vlákno kontrolovalo jednu křivku. Musíme ale zajistit, aby všechna vlákna kontrolovala korelační křivky na stejných pozicích nebo stejně rychle.

Posledním z bloků je určení polohy přijímače. Jedná se o matematický výpočet polohy na základě získaných údajů. Rozdílný čas mezi vysílači jsme zjistili pomoci porovnání korelačních křivek a určení vzdálenosti vrcholů mezi jednotlivými křivkami. Následně hodnota byla převedena na čas za použití vzorkovací frekvence.

3.3.2 Možnosti paralelizace pro průběžný systém

Blokové schéma, které bude implementováno programem je znázorněno na obrázku 3.3. Dané blokové schéma zobrazuje demodulaci signálu a zjišťování zpoždění signálu z vysílače. Implementaci systému můžeme rozdělit do méně bloků, než tomu bylo v případě implementace statického systému. Výsledné rozdělení je zobrazeno na obrázku 3.5. Implementaci bloku přijímače a posuvu signálu použijeme stejnou jako u statického systému. Stejně jako u posledního bloku, kterým je výpočet polohy.

Rozdílnou část mezi statickým systémem a průběžný systémem tvoří právě implementace zobrazena na obrázku 3.3. Tato implementace zpracovává a porovnává signál pouze s jednou referenční sekvencí. Pro každou referenci musíme použít stejnou implementace. A to je právě jedna z možností paralelizace, kde každé vlákno bude zpracovávat a porovnávat signál s jednou referenční sekvencí. Vlákna však budou muset být synchronizována, aby porovnala stejnou část dat. Další z možností paralelizace implementace je rozdělení na `Code Traking` a `Carrier Traking` (obr. 3.3), případně další rozložení těchto bloků tak abychom dosáhli zpracování signálu v reálném čase.



Obrázek 3.5: Model rozdělení programu do bloků, pro zpracování signálu průběžného systému

3.3.3 Přenos dat mezi vlákny

Jednotlivé bloky jsou propojené pomoci globální paměti mezi vlákny. Mezi přijímačem, demodulátorem a korelátorem bude sdílená paměť přenášet I/Q vzorky signálu a demodulovaná data. Vzorkovací frekvence přijatého signálu se pohybuje v řádech MHz, to je milionů vzorků za sekundu. Použití pouze proměnné, do které bude kontrolován přístup mutexem pro přenos mezi bloky, by brzdilo chod programu. Proto paměť, která bude spojovat jednotlivé bloky, bude tvořit pole. Sníží se počet operací mutex, které budeme používat pouze pro ověření čtení nebo zápisu do pole a ne pro každou hodnotu. Operace mutex slouží pro synchronizaci jednotlivých vláken a zajistí, aby více vláken nepracovalo s jednou pamětí.

Druhá z možností je plynulejší pro přenos dat. Využívá pole tvořící paměť do které přistupují dvě vlákna. V této implementaci budeme potřebovat jednu podmínkovou proměnnou s mutexem a samostatný mutex. Vlákno, které zapisuje data do paměti, používá pro signalizaci podmínkovou proměnnou. Tak oznámí dalšímu vláknu, že data může číst. Signál je posílán po uložení nové hodnoty do pole.

Paměťových bloků mezi vlákny musí být více než dva proto, aby nedocházelo k zastavení práce vlákna z důvodu čekání na uvolnění přístupu do paměti. Vlákno může čekat pouze na vstupní data, a nesmí být zastavováno nedostatkem prostoru na výstupní paměti. V případě přijímače by tak došlo k chybě, kdy přijímaná data nebudou na sebe navazovat. To by mohlo ovlivnit výsledky určení polohy, případně bychom nebyly schopni polohu určit vůbec.

3.4 Dílčí závěr

V této kapitole jsme si popsali možnosti ovládání a složení rádiového přijímače, který budeme používat pro daný projekt. Důležité jsou komponenty, ze kterých se přijímač skládá, případně zda neobsahuje další prvky, které by ovlivnili formát dat posílaných přes rozhraní USB. Přijímač používá pro komunikaci rozhraní USB a odesílá vzorky I a Q podle nastavené vzorkovací frekvence. Vzorky jsou posílány ve formátu bezznaménkového 8bitového celého čísla (uint8_t).

Následné zpracování přijatého signálu pomoci počítačového programu (softwarového radia) se odvíjí od právě definovaného přijímače a typu modulace. Popsali jsme, že data jsou posílána pomoci digitální modulace typu BPSK a pro demodulaci signálu lze použít Costasovu smyčku. Demodulovaná data používáme pro lokalizaci, konkrétně provádíme korelaci, jejíž výsledek nám popíše rozdílné vzdálenosti mezi jednotlivými vysílači. Následně můžeme ze získaných dat vypočítat polohu přijímače.

Softwarové rádio provádí mnoho operací s přijatým signálem. Proto je nutné program paralelizovat, abychom dosáhli získání polohy v reálném čase. V podkapitole 2.4 jsme si popsali, jaké jsou možnosti pro paralelizaci. V této kapitole jsme následně popsali další

případné rozložení bloků, které můžeme implementovat pro dosažení zpracování v reálném čase. V poslední podkapitole 3.3.2 jsme popsali možnost paralelizace druhého ze systémů.

Kapitola 4

Implementace zpracování data a určení polohy

V této kapitole si popíšeme implementaci obou systémů. Programování a testování probíhalo na nahraných datech, která byla uložena v souboru. Pomoci této metody jsme mohli testovat časovou náročnost jednotlivých částí programu a zvolit případnou implementaci. Bylo vytvořeno několik nahrávek a součástí testování bylo dosáhnout zpracování dat uložených v souboru rychleji, než je délka časového záznamu uloženého v souboru. Tato vlastnost zajistí, že výsledný systém bude pracovat v reálném čase.

Pro implementaci použijeme programovací jazyk C, jehož odvozená syntaxe se používá pro programování OpenCL zařízení. Jazyk C byl také zvolen, jelikož není potřeba objektově orientovaného programování, které by umožňovalo C++. Aplikace má být funkční na operačním systému Windows. Pro programování a testování implementace bylo použito vývojové prostředí Code::Blocks a Visual Studio, které umožňuje ladit programy i na grafických kartách s použitím pluginu CodeXL. Všechny příklady kódu popsané v následujících kapitolách jsou umístěny v příloze.

4.1 Získání dat z RTL-SDR zařízení

Možnosti ovládání a nastavení RTL-SDR zařízení připojeného přes USB je popsáno v podkapitole 3.1. Součástí této podkapitoly je blokové schéma použitého přijímače zobrazeného na obrázku 3.1. Je popsáno, jaký formát dat odesílá USB zařízení. Jedná se tedy o bez znaménková čísla, která je potřeba posunout. V následujících podkapitolách je popsána možnost implementace pro přijímání a posun signálu, ale také možnosti pro přenos dat mezi vlákny.

4.1.1 Přijímání dat z USB zařízení

V této kapitole se zaměříme na získávání dat z USB zařízení, kde máme na výběr ze dvou možností. První z možností je synchronní čtení dat z RTL-SDR. Parametry této funkce jsou ukazatel na RTL-SDR zařízení, ukazatel na pole, do kterého se budou ukládat data a číslo označující velikost pole v předchozím parametru. Na základě zavolání této funkce začne RTL-SDR zařízení nahrávat a odesílat přijatá data přes rozhraní USB. Pro tento způsob přijímání dat není zapotřebí žádné paralelizace, přijímání dat může probíhat ve stejném vlákně jako jeho zpracování.

Druhou z možností přijímání dat je asynchronní. Pro tento způsob přijímání dat používáme funkci `rtlsdr_read_async`, která blokuje další chod programu, dokud není použita funkce `rtlsdr_cancel_async`. Jelikož dochází k blokování programu, musíme pro toto zpracování použít paralelizaci, kdy jedno z vláken bude obsluhovat právě tuto funkci. Parametry této funkce jsou ukazatel na RTL-SDR zařízení a ukazatel na funkci, která bude data kopírovat do vytvořeného pole. Tato funkce je zavolána vždy, když jeden z bufferů je naplněn přijímanými daty. Poslední parametry určují velikost a počet bufferů. Maximální velikost a počet těchto bufferů je popsána v komentářích k deklaracím funkcí v knihovně rtl-sdr.h, které deklarují funkce obsažené ve statických a dynamických knihovnách.

Musíme vytvořit funkci, která bude kopírovat přijatá data do připraveného pole. Tato funkce musí obsahovat 3 proměnné jako parametry. První parametr je ukazatel na pole, které je tvořeno 8-bitovými prvky. Do tohoto pole jsou ukládána data z RTL-SDR zařízení, jedná se tedy o naplněný buffer. Dalším z parametrů je 32-bitové celé číslo, jehož hodnota je určená parametrem funkce `rtlsdr_read_async`. Poslední z parametrů může obsahovat jakákoliv data, která chceme přenášet pomoci parametrů. Ukazatel na tato data je také vložen pomoci volací funkce. Příklad této funkce můžeme vidět 5.1.

