Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Prostředí pro testování zařízení na platformě iOS

Plzeň 2012 Martin Rott

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. června 2012

Martin Rott

Poděkování

Rád bych touto cestou poděkoval vedoucímu své bakalářské práce panu Ing. Ladislavu Pešičkovi za vstřícný přístup, odbornou konzultaci, metodické vedení a návrhy, které se staly podkladem pro vypracování této práce.

Abstract

Anotace

Tato bakalářská práce se zabývá možnostmi testování mobilních zařízení platformy iOS, verze systému iPhoneOS 5.0. Potřebná teorie k použitým technologiím se nachází v teoretické části. Dozvíte se jakým způsobem lze testovat výkon jednotlivých komponent systému, jako jsou rychlost lokálního souborového systému, rychlost přenosu dat mezi serverem a zařízením, procesoru a grafického čipu, jak zpracovat a interpretovat výsledky testů a jejich význam.

Abstract

This paper is focused on possibilities of benchmarking devices based on iOS platform, namely running iPhoneOS 5.0. The theory for implemented technologies is located in the theoretical part. You will learn how to test the performance of individual system components, like processor, graphics accelerator, local file system and remote file system.

Obsah

1 Úvod

S rozmachem mobilních zařízení se zvětšují nároky na jejich výkon. Stále se zvětšuje rozlišení obrazovky a s ním zároveň požadavky na výkon grafického čipu. V současné době je dostupné omezené množství nástrojů poskytující porovnání výkonu mobilních zařízení na platformě iOS.

Smyslem práce je zjistit a formulovat možnosti testování výkonu jednotlivých komponent mobilního zařízení platformy iOS (iPhone), jako jsou souborový systém, procesor, grafika a sit'ové přenosy. V případě sít'ových přenosů je třeba změřit reálný výkon při přístupu k různým sít'ovým zdrojům. Hardwarová konfigurace je daná, ale okamžitou výkonnost může ovlivnit několik faktorů jako jsou zatížení zařízení v dané chvíli (multitasking dalších aplikací), množství volné operační paměti, stav zdroje napájení. Dále navržení testovacího scénáře těchto komponent, programová realizace a testů následná analýza změřených hodnot korespondujících měření.

2 Teoretická část

2.1 Testování lokálního přístupu k datům

Souborový systém je důležitou součástí každého operačního systému, tedy i mobilního operačního systému. Jeho struktura hraje důležitou roli v organizaci a správě uložených dat, organizace samotná napomáhá systému a uživateli s přístupem k datům. Ten může být realizován asynchronně, a nebo synchronně. Asynchronní přístup k datům je preferován spíše u dat umístěných na vzdáleném úložišti, avšak použít se dá i u lokálního úložiště.

Souborový systém využívaných na všech zařízeních produkovaných firmou Apple se nazývá HFS+ (Hierarchical File System). Je využíván ke správě úložišť již od roku 1998, je žurnálovací a svazky jsou rozděleny do sektorů (logických bloků) o velikosti 512 bajtů. [7]