Program v příkladu 5.1 se skládá ze dvou vláken. Jedno vlákno provádí kopírování dat získaných z RTL-SDR do globální proměnné tak, aby data mohla být zapsána do souboru. Funkce popisující chování tohoto vlákna je definována od řádku 9. Tato funkce je volána opakovaně vždy, když je jeden z bufferů definovaných v příkazu `rtlsdr_read_async` plný. Druhé z vláken obsluhuje funkci `saveData`, která je definována na řádku 17. Tato funkce slouží pro zapisování dat v binární podobě do souboru. Tento příklad by bylo možné implementovat také pomoci jednoho vlákna. Místo kopírování do globální paměti bychom zapisovali přímo do souboru. Nicméně na této implementaci můžeme testovat přenos mezi vlákny tak, aby byla splněna podmínka pro přenos v reálném čase. Pro správnou funkčnost programu by také mělo být vytvořeno vlákno, které ukončí asynchronní čtení příkazem `rtlsdr_cancel_async`.

Přenos mezi vlákny probíhá pomoci globální paměti, se kterou pracuje více vláken. Proto je zapotřebí zajistit synchronizaci přístupů do této paměti. Tuto úlohu provádějí mutexy deklarované na řádku 3, v příkladu 5.1. Funkce pro asynchronní čtení ukládá data do pole, ke kterému je přístup až po úplném zaplnění. To určuje také možnost přenosu, to znamená, že budou data přenášena po jednotlivých blocích. V následujících podkapitolách si popíšeme implementaci společné části obou systémů.

4.1.2 Posun získaných dat a přenos vzorků mezi vlákny

Všechny prvky, které budeme implementovat, pracují se signálem, jehož střed musí být v hodnotě 0. Maximální dosažitelná hodnota přijatého signálu musí mít stejnou hodnotu jako minimální dosažitelná hodnota. Jelikož data přenášená z USB zařízení jsou bez znaménkové celé číslo, musíme provést posun těchto přijatých hodnot tak, aby bylo splněno vše definované v předchozích větách. Hodnota posunu se určuje podle velikosti proměnné, do které jsou přijatá data ukládána. V tomto případě dochází k ukládání do 8-bitové proměnné, signál dosahuje hodnot 0-255. Aby střed signálu byl v nule, musí byt proveden posun odečtením hodnoty, v tomto případě $127,5$. Hodnota je stanovena tak, aby minimální a maximální hodnota vzorků měla stejnou hodnotu, ale pouze opačné znaménko. Příklad 5.2 popisuje implementaci posuvu a také přenášení jednotlivých vzorků mezi vlákny. Funkce `signalShift` nahrazuje funkci `saveData` z příkladu 5.1.

Příklad 5.2 popisuje funkci `signalShift`, která provádí posun signálu na řádku 18 a 19. Data jsou ukládána do globální proměnné, kterou tvoří pole. Funkce zároveň rozděluje I a Q vzorky. Globální paměť pro přenos těchto dat mezi vlákny tvoří pole struktur `receiverToDemodulator`. Struktury pro přenos musí být více než 2. Tyto struktury se střídají, tak aby nedošlo k pozastavení jednoho vlákna z důvodu čekání na uvolnění paměti.

Data přenášená mezi vlákny jsou umístěna do pole. Při každém uložení nového vzorku I a Q dojde k zaslání signálu (řádek 21) pro druhé vlákno, které následně provede čtení. Dochází tak k signalizaci o uložení jednotlivých vzorků do pole. To umožňuje postupné zpracování dat. Dochází k většímu rozložení zpracování dat, než v případě, kdy data jsou přenášena jako celé pole najednou jako je tomu v příkladě 5.1.

Signál pro synchronizaci mezi vlákny je nekumulativní, tedy nepřetrvává v programu dokud není odebrán. V případě, že je vyslán signál a druhé vlákno není ve stavu čekání na tento signál (řádek 50), dochází ke ztrátě signálu pro synchronizaci mezi vlákny. Tento problém je řešen právě pomocí pole. V případě, že pole je naplněno, vlákno které zapisuje data změní hodnotu proměnné `dataFull` umístěné ve struktuře. Následně nedochází k vyčkávání na signál z předchozího vlákna, dokud není dočtené celé pole. Proměnná `dataFull` může být změněna ve chvíli, kdy druhé vlákno očekává signál. První vlákno tedy musí poslat signál tak, aby nedošlo k uváznutí. Druhé vlákno potvrdí tento signál změnou proměnné `signalFull`. Tímto způsobem je zajištěno, že nedojde k uváznutí.

Poslední synchronizace, která je prováděna, slouží pro vlákno, které provádí zapisování dat do globální paměti. Tuto synchronizaci zajišťuje mutex `mutMemReady`. Vlákno pro četní dat z globální paměti tento mutex uvolní ve chvíli, kdy si z paměti přečetl všechna data. Příklad 5.2 tedy také popisuje možnost postupného přenosu dat a synchronizace mezi vlákny.

4.1.3 Testování implementace pro získávání dat a posun signálu

Při posunu signálu jsme provedli měření, jehož výsledky jsou zobrazeny v tabulce 4.1. Tato tabulka popisuje kolik času bylo potřeba pro zpracování jednoho záznamu.

CPU	Délka záznamu			
	1 sekunda	5 sekund	10 sekund	30 sekund
AMD A10-4600M	0,0256	0,1149	0,2403	0,676
AMD FX-7500	0,0168	0,0698	0,1444	0,4688
AMD Phenom II X4 840	0,0147	0,0765	0,1543	0,4634
Intel Atom x5-Z8350	0,0368	0,1796	0,3578	1,0743

Tabulka 4.1: Časové výsledky zpracování posunu signálu v sekundách

Z výsledků tabulky je zřejmé, že bylo dosaženo zpracování v reálném čase. Dále je z výsledků patrné, že potřebný čas na zpracování je mnohonásobně nižší než je délka záznamu. Výpočetní kapacita vlákna procesoru tak nebude plně využita. Pro zvýšení využitelnosti výpočetní kapacity procesoru spojíme operaci posunu signálu s přijímáním dat. Tím se sníží i počet použitých vláken při implementaci. Provedli jsme testování, jehož výsledky jsou zobrazené v tabulce 4.2. Pro testování byla zvolena vzorkovací frekvence 2MHz, což je nejvyšší vzorkovací frekvence pro použitý demodulátor. Výsledky jsou průměry 12-ti testů ze 3 zařízení.

Typ záznamu	Délka záznamu			
	1 sekunda	5 sekund	10 sekund	30 sekund
Synchronní bez posunu	1 995 776	9 964 032	19 996 672	59 867 136
Asynchronní bez posunu	1 998 848	9 986 048	19 988 480	58 851 328
Synchronní s posunem	1 986 560	9 976 320	19 710 208	59 749 632
Asynchronní s posunem	1 958 400	9 963 912	19 731 456	59 023 360

Tabulka 4.2: Počet vzorků v záznamu při vzrokovací frekvenci 2MHz

Testování probíhalo formou vytvoření záznamu rádiového signálu do souboru. Z výsledných velikostí souborů lze poznat kolik bylo zaznamenáno a uloženo vzorků. Při použité vzorkovací frekvenci 2MHz dochází k vytvoření 2 milionů vzorků za sekundu, které definují hodnoty I a Q. Z výsledků testu je zřejmé, že nedochází ke ztrátě dat i po spojení vlákna pro záznam a posun signálu. Malé rozdíly velikost jsou však viditelné mezi jednotlivými testy. Tato odchylka však vznikla při způsobu měření délky času pro nahrávání.

Záznamy vytvořené pomoci programu však ukládají neposunutá data. Přijímač pro ukládání hodnot I a Q používá proměnnou `uint8_t`, jejíž velikost je právě 8 bitů. Výsledek posunu je však desetinné číslo a ukládá se do proměnné typu `float`, která má velikost 32 bitů. To znamená, že soubor s nahranými posunutými daty bude 4krát větší než je tomu u neposunutých dat. Dalším z důvodů ukládání neposunutých dat je simulace vlákna, které kopíruje a posouvá data získaná z USB zařízení. V případě záznamu toto vlákno čte data ze souboru nikoli z USB zařízení a následně provádí posun získaných vzorků.

4.2 Implementace statického systému

Statický systém a jeho části, které potřebujeme pro získání lokalizačních dat, jsme si uvedli v podkapitole 3.2.2. Jeho následné rozdělení do jednotlivých bloků popisuje podkapitola 3.3.1. V předchozích podkapitole 4.1 je popsána implementace prvních dvou bloků zobrazených na obrázku 3.4.

V následujících podkapitolách budeme popisovat implementaci zbylých bloků pro statický systém. Pro testování budeme používat uložený záznam neposunutého signálu. První dva bloky ze schématu 3.4 budou implementovány jako čtení ze souboru s posunem signálu. Podmínkou pro jednotlivé bloky je správně zpracovat data a zároveň dosáhnout nižšího času zpracování dat než je délka záznamu. Tak bude zajištěno zpracování dat v reálné čase.

4.2.1 Demodulace signálu

Lokalizační data jsou ve statickém systému posílány pomoci digitální modulace BPSK. Data tvoří změna fáze signálu tak jak je popsáno v podkapitole 2.2.1. Na obrázku 3.2 dále můžeme vidět možnost implementace pro demodulaci signálu. Jedná se o obvod Costasovy smyčky, jejichž výsledkem jsou právě hodnoty 1 a -1. Implementace Costasovy smyčky se skládá z několika prvků.