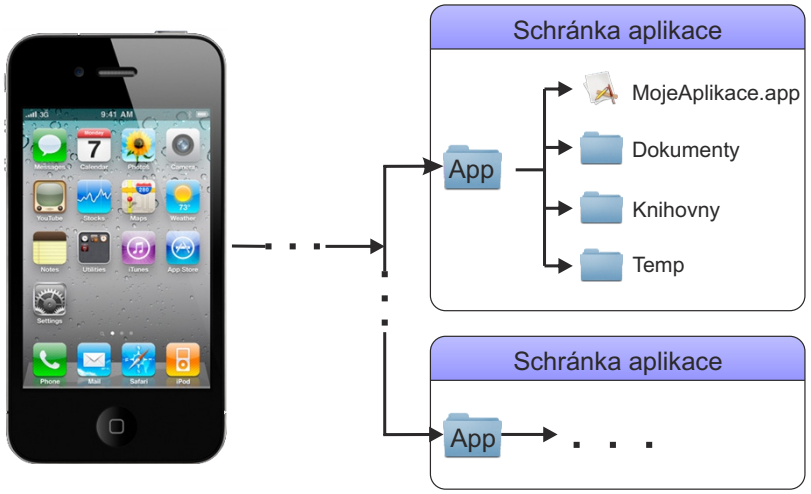
2.1.1 Souborový systém iOS

Souborový systém v iOS, zvládá perzistentní uložení datových souborů, aplikací a souborů spojených s operačním systémem, je založený na unixovém souborovém systému. Všechna datová úložiště připojená k zařízení, ať už jsou fyzicky připojeny nebo pomocí síťového spojení, skládají úložný prostor do jednotné kolekce souborů. Počet souborů může dosahovat několika milionů, proto souborový systém používá složky k vytvoření hierarchie souborů. Základ souborové struktury iOS je podobný unixové, avšak jsou zde rozdíly v organizaci aplikací a uživatelských dat.

Pohled na aplikace

Interakce aplikací se soubory na platformě iOS jsou omezeny hlavně na složky uvnitř aplikační schránky (sandboxu). Při instalaci nové aplikace, instalační kód vytvoří domácí složku aplikace, umístí do ní aplikaci a vytvoří několik dalších klíčových složek. Tyto složky představují primární strukturu aplikace v souborovém systému. (Obrázek 2.1)

Vzhledem k tomu, že aplikace je umístěna ve schránce, tak má obecně za-



Obrázek 2.1: Každá aplikace iOS operuje uvnitř své schránky.

kázaný přístup k vytváření souborů ve složkách umístěných mimo její vlastní schránku. Jedinou vyjímkou je použití veřejného systémového rozhraní k přistupování uživatelským datům, jako jsou například kontakty nebo hudba. V těchto případech, systémové rozhraní spravuje veškeré operace spojené s modifikací příslušných souborů. [15]

Důležitá datová úložiště aplikace ve schránce:

1. <Domovská složka aplikace>/Název Aplikace.app - Balíček obsahující samotnou aplikaci, do této složky se nic více nezapisuje, při instalaci je podepsána k zabránění modifikace, případný zápis do ní znemožní další spuštění.

2. <Domovská složka aplikace>/Documents/ - Používá se k ukládání uživatelských datových souborů, které nemohou být znovu vytvořeny aplikací.

3. <Domovská složka aplikace>/Library/ - Složka nejvyšší úrovně pro soubory, které nejsou uživatelskými. Typicky se zde ukládají soubory do jedné nebo více podsložek, ale je zde možné i vytvoření vlastních složek pro zálohování souborů, které nejsou přímo dostupné uživateli. Obsah této složky je zálohován pomocí iTunes[1].

[1]iTunes je programem poskytovaným firmou Apple pro synchronizaci dat a aplikací mezi mobilním zařízením a počítačem.

4. <Domovská složka aplikace>/tmp/ - Používá se k ukládání dočasných souborů, které se nemusí uchovávat mezi jednotlivými spuštění aplikace. Aplikace samotná by měla dočasné soubory odstraňovat, pokud již nejsou třeba, avšak systém samotný složku vyčistí po vypnutí aplikace.

2.1.2 Čtení a zápis souboru

Čtení a zápis do souboru zahrnuje přenos sekvence bajtů mezi pamětí a datovým úložištěm. Je formou správy souborů nejnižší úrovně a také základem sofistikovanějších technik, i nejsložitější datové struktury musejí být v určitém okamžiku převedeny na sekvenci bajtů před uložením na disk. Podobně ta samá data musejí být z disku přečtena jako sekvence bajtů před použitím k rekonstrukci datové struktury, kterou reprezentují.

Existuje několik různých technologií pro čtení či zápis do souboru, většina z nich je podporována iOS. Všechny však v podstatě fungují na stejném principu s malými rozdíly. Některé technologie vyžadují čtení a zápis dat v sekvencích, jiné dovolují skoky a práci s částí souboru. Některé technologie automaticky poskytují asynchronní čtení a zápis, kdežto jiné vykonávají operace synchronně, což poskytuje větší kontrolu nad jejich vykonáváním. Výběr z dostupných technologií, je tedy volbou mezi úrovní kontroly nad procesem zápisu či čtení a množstvím práce, které je třeba strávit nad psaním kódu správy zápisu či čtení souboru. Vysokoúrovňové technologie jako streamy omezují flexibilitu, avšak poskytují jednoduše použitelné rozhraní.

Asynchronní čtení a zápis

Provádění operací čtení a zápisu na disk bývá preferováno asynchronně. Technologie jako streamy Cocoa[2] je navržena tak aby byla operace prováděna vždy asynchronně. Tato skutečnost však neznamená, že ostatní technologie nemohou být používány asynchronně, avšak neposkytují takovéto chování automaticky. Vývojáři je tímto způsobem umožněno zaměření na čtení a zápis dat, bez nutnosti správy spouštění daného kódu implementující čtení a zápis.

[2]Cocoa je nativní objektově orientované API poskytované firmou Apple.

Zpracování souboru pomocí streamů

Pro čtení a zápis celých souborů (od začátku do konce), poskytují streamy jednoduché rozhraní pro asynchronní provedení těchto operací. Typické použití streamů je správa soketů a dalších zdrojů dat, které se stanou dostupnými s postupem času, avšak streamy se dají použít i pro čtení či zápis celého souboru v jedné nebo více iteracích. Platforma iOS poskytuje dva typy streamů: **NSOutputStream** pro sekvenční zápis na disk a **NSInputStream** pro sekvenční čtení z disku.

Objekty streamu využívají smyčku běhu vlákna k naplánování operace čtení a zápisu na streamu. Vstupní stream probudí smyčku běhu a informuje jejího delegáta[3], když jsou dostupná data ke čtení. Výstupní stream probudí smyčku běhu a informuje delegáta, když je dostupné místo pro zápis dat. Při zpracovávání souboru, toto chování obvykle znamená, že běhová smyčka je vzbuzena několikrát v krátkých intervalech tak aby kód delegáta mohl číst nebo zapisovat data. Což také znamená, že kód delegáta je volán opakovaně dokud není uzavřen objekt streamu nebo v případě vstupního streamu dokud stream nedosáhne konce souboru.

Synchronní čtení a zápis

Rozhraní související se soubory, operující nad daty synchronně umožňují vývojáři flexibilitu k nastavení vykonávání kontextu kódu. Synchronní výkon operací avšak neznamená, že jsou méně efektivní nežli asynchronní, pouze rozhraní neposkytuje automaticky asynchronní vykonávání. Pro využití těchto technologií k asynchronnímu čtení či zápisu dat, by bylo třeba poskytnout asynchronní kontext k vykonávání. V závislosti na tom, zda je požadován zápis či čtení, nebo oboje, jsou volány metody **NSFileHandle** (viz dále) ke čtení nebo zápisu bajtů. Po skončení je uvolněn popisovač a soubor uzavřen.

Čtení a zápis souborů s použitím NSFileHandle

Třída **NSFileHandle** je blízkou paralelou ke čtení a zápisu souborů na úrovni POSIX (Portable Operating System Interface, což je množinou stan-

[3]Delegát je objektem, který zastupuje nebo spolupracuje s jiným objektem v případě, že objekt je ovlivněn událostí vyvolanou v programu.(Uživatelská interakce apod.)

dartů specifikovaných k udržení kompatibility mezi operačními systémy). Základním postupem je otevření souboru, volání čtení či zápisů a uzavření souboru po skončení. V případě **NSFileHandle** je soubor otevřen automaticky při vytvoření instance třídy. Soubor je reprezentován objektem deskriptoru.