Prvek NCO (fázový závěs) je implementován jednoduchou podmínkou. Jedná se o určení fáze příchozí hodnoty. Pokud je příchozí hodnota záporná, výsledkem prvku NCO je hodnota -1, v opačném případě je výsledek 1. Dalším prvkem ve schématu Costasovy smyčky je filtr dolní propusti (LPF). Filtr lze jednoduše implementovat pomoci pole, které bude implementovat posuvný registr. Nová hodnota bude vložena na první prvek pole a všechny

ostatní prvky budou posunuty. Dojde tak k odstranění posledního prvku v poli. Následně provedeme součet hodnot v poli, který vydělíme počtem prvků. Tím získáme výsledek filtru dolní propusti.

Referenční oscilátor (OSC)

Referenční oscilátor je prvek tvořený fázovým závěsem. Referenční oscilátor vytváří referenční oscilační signál, vzor přijímaného signálu bez změn fáze. Výstupem tohoto prvku je hodnota, která je při inicializaci nastavena na nulovou hodnotu. Následně je k proměnné přičítána nebo odčítaná hodnota stanovená vzorcem 4.1. Přičtení nebo odečtení této hodnoty je stanoveno fázovým závěsem. Fázový závěs provádí rozhodování na základě vstupní hodnoty.

$$atan(sin(\frac{2 * \pi * f_c}{f_s}))/10 \tag{4.1}$$

Vzorec obsahuje dvě proměnné. Jednou z těchto proměnných je f_s, kterou je vzorkovací frekvence. Tato frekvence je nastavena na začátku přijímaní dat a nastavujeme ji na USB přijímači. Druhou z těchto proměnných je f_c. Tato proměnná obsahuje hodnotu modulačního kmitočtu, která se může v průběhu přijímaní signálu měnit vlivem prostředí nebo Dopplerovým jevem. Proto tuto proměnnou musíme zjistit.

Pro zjištění modulační frekvence budeme potřebovat I a Q vzorky přijatého signálu. Vzorek I vydělíme vzorkem Q a z výsledné hodnoty získáme arcus tangens, jak to popisuje vzorec 3.2. Výsledkem rovnice bude hodnota jednoho vzorku a výsledný signál tvořený těmito vzorky bude mít dvojnásobnou frekvenci než je modulační frekvence. Frekvenci zjistíme podlé délky amplitudy.

Potřebujeme tedy zjisti délku amplitudy. Budeme hledat bod signálu, kdy přechází z kladné do záporné hodnoty. Tato pozice může být označena jako počáteční s hodnotou 0. Následně pokračuje záporná část amplitudy, na kterou navazuje opět kladná část. Po návratu do záporné části amplitudy dochází k opakování, tedy poslední kladný bod označuje konec jedné amplitudy. Po zjištění počtu vzorků mezi body můžeme pomoci vzorkovací frekvence vypočítat frekvenci námi zjištěné amplitudy. Nicméně, pokud budeme měřit pouze podle jedné amplitudy, výsledek měření může být nepřesný. Pro vetší přesnost budeme zjišťovat vzdálenost začátku více vzdálených amplitud signálu. Čím více bude amplitud mezi jednotlivými body, tím přesnější bude měření, ale zase bude potřeba více vzorků pro zjištění této vzdálenosti.

Testování implementace

Po vytvoření demodulátoru tak, jak je popsán v předchozích částech, jsme provedli testování rychlosti zpracování. Pro zpracování v reálném čase musíme dosáhnout nižšího času než je délka záznamu. Výsledky testování můžeme vidět v tabulce 4.3.

Z výsledků testů je jasné, že potřebný čas pro zpracování je v některých případech delší než délka záznamu. Proto nelze použít tuto implementaci pro zpracování dat v reálném čase. Pro dosažení musíme provést optimalizaci implementace, případně další paralelizaci.

Optimalizace a její testování

Prvek, který budeme optimalizovat je LPF (*Low Pass Filter*). Filtry dolní propusti jsem implementovali jako posuvné registry, kdy při každém vložení nového prvku dochází k od-

CPU	Délka záznamu		
	1 sekunda	5 sekund	10 sekund
AMD A10-4600M	0,9659	5,6533	9,8143
AMD FX-7500	1,0123	5,2043	9,8823
AMD Phenom II X4 840	0,5701	2,8494	5,7323
Intel Atom x5-Z8350	1,1535	5,6788	11,3104

Tabulka 4.3: Časové výsledky demodulace záznamu signálu v sekundách

stranění nejstaršího a z výsledku je vypočítaná průměrná hodnota. Průměrná hodnota tvoří výsledek filtru dolní propusti. Počet hodnot v posuvném registru je nastaven při vytváření tohoto prvku. Každý z těchto filtrů může mít rozdílnou délku. Prvek upravíme na strukturu, která bude obsahovat pole tvořící paměť filtru. Posuvný registr je tvořen pomocí proměnné, do které se zapisuje pozice posledního zapsaného prvku. Není tedy zapotřebí přesouvat všechny hodnoty, ale přepsat pouze jednu. Dalším prvkem je proměnná určující délku pole. Poslední proměnná této struktury je suma všech hodnot v posuvném registru.

Při příchodu nové hodnoty je od sumy odečtena nejdéle uložena hodnota v poli. Pozici této hodnoty určuje proměnná, která byla inkrementována po zapsání nového prvku. Ukazuje na buňku v poli, která je umístěná za posledním uloženým prvkem. Po odečtení této hodnoty od celkové sumy, bude nová hodnota přičtena a uložena na pozici odečteného prvku. Následně dojde k inkrementaci proměnné určující pozici v poli. V případě, že je tato proměnná větší než je počet prvků v poli, je její hodnota nastavena na 0. Posledním provedeným výpočtem bude podíl proměnné obsahující sumu a délku pole, abychom dostali průměrnou hodnotu a tím i výsledek. Tato implementace LPF snižuje počet operací a tak i čas potřebný pro zpracování, než je tomu v případě použití pouze pole a posunu všech prvků pole s následným cyklem pro zjištění sumy.

Po implementaci optimalizace jsme provedli novou sadu testů, jejíž výsledky jsou zobrazeny v tabulce 4.4. Výsledky ukazují, že optimalizací bylo dosaženo rychlejšího zpracování a zároveň tak bylo dosaženo zpracování v reálném čase.

CPU	Délka záznamu		
	1 sekunda	5 sekund	10 sekund
AMD A10-4600M	0,3482	1,7509	3,4818
AMD FX-7500	0,3638	1,8248	3,665
AMD Phenom II X4 840	0,265	1,328	2,673
Intel Atom x5-Z8350	0,5516	2,8423	5,5283

Tabulka 4.4: Časové výsledky demodulace záznamu signálu po optimalizaci v sekundách

Pro ověření zpracování v reálném čase musíme všechny implementované části na sebe navázat a zjisti dobu zpracování zaznamenaného signálu. Jedno vlákno bude nahrávat a posouvat data ze souboru. Druhé vlákno pak bude provádět demodulaci. Ale také jsme provedli testování, kdy všechny tyto operace provádí pouze jedno vlákno. Čas pro zpracování signálu ze záznamu (tabulka 4.5).

CPU	Délka záznamu		
	1 sekunda	5 sekund	10 sekund
AMD A10-4600M	0,4001	2,0738	4,1448
AMD FX-7500	0,3638	1,8248	3,665
AMD Phenom II X4 840	0,2669	1,3516	2,7341
Intel Atom x5-Z8350	0,5881	2,8373	5,729

Tabulka 4.5: Časové výsledky posunu a demodulace záznamu signálu po optimalizaci v sekundách

4.2.2 Korelace

Korelaci můžeme provádět dvěma způsoby a to násobením vzorků referenční sekvence s přijatým signálem nebo porovnáváním. Aby jsme zjistili, jaká z těchto metod je rychlejší provedli, jsme testování na procesorech CPU. Test se skládal z načtení celého souboru s demodulovanými vstupními daty, která byla následně porovnávána s referenční sekvencí. Po uložení celého souboru do paměti a vytvoření referenční sekvence, byly spuštěny korelace o několika cyklech, kterým byl měřený potřebný čas pro zpracování. Výsledky těchto testů můžeme vidět v tabulce 4.6. Tyto testy byly spuštěny pouze jako jedno vlákno na CPU. Všechny korelace měli délku porovnávané sekvence 102 300 znaků. Každý z testů se skládá z více testů rozdílných na počtu provedených korelací. Každý počet korelací je proveden 10krát a následný výsledek je průměrem těchto hodnot. Provedené korelace porovnávají signál pouze s jednou referenční sekvencí. Čas, kterého korelace musí dosáhnout, aby byl signál zpracováván v reálném čase je v tabulce 4.7.

CPU	Počet korelací	Průměrná délka testu (s)		
		Test 1	Test 2	Test 3
AMD Phenom II X4 840	10 000	3,564	3,716	3,156
	50 000	17,842	18,592	15,789
	200 000	71,146	74,219	63,131
Intel Core i7-4790	10 000	1,016	2,144	2,114
	50 000	5,101	10,683	10,570
	200 000	20,470	42,627	42,560
AMD FX-7500	10 000	2,337	2,244	2,799
	50 000	12,645	15,863	14,084
	200 000	48,964	62,919	56,188
Intel Core i7-6700HQ	10 000	1,429	2,243	1,962
	50 000	7,346	12,394	9,934
	200 000	28,706	45,600	39,584

Tabulka 4.6: Časové výsledky testů pro korelaci

Jednotlivé testy prováděli korelaci, která byla implementována těmito způsoby:

Test 1: Implementace korelace pomoci násobení dvou 32-bit celých čísel 1, -1 (integer), výsledek násobení přičten do celkového výsledku.

Test 2: Implementace korelace pomoci porovnávání dvou 32-bit celých čísel 1, -1 (integer), v případě shody inkrementace výsledku, v opačném případě dekrementace.

Test 3: Implementace korelace pomoci porovnávání *true*, *false* (bool), v případě shody inkrementace výsledku, v opačném případě dekrementace

Počet korelací	Čas (s)
10 000	0,005
50 000	0,025
200 000	0,1

Tabulka 4.7: Maximální čas pro zpracování v reálném čase při 2MHz vzorkovací frekvenci

Z výsledků těchto testů je patrné, že nejpomalejší z implementací je porovnávání 32-bitových proměnných. To je dáno tím, že se porovnávají delší čísla, než je tomu v případě testu 3. Rozdíl potřebného času pro korelaci je tedy důležitý mezi testem 1 a 3. Většina z testovacích zařízení provádí rychleji korelaci pomoci násobení, než porovnávání. Nicméně první z procesorů uvedených v tabulce 4.6, dosahuje obrácených výsledků. Proto jsme provedli více testů, z nichž většina výsledků byla rychlejší pro test 1 než test 3. Rychlejší zpracování testu 3 se objevovalo pouze ve vyjímečných případech, záleží tedy na hardwaru. Všechny z těchto výsledků však nedosahují zpracování v reálném čase a pro dosažení je zapotřebí zrychlit výpočet přibližně 200 krát. Optimalizací bychom takovéhoto posunu rychlosti nedosáhly a proto použijeme pro paralelizací GPU. Z důvodu lepších časových výsledků budeme implementovat korelaci pomoci násobení.

Korelaci budeme provádět na grafických procesorech s použitím standardu OpenCL, který je popsán v podkapitole 2.4.3. Pro implementaci jsme použili OpenCL verze 1.2, který je podporován grafickými kartami ATI i nVidia. Lokalizační data získáváme z bloku, který demoduluje data z přijímaného signálu na 1 a -1. Výsledky jsou ukládány do pole typu `short` tedy 16-bitového celého čísla.

Inicializace OpenCL zařízení

Inicializaci a obsluhu OpenCL zařízení zajišťuje hostitelský zařízení, což je procesor CPU. Součástí inicializace je zjištění informací a adres zařízení. Dalším krokem je přeložení programu na zařízení. Program, který se provádí na OpenCL zařízení je překládán při každém spuštění. Každé zařízení si tedy překládá program zvlášť, a zde muže být rozdíl podle velikosti instrukční sady a nebo jiných vlastnostech zařízení.

Poslední částí, kterou musíme inicializovat je globální paměť na OpenCL zařízení. Tato paměť slouží právě pro přenos dat mezi host zařízení a OpenCL zařízení. Příklad 5.3 zobrazuje příklad posloupnosti operací a jejich závislost tak, aby bylo možné provádět výpočty na OpenCL zařízení.

Příklad 5.3 zobrazuje jednu z možností inicializace jednoho OpenCL zařízení. Bude se jednat o grafickou kartu, což určuje parametr `CL_DEVICE_TYPE_GPU` na řádku 7. V tomto příkaze je také vidět, že chceme dostat ukazatel pouze na jedno zařízení, které zobrazuje následující parametr. Pro získání ukazatele na OpenCL zařízení potřebujeme znát ID celé platformy, které se získává na řádku 3. Další úlohou je vytvořit kontext. Jedná se o spojení mezi OpenCL zařízením a host zařízením. V jednom kontextu může být umístěno více

OpenCL zařízení. V případě, že je více zařízení v jenom kontextu, můžeme provádět synchronizaci pomoci událostí (event). Kdyby ale byla OpenCL zařízení v jiných kontextech, synchronizaci by muselo provádět host zařízení. Vytvoření kontextu pro jedno zařízení je na řádku 11.

Další úlohou, kterou potřebujeme udělat, je inicializace programu, tedy zdrojového kódu, který bude OpenCL zařízení vykonávat. Tento zdrojový kód může být přečten z textového souboru nebo být inicializován jako řetězec v programu, který zařízení inicializuje. Následně docház k přenosu zdrojového kódu do kontextu tak, aby všechna zařízení k tomuto kódu měla přístup. Posledním příkazem je spuštěné přeložení programu na OpenCL zařízení(řádek 16). Pokud v jednom kontextu je více OpenCL zařízení, musí proběhnout překlad na každém zařízení zvlášť.

Poslední úlohu proto, abychom mohli spouštět přeložený program, je získání ukazatele na funkci. Součástí jednoho programu může tedy být více funkcí. Následně je vytvořena fronta úloh, do které budou vkládány ukazatele na funkci společně s parametry.

Přenos dat a synchronizace

Součástí příkladu 5.3 je i inicializace paměti pro přenos dat mezi zařízeními a událostí pro synchronizaci mezi zařízeními v jednom kontextu. Na řádku 28 je napsán příkaz pro vytvoření sdílené paměti, která umožňuje přenos dat mezi OpenCL zařízením a host zařízením. Paměť je tvořená v rámci kontextu a v tomto případě se vytváří na OpenCL zařízení. Paměť je nastavena pouze pro čtení, tedy hostovací zařízení může zapisovat data, ale OpenCL zařízení může pouze data číst. Tato paměť bude použita pro přenos dat mezi zařízeními, kde na základě 2. parametru určujeme jaká práva bude mít jaké zařízení.

Další příkaz po vytvoření paměti je příkaz pro vytvoření události. Událost slouží pro synchronizaci mezi zařízeními v jednom kontextu. Synchronizaci mezi kontexty by muselo zařizovat hostovací zařízení. Událost může nabývat několika stavů. Pro účel synchronizace budeme používat příkaz, který kontroluje a čeká na stav `CL_COMPLETE`. Události použijme pro synchronizaci mezi zapisováním dat, čtením dat a vykonávání výpočtu na GPU.

Výpočet korelace bude probíhat na GPU. Nahrávat data na GPU však musí hostovací zařízení. Tuto činnost může provádět nové vlákno nebo vlákno, které provádí demodulaci.

Implementace korelace

Implementace korelace byla prováděna na grafických procesorech s použitím standardu OpenCL. Při paralelizaci na GPU, byl jako první implementován základní postup, který probíhal v testu 1 popsáným pod tabulkou 4.6. V této implementaci však byly násobeny mezi sebou 16-bitová celá čísla. Implementaci je možné vidět v příkladu 5.4.

Funkce se spouští na grafickém procesoru, kde globální ID tvoří dvě hodnoty. První hodnotou je počet iterací na jeden buffer, to je pozice začátku porovnávání sekvence v poli inBuffer. Druhou hodnotu tvoří počet masek. Jedná se o vytvořené referenční masky, které jsou umístěny do pole. Na příkladu je vidět, že funkce obsahuje dva cykly, ale také i několik operací násobení. Tabulka 4.8 zobrazuje rychlost korelace na grafických procesorech. Počet korelací popisuje první hodnotu z globálního indexu pro spuštění korelace. Čas, kterého musí korelace dosáhnout, aby byl signál zpracováván v reálném čase je v tabulce 4.7.

Výsledky testů ukazují, že bylo v některých případech dosaženo zpracování v reálném čase ale pouze pro jeden vysílač. Se zvyšujícím počtem vysílačů dochází k zvyšování času pro zpracování. V případě korelace 10 000 vzorků při jednom spuštění nedochází k zvyšování času pro zpracování. Jedna událost plánovaná do fronty má nízký počet iterací a tak většinu

GPU	Počet korelací	Počet vysílačů			
		1	3	5	7
Radeon R9 290X	10 000	20,2332	19,75534	19,7242	19,7132
	50 000	20,2348	41,3557	62,05422	82,8113
	200 000	60,7461	144,0364	244,0634	328,3572
Radeon R9 280X	10 000	19,9012	20,1673	22,1595	39,5682
	50 000	22,1717	60,8277	89,9108	125,9047
	200 000	79,6832	210,2395	354,5591	477,4559
Radeon R7 M260DX	10 000	32,1705	93,7205	153,344	212,1449
	50 000	149,821	434,524	734,112	1059,19
	200 000	579,98	1730,23	2983,23	4182,68

Tabulka 4.8: Časové výsledky testů (ms) korelace implementovaného z příkladu 5.4

času zabírá inicializace a spuštění události. Nicméně ani touto možností nebylo dosaženo zpracování v reálném čase.

Pro dosažení zpracování v reálném čase jsme proto museli optimalizovat implementaci. Při rozboru implementace z příkladu 5.4 pomoci programu CodeXL vyšla jako jedna z nejsložitější operací právě násobení. Proto v implementaci 5.5 bylo sníženo počet těchto operací a dále sloučení porovnávání tak, aby se nezvyšovala časová složitost při vyšším počtu vysílačů. Tento příklad používá jinou proměnnou v druhé globání hodnotě. Tuto hodnotu tvoří číslo 1023, což je počet znaků v jedné generované masce. Následně pomoci sčítání je vytvořen součet všech vzorků v jednom bitu z přijatého signálu, který je nakonec porovnán, a přičten nebo odečten k celkovému výsledku.