2.1.3 Vlastní testování výkonu souborového systému

Z výše jmenovaných způsobů zápisu a čtení souborů byl pro zjištění výkonu operací synchronní přístup pro jednodušší správu operací s oddělením do samostatného vlákna. Samotné měření je realizováno uložením systémového času před výkonem operace a po provedení je získána rychlost rozdílem aktuálního systémového času a předchozího. K získání reprezentativních výsledků jsou operace prováděny několikanásobně a nad soubory různých velikostí. Veškeré zápisy do souborů probíhají v dočasné složce aplikace viz (Důležitá datová úložiště aplikace ve schránce).

2.2 Testování GPU

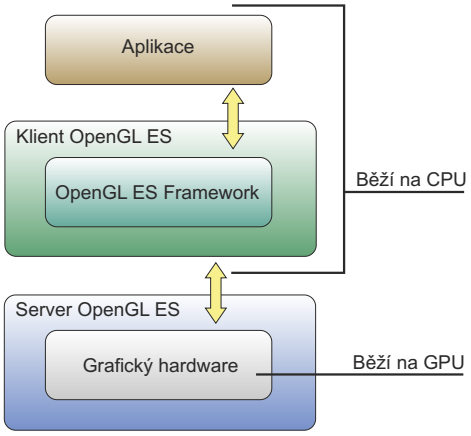
Pro testování grafického výkonu mobilního zařízení je třeba použít API využívající akceleraci grafickým procesorem. Na platformě iPhone jsou dostupné: CoreImage a OpenGL ES. CoreImage je navrhnuto ke zpracování obrázků k vytvoření vizuálních efektů za pomoci GPU. OpenGL ES podporuje vykreslování a výpočty pomocí programovatelných shaderů. Pro otestování grafického výkonu je tedy lepší využít technologie OpenGL ES, pro její univerzálnost a rozšířenost v oblasti mobilních zařízení.

2.2.1 OpenGL ES

Open Graphics Library (OpenGL) se používá pro vizualizaci 2D a 3D dat. Je víceúčelovou knihovnou, která podporuje aplikace s vytvářením digitálního obsahu, designu, her apod. OpenGL umožňuje vývojářům nastavení 3D pipeline[4] a následné zasílání dat. Vertexy[5] jsou transformovány, nasvíceny a sestaveny do primitiv, dále rasterizovány pro vytvoření 2D obrazu. OpenGL

[4]Pipeline určená k zasílání grafických primitiv a je podporována grafickým hardwarem.
[5]Vertex je datovou strukturou popisující bod v 2D nebo 3D prostoru.

je navrhnuto k překladu volání funkcí do grafických příkazů, které mohou být zaslány grafickému hardwaru. Ten je specializován na zpracování grafických příkazů, které v případě OpenGL probíhají velmi rychle. OpenGL ES je zjednodušenou verzí OpenGL, kde není zahrnuta veškerá funkcionalita z důvodu složité implementace pro mobilní hardware [9](Obrázek 2.2).



Obrázek 2.2: OpenGL ES.

Apple poskytuje implementace dvou verzí OpenGL ES 1.1 a 2.0.

2.2.2 Verze OpenGL ES

OpenGL ES 1.1 implementuje definovanou pipeline s předdefinovanými funkcemi. Poskytuje tradiční model nasvícení a shaderů, který umožňuje konfiguraci každé fáze pipeline k provedení specifických úkolů, nebo zakázání funkcionality, která není vyžadována.

OpenGL ES 2.0 má mnoho funkcí společných s OpenGL ES 1.1, avšak neobsahuje funkce jež se zaměřují na fáze předdefinovaných vertexových funkcí. Na místo nich, implementuje nové funkce, které poskytují přístup k víceúčelové pipeline na bázi shaderů. Ty umožňují psát vlastní vertexové a fragmentové funkce, které jsou spouštěny přímo na grafickém hardwaru. S shadery může aplikace více přizpůsobit vstup pipeline a výpočty prováděné na každém fragmentu vertexu.

OpenGL ES je platformě nezávislé API, založené na jazyce C, je přenositelné a široce podporované. Jakožto C API se hladce integruje do aplikací založených na Objective-C. Specifikace OpenGL ES nedefinuje zobrazovací

vrstvu okna, místo toho musí hostovací systém poskytnout funkce k vytvoření vykreslovacího kontextu OpenGL ES, který akceptuje příkazy a framebuffer, do kterého se zapisují výsledky vykreslovacích příkazů. [11]

2.2.3 Objekty OpenGL ES zahrnuté ve zdrojích aplikace

Objekty jsou průhlednými kontejnery, které aplikace používá k udržení stavů konfigurace, nebo dat potřebných k vykreslování. Protože jediný způsob přístupu k objektům je skrz procedurální API, při implementaci OpenGL ES si můžeme vybrat mezi různými strategiemi pro alokaci objektů v aplikaci. Můžeme tedy ukládat data ve formátu nebo umístění v paměti, které je nejvhodnější pro grafický procesor. Další výhodou objektů, je jejich znovupoužitelnost, která dovoluje aplikaci nastavení jedenkrát, avšak použití vícenásobné.

Nejdůležitější objekty OpenGL ES jsou tyto:

1. Textura (texture) - je obraz, který může být zpracován grafickou pipeline. Je používán k mapování barev obrazu na primitiva, ale může být také použit k mapování jiných dat, jako jsou třeba informace o nasvícení.

2. Buffer - je blokem paměti použitý OpenGL ES k ukládání dat aplikace. Buffery jsou používány k přesnému ovládání procesu kopírování dat mezi aplikací a OpenGL ES. Například, pokud je poskytnuto pole vertexů, OpenGL ES musí kopírovat data při každém poskytnutím dat a volání vykreslování. Avšak pokud aplikace ukládá svá data do vertexového bufferu, data jsou zkopírováno pouze pokud aplikace zašle příkaz ke změně obsahu vertexového bufferu. Používání bufferů může významně zvýšit rychlost aplikace.

3. Pole vertexů (vertex array) - je objektem udržující nastavení atributů vertexů, které budou čteny grafickou pipeline. Mnoho aplikací vyžaduje různé konfigurace pipeline pro každou entitu k vykreslení. Uložením konfigurace do vertexového pole, se vyhneme přenastavování pipeline a můžeme implementaci optimalizovat ke zpracování dané konfigurace vertexů.

4. Shadery - jsou také objekty. Aplikace využívající OpenGL ES 2.0 vytváří vertexové a fragmentové shadery ke specifikaci výpočtů, které budou provedeny jednotlivě na každém vertexu či fragmentu.

5. Renderbuffer - je jednoduchým dvourozměrným obrazem ve specifikovaném formátu, ten je obvykle definován jako barva, hloubka nebo vzorová data. Renderbuffer se obvykle nepoužívá odděleně, ale je používán s framebufferem.

6. Framebuffer - je kontejnerem, který na sebe navěšuje textury a renderbuffery, pro vytvoření konfigurace potřebné k vykreslování.

2.2.4 Vykreslování s OpenGL ES

Specifikace OpenGL ES vyžaduje, aby každá implementace poskytovala mechanismus k udržování vykreslovaného obrazu (vytvoření framebufferu).

Pro správné vytvoření objektu framebufferu, je třeba inicializovat tento objekt, dále vytvořit jeden nebo více cílů objektu (renderbuffery nebo textury), alokovat jejich úložiště a připojit je na framebuffer.