Implementace programu byla testována a výsledky je možné vidět v tabulce 4.9, ve které jsou zobrazeny časy pro provedení korelace na GPU. Pro dosažení zpracování v reálném čase musí čas dosahovat nižších hodnot oproti tabulce 4.7.

GPU	Počet korelací	Počet vysílačů			
		1	3	5	7
Radeon R9 290X	10 000	2,9744	3,0425	3,1807	3,3624
	50 000	14,5298	14,9124	18,3272	23,2144
	200 000	59,7752	68,4022	79,0105	89,7841
Radeon R9 280X	10 000	3,8692	4,0709	4,2155	4,4424
	50 000	18,8452	22,202	27,0028	27,0737
	200 000	81,6645	93,778	109,5818	125,4279
Radeon R7 M260DX	10 000	14,7698	29,961	41,967	60,3421
	50 000	140,002	205,537	270,5778	336,6084
	200 000	559,204	809,213	1053,869	1309,231

Tabulka 4.9: Časové výsledky testů (ms) korelace implementovaného z příkladu 5.5

Po optimalizaci je zřejmé, že došlo ke splněné podmínky pro zpracování v reálném čase, na některých grafických kartách. Při této implementaci nedochází k vysokým časovým rozdílům mezi počtem vysílačů, jak tomu bylo v příkladu 5.4. Poslední úpravou programu pro další zrychlení je možnost využití více grafických karet pro paralelní zpracování.

Pro tento účel je vytvořena struktura, která obsahuje ukazatele na vstupní a výstupní buffer, události pro synchronizaci mezi GPU a host zařízením a také mutexy pro synchronizaci mezi zapisováním dat a čtením. Ke každému zařízení jsou přiřazeny minimálně dvě tyto struktury tak, aby s jednou strukturou mohlo pracovat GPU a s druhou host zařízení. Tyto struktury byly následně přiřazené střídavě k více zařízením, aby došlo k rozložení zátěže. V případě rozdílného výpočetního výkonu grafických karet můžeme počet struktur upravit, aby došlo k úplnému využití výpočetního výkonu zařízení. Zrychlil se tak výpočet a to přibližně na součet rychlostí obou zařízení.

4.2.3 Zjištěni času mezi vrcholy

Poslední částí pro získání rozdílů časů mezi jednotlivými vysílači je vyhledávání vrcholů jednotlivých korelačních křivek. Informace jsou uložené v poli na grafické kartě. Tyto informace jsou po dokončení korelace určité části signálu překopírovány zpět do paměti CPU. Konkrétně si data zkopíruje vlákno, které následně tato data prohledává.

Vyhledávání vrcholů probíhá jak v kladné tak i záporné části, kde výsledek je pozice nalezeného vrcholu. Každá korelační křivka má vlastní vrchol, který právě hledáme. Počet vzorků mezi těmito vrcholy určuje právě rozdílný čas mezi jednotlivými vysílači. Tento počet následně bude vydělen vzorkovací frekvencí a tak získáme časový údaj. Provedli jsme testování implementace na CPU. Součástí testu je rozdílný počet vysílačů a výsledky popisuje tabulka 4.10.

CPU	Délka záznamu / Počet vysílačů					
	5 sekund			10 sekund		
	3	5	7	3	5	7
AMD A10-4600M	0,4378	0,7348	0,9812	0,8915	1,4168	2,0165
AMD FX-7500	0,3477	0,5953	0,8278	0,688	1,1863	1,6444
AMD Phenom II X4 840	0,3679	0,6183	0,8675	0,7286	1,2393	1,7168
Intel Atom x5-Z8350	0,7372	1,2988	1,7641	1,4801	2,5592	3,5206

Tabulka 4.10: Časové výsledky hledání vrcholů v sekundách

4.3 Implementace průběžného systému

Průběžný systém jsme také rozdělili do několika na sobě navazujících částí (obrázek 3.5). Prvními dvěma bloky jsou čtení dat z přijímače a posun signálu. Implementace obou těchto bloků je popsána v podkapitole 4.1. Součástí podkapitoly je popsána a otestována možnost paralelizace a případného sjednocení obou bloků. Na základě testů jsme tyto dva bloky spojili, což snížilo potřebný počet vláken pro tyto dva bloky.

V této podkapitole se budeme soustředit na implementaci zpracování přijatých dat, případnou optimalizaci a paralelizaci, abychom dosáhli zpracování v reálném čase. Pro implementaci systémů pro průběžnou synchronizaci přijímaného signálu s referenční sekvencí, byl použit postup popsaný v podkapitole 3.2.3 a zobrazen na obrázku 3.3. Nejdříve bylo implementováno zpracování signálu z jednoho vysílače. Toto zpracování probíhalo ze záznamů uložených v souborech.

Ze schématu uvedeného na obrázku 3.3 je zřejmé, že provádí porovnávání referenční sekvence s přijatým signálem. Kontrolu posunu referenční sekvence provádí prvek `Code`

`diskriminator` a kontrolu posuvu provádí každých 300 vzorků. Stejnou velikost mají také filtry dolní propusti, do kterých se vkládají porovnané výsledky signálu s referenční posunutou sekvencí. Tyto filtry budou tvořit pouze proměnné, které budou obsahovat sumu a vždy po 300 vzorcích se provede jejich průměr. Průměrné hodnoty jsou označené I_E, I_P, I_L, Q_E, Q_P, Q_L. `Code diskriminator` na základě těchto výsledků provede posun referenční sekvence o jeden vzorek. Výsledný posun určuje zpoždění signálu.

Následně potřebujeme zjistit, jestli je referenční sekvence synchronizována se signálem. Tuto část zajišťuje prvek `PLL lock detect`, který provádí kontrolu synchronizace každých 20 000 vzorků. Součástí tohoto prvku je filtr dolní propusti (LPF) a fázový závěs. Vstupem je součet středových hodnot I a Q porovnaných s referenční sekvencí, které jsou následně vynásobeny s referenční sinusovou křivkou. pro odstranění parazitní amplitudy přijatých dat. Následně tento součet je veden do fázového závěsu, který vytvoří hodnotu 1 nebo -1, který je uložen do filtru dolní propusti. Kapacita tohoto filtru je stejná jako počet vzorků, po kterých je provedena kontrola. Následný výsledek LPF, určuje zda-li je signál synchronní. Signál je synchronní pokud průměrná hodnota je vyšší jak 0,4. V případě, že výsledek je nižší dojde k jednorázovému posunu referenční masky o deset vzorků vpravo.

Po dosažení synchronizace signálu s referenční sekvencí je výsledný posun, označující zpoždění signálu z vysílače, vkládán do dalšího filtru dolní propusti, který umožňuje určit zpoždění přesněji než pouze na jeden vzorek. Všechny tyto hodnoty počtu vzorků než je provedena akce `Code diskriminator`, `PLL lock detect`, ale také vzdálenost hodnot *early, prompt a late*, zle měnit pro zvýšení přesnosti. Poslední částí je zjištění frekvence amplitudy parazitního jevu dat. Tento problém je ve schématu řešen pomoci prvku FFT, nicméně námi použitá implmenentace pro zjištění této frekvence je stejná jako ve statickém systému. Výsledky testů jsou zobrazeny v tabulce 4.11.

CPU	Délka záznamu / Počet vysílačů					
	5 sekund			10 sekund		
	3	5	7	3	5	7
AMD A10-4600M	0,4378	0,7348	0,9812	0,8915	1,4168	2,0165
AMD FX-7500	0,3477	0,5953	0,8278	0,688	1,1863	1,6444
AMD Phenom II X4 840	1,5541	2,3082	3,5042	3,2194	4,9429	7,0395
Intel Atom x5-Z8350	3,5529	4,5353	6,2688	6,3842	9,0478	12,713

Tabulka 4.11: Časové výsledky průběžného systému v sekundách

4.4 Struktura, zpracování XML a určení polohy přijímače

V předchozích podkapitolách jsme provedli implementaci zpracování signálu lokalizačních systémů. Výsledkem tohoto zpracování jsou rozdílné časy přijetí sekvence z jednotlivých vysílačů. V této podkapitole si popíšeme, jak jsou informace o lokalizačním systému uloženy, jaké přesnosti bude implementace dosahovat. Ale také otestujeme, zda výpočet polohy musí být implementován jako samostatné vlákno nebo může být s některým již implementovaným vláknem spojen.

Informace o lokalizačním systému jsou uloženy v souboru typu XML. V tomto souboru jsou informace s hodnotou nosné frekvence, vzorkovací frekvence, ale i časové délce jedné posílané hodnoty. Dalšími informacemi uloženými v tomto souboru jsou polohy jednotlivých

vysílačů a hodnoty pro vytvoření referenční sekvence. Na základě těchto informací je potom možné získat polohu přijímače. Pro získání informací z XML souboru jsme použili knihovnu libxml. Tato knihovna je připravena pro Windows ale i Linux, takže může byt program upraven pro práci na více operačních systémech.