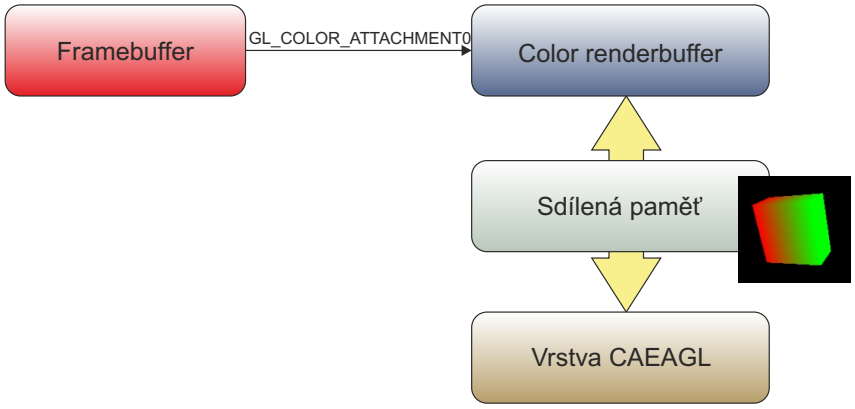
Většina aplikací, využívající OpenGL ES potřebuje vykreslit obsah framebufferu na obrazovku zařízení, tedy zobrazit uživateli. Na platformě iOS jsou všechny obrázky zobrazované na obrazovce obstarány vrstvou CoreAnimation[6]. Veškeré zobrazení je podloženo odpovídající vrstvou CoreAnimation. OpenGL ES je tedy připojeno k CoreAnimation skrz speciální vrstvu a tou je CAEAGLLayer[7]. Ta umožňuje renderbufferu chovat se jako obsah vrstvy CoreAnimation, čili umožňuje obsahu renderbufferu být transformován a vyskládán vedle dalšího obsahu vrstvy. Jakmile je složen výsledný obraz vrstvy CoreAnimation, je následně zobrazen na hlavní obrazovce zařízení. (Obrázek 2.3)

Ve většině případů, aplikace přímo nealokuje objekt vrstvy CAEAGL, místo toho aplikace definuje podtřídu **UIView**[8], která alokuje objekt vrstvy CAEAGL. Za běhu tedy aplikace vytvoří instanci náhledu a umístí ji do hie-

[6]CoreAnimation je kolekcí tříd pro grafické vykreslování projekce a animace.

[7]Objekt vrstvy CAEAGL je obálkou povrchu vrstvy CoreAnimation a je plně kompatibilní s funkcemi OpenGL ES.

[8]UIView definuje obdélníkovou plochu na obrazovce a rozhraní pro správu obsahu na této ploše.



Obrázek 2.3: Sdílení renderbufferu vrstvami CoreAnimation a OpenGL ES.

rarchie náhledů. Po vytvoření instance, aplikace inicializuje kontext OpenGL ES a vytvoří objekt framebufferu, který se připojí na vrstvu CoreAnimation.

Jen velmi zřídka se využívá OpenGL ES k renderování statických objektů, proto je nutné nastavit smyčku pro animace, kterou lze nastavit s objektem CADisplayLink, ten je objektem synchronizující vykreslování s obnovovací frekvencí displeje. Tato synchronizace umožňuje plynulé překreslování obrazovky.

2.2.5 Vlastní testování výkonu GPU

Je více způsobů jak měřit výkon GPU, čili rychlosti renderování objektů. Jedním z nich by bylo nastavení časovače na jednu sekundu a čítače renderovaných scén a libovolně-krát opakovat měření, tímto přístupem se dá získat počet renderovaných snímků za vteřinu, což je sice způsob velice využívaný, avšak takto získané výsledky si budou dosti podobné, protože hodnota získaná za jednu vteřinu bude sama o sobě již průměrnou. Vzhledem k této skutečnosti, jsem použil i jiný přístup a to využití čítače renderovaných snímků a ukládání doby renderování deseti snímků. Touto metodou je docíleno větší diverzity výsledků.