Určení polohy probíhá pomoci časových rozdílů získaných ze tří vysílačů. Tento výpočet je prováděn podle výpočtů uvedených v podkapitole 2.3.4. Podíváme-li se na přesnost této lokalizace, je především u statického systému ovlivněna maximální vzorkovací frekvencí, kterou RTL-SDR zařízení dosahuje. Při zvýšení této frekvence dojde i ke zvýšení časové náročnosti na zpracování signálu. Podle vzorkovací frekvence 2MHz můžeme určit, že přesnost se bude pohybovat kolem 150m. Jde totiž o časový rozdíl mezi jednotlivými vzorky, a tento rozdíl při vynásobení s rychlostí světla určuje přesnost. V případě průběžného systému však nedochází k posunu pouze o jednotlivé vzorky. Proto jeho přesnost je při stejné vzorkovací frekvenci vyšší než u statického systému. Přesnost systémů také lze zvýšit pomoci postupu zvaného multilaterace, pro který bychom potřebovali více než tři vysílače. Dále také nebyla implementována možnost, kdy jsou všechny vysílače v jedné přímce. Tato možnost se pro tyto systémy téměř nevyužívá.

Pro správné určení polohy je dále zapotřebí zjistit z jakých vysílačů přijímáme signál a určit, který z těchto vysílačů je středový, levý a pravý, aby bylo možné určit polohu. Musíme tedy zkontrolovat, ze kterých vysílačů přijímáme signál. Tyto informace zjistíme projitím pole struktur, které jsme si vytvořili. V této struktuře je napsaný čas, proměnná, která určuje jestli signál byl nalezen, ale i pozice vysílače. Zjistíme tak, ze kterých vysílačů přijímáme signál a následně provedeme porovnání poloh vysílačů, abychom mohli určit jejich vzájemnou pozici. Při tomto testování zároveň provedeme testování počtu přijímačů, abychom věděli jestli máme dostatek informací k určení polohy.

Implementovali jsme algoritmus pro výpočet polohy přijímače. Součástí této implementace je také určení správného vzájemného rozložení vysílačů tak, aby bylo zajištěno, že levý vysílač bude nalevo od středového vysílače a u pravého na pravé straně. Implementaci jsme otestovali a výsledky jsou v tabulce 4.12. K výpočtu polohy však nedochází pro každý přijatý vzorek, ale pouze několikrát za vteřinu. V případě statického systému je to jednou za opakování sekvence a uvedli jsme si, že délka sekvence je 102 300 vzorků. Při vzorkovací frekvenci 2MHz tak dochází přibližně 20 krát k výpočtu polohy přijímače. V případě průběžného systému se výsledek opakuje každých 100 vzorků, budeme tedy muset vypočítat polohu 20 000 krát za vteřinu.

CPU	Počet výpočtů		
	50 000	100 000	200 000
AMD A10-4600M	0,0446	0,0964	0,1870
AMD FX-7500	0,041	0,0833	0,1674
AMD Phenom II X4 840	0,0361	0,0717	0,1437
Intel Atom x5-Z8350	0,0875	0,1728	0,3483

Tabulka 4.12: Časové výsledky výpočtů polohy v sekundách

Výsledky v tabulce 4.12 určují, že není potřeba provádět žádné optimalizace či paralelizace. Výpočet zvládají v reálném čase a můžeme také otestovat, zda nejde výpočet polohy spojit s jiným vláknem. V případě průběžného systému zde není žádné další vlákno, které by pracovalo s výsledky pro všechny vysílače. Statický systém však používá vyhledávání vrcholů, které zpracovává výstup korelátoru. Výsledky testů pro vyhledávání vrcholů ko-

relačních křivek můžeme vidět v tabulce 4.10. Provedeme sloučení vlákna pro vyhledávání vrcholů a výpočet polohy. Tabulka 4.13 zobrazuje výsledky testů. Výsledky zobrazují, že je možné použít tuto implementaci pro zpracování v reálném čase

CPU	Délka záznamu / Počet vysílačů					
	5 sekund			10 sekund		
	3	5	7	3	5	7
AMD A10-4600M	0,4361	0,7361	0,9846	0,8928	1,394	1,9258
AMD FX-7500	0,3465	0,5893	0,8169	0,693	1,193	1,6443
AMD Phenom II X4 840	0,3661	0,6146	0,8538	0,7311	1,2328	1,7111
Intel Atom x5-Z8350	0,752	1,2857	1,7419	1,5622	2,5484	3,4893

Tabulka 4.13: Časové výsledky hledání vrcholů a výpočtu polohy v sekundách

4.5 Dílčí závěr

Testování probíhalo s uloženým záznamem v souboru. Záznam pro statický systém byl tvořen signálem z jednoho vysílače, který vysílal jednu sekvenci. Signál jednoho vysílače byl tvořen pomoci generátoru signálu. Program byl tvořen a laděn pro jedenu sekvenci, u kterého se povedlo získat vrcholy. Následně bylo testováno, jak dlouho trvá zpracovat záznam. Pro tento účel bylo vytvořeno několik záznamů o jiné délce trvání, všechny tyto signály však tvořila jedna sekvence. V případě průběžného systému jsme použily signál tvořený ze dvou dvou vysílačů. Generátorem signálu jsme vysílali dvě sekvence, které se navzájme rušily. Statický i průběžný systém mají společné části jako přijímání a posun signálu, ale také výpočet polohy přijímače.

Provedli jsem paralelizaci statického systému. Na procesoru CPU potřebujeme pro zpracování minimálně dvě vlákna. Jedno vlákno přijímá a demoduluje data. Demodulovaná data v dosahují hodnoty 1 a -1 a jsou kopírována do paměti grafické karty. Grafická karta provádí korelaci, tedy porovnání demodulovaného signálu s vytvořenými referenčním sekvencemi. Druhé vlákno na CPU potom čte výsledky korelace prováděné na GPU a určuje polohu přijímače. Z výsledků paralelizace korelace provedené na GPU v tabulce 4.9 je jasně patrné, že bylo dosaženo rychlejšího zpracování při použití grafických procesorů, než tomu bylo u CPU (tabulka 4.6). Dále při porovnání výsledků implementace korelace 4.9 s podmínkami pro zpracování v reálném čase 4.7 je vidět, že u některých grafických karet bylo dosaženo zpracování v reálném čase.

Testování statického systému jsme prováděli se signálem z jednoho vysílače. Při použití implementace pro více signálů docházelo ke ztrátám informací z hlediska vzájemného rušení. Každý z vysílačů vysílal jiné sekvence, které se navzájem rušili. Toto rušení následně způsobilo problém při demodulaci signálu pomoci Costasovy smyčky. Vlivem rušení a špatné demodulace tak není možné pomoci korelace najít čas kdy byl signál vyslán. Tento problém se vyskytl už u použití dvou vysílačů. Pro určení polohy je však zapotřebí nejméně tří vysílačů. Proto tuto metodu nelze použít pro určení polohy. Nicméně implementace programu byla upravena tak, že program pracuje s jedním signálem a vzájemným posunem referenční generované sekvence. Na základě takto generovaných sekvencí je určena poloha. Program statického systému tedy slouží pro simulační účely.

V případě průběžného systému je délka změny fáze, pro jednotlivé znaky sekvence, menší. Implementace zpracování signálu je přizpůsobena, aby minimalizovala ztrátu dat způsobenou vzájemným rušením vysílačů. Z výsledků testování je vidět, že některé z procesorů CPU zvládají určit polohu v reálném čase. Se zvyšujícím se počtem vysílačů v lokalizačním systému však stoupá počet vláken a tak i výpočetní náročnost. Některé procesory však nezvládají zpracovávat data v reálném čase ani pro jeden vysílač. Proto by bylo vhodné provést paralelizaci na GPU. K dispozici pro testování jsme měli pouze jeden generátor signálu, který je schopný generovat dvě sekvence. Pro testování jsme používali signál ze dvou vysílačů. Pro simulaci více vysílačů byla vytvořena vlákna, která porovnávala stejnou sekvenci.

Kapitola 5

Závěr

Projekt byl zaměřen na zpracování signálu a výpočet polohy přijímače s využitím softwarově definovaného rádia. Jednou z podmínek bylo určení polohy v reálném čase za použití počítačů. Výsledkem je zobrazení hodnoty vypočítané polohy. Pro zpracování lokalizačních dat jsme vytvořili dvě implementace. První z implementací je označena jako statický systém. Zde nedocházelo k posuvu referenční sekvence. Pro dosažení zpracování dat v reálném čase musela být prováděna paralelizace implementace jak na CPU tak i na GPU. V případě druhého systému, který jsme označili jako průběžný, dochází k posuvu vygenerované referenční sekvence. Dochází tak k synchronizaci signálu a referenční sekvence a výsledný posuv této sekvence označuje zpoždění signálu.

Pro definování vlastností lokalizačního systému jsou použita data, která jsou uložena v souboru formátu XML. To umožňuje použití programů pro více lokalizačních systémů. Vypočítaná poloha je v rámci lokalizačního systému. Dále tato implementace umožňuje upravovat pozice vysílačů lokalizačních systémů bez nutnosti úpravy zdrojového kódu programu.

Při implementaci systému statické synchronizace bylo dosaženo přibližně 500 násobného zrychlení při provádění výpočtů i na grafické kartě. Díky tomuto zrychlení výpočtu jsme dosáhli zpracování v reálném čase. Tato hodnota se může měnit v závislosti na hardwaru výpočetního zařízení. Statický systém při použití metody CDMA při přenosu signálu, nelze použít pro lokalizaci vlivem vzájemného rušení vysílačů. Toto vzájemné rušení způsobuje problém při demodulaci signálu pomoci Costasovi smyčky. Proto tato implementace zůstala pro simulační účely, kde používáme nahrávku signálu jednoho vysílače. Přijímač vytváří sekvenci generovanou vysílačem, kterou posune. Tím je simulováno více vysílačů a vzájemné zpoždění signálu z těchto vysílačů. Pro odstranění vzájemného rušení vysílačů můžeme použít metodu multiplexu FDMA. Potom bychom však museli mít více přijímačů, přesněji pro každý vysílač jeden přijímač. Zvyšuje se tak počet přijímaných signálů, které je potřeba zpracovat a dále se zvyšuje počet potřebných frekvencí pro jeden lokalizační systém. Z těchto důvodů zůstala tato implementace pouze pro simulační účely.

U průběžného systému byla paralelizace prováděna pouze na CPU, a lze jí použít pro určení polohy v reálném čase u některých procesorů s omezeným počtem vysílačů. Implementaci jsme testovali v laboratoři, kde jsme měli generátor signálu přímo připojený k USB přijímači. Generátor signálu vytvářel signál ze dvou vysílačů. Signál se tedy skládal ze dvou různých sekvencí. Testováním pak bylo ověřeno, že implementace je schopna zpracovat signál, kde dochází k vzájemnému rušení ze dvou vysílačů. Pro dosažení rychlejšího zpracování by bylo vhodné provést paralelizaci implementace na grafických procesorech. Vhodná část pro paralelizaci jsou jednotlivé korelátory, které synchronizují referenční sek-

venci s přijímaným signálem. Pro reálné použití této implementace je potřeba provést další testování s vyšším počtem vysílačů. Dále by bylo potřeba provést testování kde vysílače nejsou přímo připojené k přijímači a tak by docházelo k rušení signálu vlivem prostředí případně Dopplerova jevu.

Přesnost výpočtu polohy u statického systémů je ovlivněna vzorkovací frekvencí. Vzorkovací frekvence je pro různá RTL-SDR zařízení rozdílná. Průběžný systém však není v přesnosti určení polohy tolik závislý na vzorkovací frekvenci přijímače jako statický systém. Čím vyšší vzorkovací frekvence bude, tím vyšší budou nároky na hardware pro zpracování v reálném čase. Na druhou stranu však díky vyšší vzorkovací frekvenci budeme schopni přesněji určit polohu přijímače. Pro zlepšení přesnosti bez zvýšení vzorkovací frekvence můžeme použít také metodu multilaterace. Pro použití této metody však potřebujeme víc než tří vysílačů. Implementace multilaterace za použití průběžného systému, společně se zvyšováním vzorkovací frekvence a počtu vysílačů pro zvýšení přesnosti lokalizace v závislosti na využití hardwaru, by mohlo být pokračováním této práce.

Literatura

[1] RTL-SDR: Block Diagram and Information . [online], 2001, Naposledy navštíveno 18. 1. 2017.
URL http://aaronscher.com/wireless_com_SDR/rtl_sdr_info.html

[2] *Základy teorie pasivních systémů I*, kapitola Časoměrně – hyperbolická metoda určování polohy zdroje signálu. Univerzita obrany Brno, 2007.

[3] Cvičení 3 – Vlákna (threads). ČVUT - fakulta elektrotechnická [online], 2009, Naposledy navštíveno 19. 1. 2017.
URL https://support.dce.felk.cvut.cz/pos/cv3

[4] Introduction to OpenCL™ Programming. [online], 2010, Naposledy navštíveno 23. 1. 2017.
URL https://www.khronos.org/registry/OpenCL/specs/opencl-2.2.pdf

[5] R820T Datasheet . Thaiwan: Rafael Microelectronics [online], 2011, Naposledy navštíveno 21. 12. 2016.
URL
http://superkuh.com/gnuradio/R820T_datasheet-Non_R-20111130_unlocked.pdf

[6] What is I/Q Data? [online], 30. 3. 2016, Naposledy navštíveno 18. 1. 2017.
URL http://www.ni.com/tutorial/4805/en/

[7] SDR Radio. [online], Naposledy navštíveno 12. 3. 2017.
URL http://www.sdr.ipip.cz

[8] About RTL-SDR. [online], Naposledy navštíveno 15. 3. 2017.
URL http://www.rtl-sdr.com/about-rtl-sdr

[9] CUDA. GPGPU [online], Naposledy navštíveno 21. 1. 2017.
URL http://gpgpu.org/developer/cuda

[10] STREAM. GPGPU [online], Naposledy navštíveno 21. 1. 2017.
URL http://gpgpu.org/developer/stream

[11] Realtek RTL2832U . Taiwan: Realtek Semoconductor Co. [online], Naposledy navštíveno 21. 12. 2016.
URL http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PFid=35&Level=4&Conn=3&ProdID=257

[12] XML tutorial. W3schools.com [online], Naposledy navštíveno 24. 1. 2017.
URL http://www.w3schools.com/xml

[13] Barney, B.: POSIX Threads Programming. Lawrence Livermore National Laboratory [online], Naposledy navštíveno 30. 12. 2016.
URL https://computing.llnl.gov/tutorials/pthreads

[14] Bourd, A.: The OpenCL Specification. [online], 2016, Naposledy navštíveno 23. 1. 2017.
URL http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/01/Introduction_to_OpenCL_Programming-201005.pdf

[15] Frenzel, L.: Fundamentals of Communications Access Technologies: FDMA, TDMA, CDMA, OFDMA, AND SDMA. [online], 22. 1. 2013, Naposledy navštíveno 13. 3. 2017.
URL http://electronicdesign.com/communications/fundamentals-communications-access-technologies-fdma-tdma-cdma-ofdma-and-sdma

[16] Hagemann, E.: The Costas Loop - An Introduction. [online], 2001, Naposledy navštíveno 18. 1. 2017.
URL http://dsp-book.narod.ru/costas/DSP010315F1.pdf

[17] pplk. Ing. Petr Hubáček: *Optimalizace topologie TDOA systémů z hlediska přesnosti utčení polohy cíle*. Dizertační práce, Univerzita obrany v Brně, Fakulta vojenských technologií, 2010.

[18] Littschwager, T.: GPGPU: 200x rychleji než CPU. *CHIP*, 2007.

[19] Michlovský, J.: *Standard OpenCL*. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011.

[20] Mohammed Zishan Ansari, Shobhit Mangla, Arijit Chattopadhyaya, Deepak Gahlawat, Puneet Shah, Vaibhav Singh: Gold Code Sequences. Technická zpráva, National Institute of technology, Durgapur, 2008.
URL
http://home.chosun.ac.kr/~multimedia/shyrapba/Gold_Seq_Project[1].pdf

[21] Navipedia: Delay Lock Loop (DLL). [online], 2014, Naposledy navštíveno 16. 3. 2017.
URL http://www.navipedia.net/index.php/Delay_Lock_Loop_(DLL)

[22] Roppel, T. A.: BPSK - BINARY PHASE SHIFT KEYING. [online], Naposledy navštíveno 13. 3. 2017.
URL http://www.eng.auburn.edu/~roppeth/courses/TIMS-manuals-r5/TIMS%20Experiment%20Manuals/Student_Text/Vol-D1/D1-08.pdf

[23] Vojnar, T.: Operační systémy: Synchronizace procesů. [online], 2016, Naposledy navštíveno 19. 1. 2017.
URL https://www.fit.vutbr.cz/study/courses/IOS/public/prednasky/ios-prednaska-06.pdf

[24] Wikipedia: Barker code — Wikipedia, The Free Encyclopedia. [online], 2017, Naposledy navštíveno 5. 5. 2017.
URL
"https://en.wikipedia.org/w/index.php?title=Barker_code&oldid=774143559"

[25] Wikipedia: PCI Express — Wikipedia, The Free Encyclopedia. [online], 2017, Naposledy navštíveno 8. 5. 2017.
URL
"https://en.wikipedia.org/w/index.php?title=PCI_Express&oldid=776079228"

[26] Wikipedia: Video card — Wikipedia, The Free Encyclopedia. [online], 2017, Naposledy navštíveno 8. 5. 2017.
URL
"https://en.wikipedia.org/w/index.php?title=Video_card&oldid=778475532"

[27] Wikipedia: Gold code — Wikipedia, The Free Encyclopedia. [online], Naposledy navštíveno 14. 3. 2017.
URL https://en.wikipedia.org/wiki/Gold_code

[28] Zídek, K.: *Akcelerace výpočtů prostřednictvím GPU*. Diplomová práce, Mendelova univerzita v Brně, Provozně ekonomická fakulta, 2012.

Počítačové sestavy pro testování

Sestava 1

CPU: AMD A10-4600M APU with Radeon(tm) HD Graphics, 2,3 GHz, 4 jádra

RAM: 8 GB DDR3, 1 600 MHz

GPU: AMD Radeon HD 7660G 1GB + HD 7600M Dual Graphics

HDD: 256GB SSD + 1TB HDD

OS: Windows 10 Pro 64bit

Sestava 2

CPU: AMD FX 7500, 2,1GHz, 4 jádra

RAM: 8GB DDR3L, 1 600 MHz

GPU: AMD Radeon R7 M260 2GB + AMD Radeon R7 Dual Graphics

HDD: 256GB SSD + 1TB SSHD

OS: Windows 10 Pro 64bit

Sestava 3

CPU: Intel Core i7-4790, 3,6GHz, 4 jádra

RAM: 8GB DDR3, 1 600 MHz

GPU: nVidia GeForce GTX 760 2GB

HDD: 1TB HDD

OS: Windows 7 Home 64bit

Sestava 4

CPU: Intel Core i7-6700HQ, 2,6GHz, 4 jádra

RAM: 8GB DDR4, 2 400 MHz

GPU: nVidia GeForce GTX 960M 4GB

HDD: 1TB SSHD

OS: Windows 10 Pro 64bit

Sestava 5

CPU: Intel Atom x5-Z8300, 1,44GHz, 4 jádra

RAM: 4GB DDR3

GPU: Intel HD Graphics

HDD: 64GB Flash

OS: Windows 10 Home 64bit

Sestava 6

CPU: AMD Phenom II X4 840, 3,2 GHz, 4 jádra

RAM: 8 GB DDR3, 1 333 MHz

GPU: AMD Radeon HD 5450 1GB

HDD: 2TB HDD

OS: Windows 10 Pro 64bit

Sestava 7

CPU: Intel Core i5-4440S, 2,8GHz, 4 jádra

RAM: 8 GB DDR3, 1 600 MHz

GPU: 3x AMD Radeon R9 290X + 2x AMD Radeon R9 280X

HDD: 80 GB HDD

OS: Windows 10 Pro 64bit

Přiklady implementace

```
// globalni pamet
  uint8_t *mem; //sizeof(BUFF_LEN)
  pthread_mutex readyWrite_m, readyRead_m;

// prikaz pro zapnuti asychrnoniho cteni, umisten v hlavni funkci
rtlsdr_read_async(dev, getData, (void *)NULL, BUFF_COUNT, BUFF_LEN);

// funkce spoustena asynchronnim cteni pri dokonceni naplneni jednoho z
    bufferu
static void getData(unsigned char *buf, uint32_t len, void *ctx){
  pthread_mutex_lock(&readyWrite_m);
  memcpy(mem, buf, len);
  pthread_mutex_unlock(&readyRead_m);
}

//funkce popisujici cinnost druheho vlakna, ktere zapisuje data do souboru
static void* saveData(){
  FILE *output;
  output = fopen("..\\data\\zaznam.dat", "w");

  while (true) {
    pthread_mutex_lock(&readyRead_m);
    fwrite(mem, sizeof(uint8_t), REC_BUFF_LEN, output);
    pthread_mutex_unlock(&readyWrite_m);
  }
}
```

Příklad 5.1: Možnost implementace programu pro získávání dat z RTL-SDR

```
// globalni pamet
typedef struct receiverToDemodulator {
  float *dataI, *dataQ;  //velikost definovana hodnotou IQ_BUFFER_SIZE
  bool dataFull, signalFull;

  pthread_cond_t condReadData;
  pthread_mutex_t mutReadData;
  pthread_mutex_t mutMemReady;
} receiverToDemodulator;

receiverToDemodulator *recBuffers; //pocet definovan hodnotou IQ_BUFFER_COUNT

// funkce pro posun signalu
static void* signalShift(){
  int writeToPlace = 0, writeRecBuffer = 0;

  while (!end) {
    pthread_mutex_lock(&receiverMemRead);
    for (int i = 0; i < REC_BUFF_LEN * 2; i += 2)
    {
      //zapis dat do globalni pameti
      recBuffers[writeRecBuffer].dataI[writeToPlace] = inputBuffer[i] -
    127.5;
      recBuffers[writeRecBuffer].dataQ[writeToPlace] = inputBuffer[i + 1] -
    127.5;

      //zaslani signalu a posun na dalsi misto v pameti
      pthread_cond_signal(&(recBuffers[writeRecBuffer].condReadData));
      writeToPlace++;
      if (writeToPlace >= IQ_BUFFER_SIZE) { //end of I Q field
        writeToPlace = 0;
        recBuffers[writeRecBuffer].dataFull = true;
        recBuffers[writeRecBuffer].signalFull = true;

        while(recBuffers[writeRecBuffer].signalFull)
          pthread_cond_signal(&(recBuffers[writeRecBuffer].condReadData));

        writeRecBuffer = (writeRecBuffer + 1) % IQ_BUFFER_COUNT;
        pthread_mutex_lock(&(recBuffers[writeRecBuffer].mutMemReady));
      }

    }
    pthread_mutex_unlock(&receiverMemWrite);
  }
}

//funkce popisujici cinnost druheho vlakna, ktera zpracovava posunuta data
static void* readData(){
  float inputI, inputQ;
  int placeInBuffer = 0, readFromBuffer = 0;

  while (!end) {
    if (!recBuffers[readFromBuffer].dataFull)
      pthread_cond_wait(&recBuffers[readFromBuffer].condReadData, &recBuffers
    [readFromBuffer].mutReadData);
    else
      recBuffers[readFromBuffer].signalFull = false;

```

```
56      //Cteni informaci z globalni pameti
57      inputI = recBuffers[readFromBuffer].dataI[placeInBuffer];
58      inputQ = recBuffers[readFromBuffer].dataQ[placeInBuffer];
59
60      //Posun na dalsi misto v globalni pameti
61      placeInBuffer++;
62      if (placeInBuffer >= IQ_BUFFER_SIZE) {
63        placeInBuffer = 0;
64        pthread_mutex_unlock(&(recBuffers[readFromBuffer].mutMemReady));
65        recBuffers[readFromBuffer].dataFull = false;
66        readFromBuffer = (readFromBuffer + 1) % IQ_BUFFER_COUNT;
67      }
68
69      //nasledne zpracovani hodnot inputI, inputQ
70    }
71  }
```

Příklad 5.2: Možnost implementace programu pro posuv signálu a přenos jednotlivých vzorků signálu mezi vlákny

```
1  //inicializace platrofmy, ziskani ID platformy
2  cl_platform_id platform;
3  clGetPlatformIDs(1, &platform, NULL);
4
5  //ziskani ukazatele (ID) na zvolene OpenCL zarizeni
6  cl_device_id device;
7  clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
8
9  //vytvoreni kontextu, ve kterem je jedno zarizeni
10 cl_context context;
11 context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
12
13 //po precteni programu ze soubouru (do promenne source_str), dojde k jeho
     nacteni a prelozeni
14 cl_program program;
15 program = clCreateProgramWithSource(gpuControlData->context, 1, (const char
     **)&source_str, (const size_t *)&source_size, &err);
16 clBuildProgram(program, 1, &device, NULL, NULL, NULL);
17
18 //Inicializace ukazatele na zacatek funkce v prelozenem programu
19 cl_kernel kernel;
20 kernel = clCreateKernel(program, "correlationMultiplication", &err);
21
22 //Inicializace a vytvoreni fronty uloh, ktere sem maji provest
23 cl_command_queue cmdQueue;
24 cmdQueue = clCreateCommandQueue(context, device, 0, &err);
25
26 //nasleduje inicializace bufferu a udalosti
27 cl_mem inBuffer;
28 inBuffer = clCreateBuffer(context, CL_MEM_READ_ONLY, sizeof(short) *
     inOutDataSize, NULL, &err);
29 cl_event *eventPrepareData;
30 eventPrepareData = clCreateUserEvent(context, &err);
```

Příklad 5.3: Možnost inicializace openCL zařízení

```
__kernel void correlationMultiplication(__global short *inBuffer, __global
    int *outBuffer, __constant short *mask, int maskLen)
{
  int result = 0;
  int bufferLen = get_global_size(0);
  uint startPlace = get_global_id(0);
  uint sequence = get_global_id(1);

  int compareSequence = sequence * 1023;

  for (int i = 0; i < 1023; i++) {
    int compareBit = startPlace + i * maskLen;
    int maskBit = mask[compareSequence + i];
    for (int j = 0; j < maskLen; j++) {
      result += inBuffer[compareBit + j] * maskBit;
    }
  }
  outBuffer[startPlace] = result;
}
```

Příklad 5.4: Implementace funkce provádějící korelaci na GPU

```
__kernel __attribute__((vec_type_hint(short)))
void correlationMultiplication(__global short *inBuffer, __global int *
    outBuffer, __constant short *mask, int maskLen, int maskCount)
{
  short result = 0;
  uint bufferLen = get_global_size(0);
  uint maskBit = get_global_id(1);

  inBuffer +=  maskBit * maskLen + get_global_id(0);
  outBuffer += get_global_id(0);
  mask += maskBit;

  for (int i = 0; i < maskLen; i++) {
    result += inBuffer[i];
  }

  for (int i = 0; i < maskCount; i++) {
    if (maskBit == 0)
      outBuffer[i*bufferLen] = 0;

    outBuffer[i*bufferLen] += result * mask[i*1023];
  }
}
```

Příklad 5.5: Optimalizovaná implementace funkce provádějící korelaci na GPU