2.3 Testování CPU

Měření výkonu procesoru je prováděno vykonáváním souboru operací za účelem odhadnutí relativního výkonu, obecně vykonáním několika iterací měření. Takové měření umožňuje uživateli porovnání výkonu jednotlivých zařízení za stejných podmínek. Výkon CPU se dá porovnat pomocí frekvence procesoru a technických specifikací, v dnešní době existuje mnoho architektur s různými sadami instrukcí. Procesory s různými instrukčními sadami o stejné frekvenci dosahují různého výpočetního výkonu, proto porovnávání specifikací procesorů není dostačujícím ukazatelem. Dále moderní procesory nemají pevně nastavenou frekvenci na konstantní hodnotu z důvodu úspory energie, nebo prodloužení životnosti procesoru, proto je třeba procesor zatížit výpočty pro získání maximálního potenciálu. [4] [5]

2.3.1 Procesory v zařízeních iPhone

Apple oficiální cestou neposkytuje žádné informace o procesorech použitých v mobilních zařízeních, u iPhone, 3G a 3GS pouze konstatuje, že iPhone je uzavřenou platformou. Zařízení iPhone 4 podle oficiálního prohlášení firmy obsahuje procesor A4[9] avšak není uvedena jeho frekvence ani další parametry, podobně u zařízení iPhone 4S s tím, že obsahuje dvoujádrový procesor. Některé z technických specifikací byly zjištěny experimentálně na odblokovaných zařízeních.

2.3.2 Algoritmy vhodné pro testování rychlosti CPU

Matematické operace s celými čísly

Jsou základními operacemi, které využívá veškerý software a jsou tak i indikátorem rychlosti procesoru. Tyto algoritmy využívají velké soubory náhodných celých čísel (velikosti 32 nebo 64 bitů) a provádějí operací sčítání, odčítání, násobení a dělení.

[9]Procesor navržený firmou Apple a vyráběný firmou Samsung s instrukční sadou ARMv7

Komprese dat

Test komprese dat je založen na měření rychlosti komprese bloků dat do bloků o menší velikosti bez ztráty původních dat. [14]

Hledání prvočísel

Testuje jak rychle procesor vyhledává prvočísla, běží ve smyčce a zjišťuje rychlost porovnávání a operací (např. odmocnění) s čísly (o velikosti 32 bitů). [17]

Řešení lineárních rovnic

Využívá k ohodnocení výkonu procesoru matematický problém z oblasti lineární algebry, konkrétně násobení náhodné matice vektorem, nebo řešení lineární soustavy rovnic. [2]

2.3.3 Vlastní testování výkonu CPU

K testování výkonu procesoru bylo využito výpočtu lineárních rovnic. Test vyžaduje matici dimenze o velikosti 1000, a vektor kterým bude matice násobena. Nejdříve je třeba provést výpočet LU - faktorizace[10] matice a poté využít faktorizaci k řešení lineárního problému vektor krát matice. Výkon procesoru bude vyjádřen v MegaFLOPS (což je počet milionu operací za vteřinu v závislosti na frekvenci procesoru).

2.4 Testování vzdáleného přístupu k datům

Na platformě iOS je dostupných několik frameworků a knihoven pro síťovou komunikaci, vývojářům je tímto způsobem dostupná většina služeb ve Foundation a CoreFoundation frameworku, nebo za pomoci CFNetwork či BSD soketů. Při využívání těchto rozhraní, není třeba určovat síťové rozhraní na

[10]LU faktorizace je operací rozdělení matice původní na dvě a to dolní a horní trojúhelníkovou matici.

kterém bude probíhat komunikace, operační systém zařízení vybere rozhraní sám, případně bez problému komunikaci přesměruje na jiné rozhraní.

2.4.1 Foundation framework

Foundation framework zahrnuje třídy pro správu síťových služeb Bonjour. Tyto třídy korespondují se dvěma základními elementy síťových služeb a to službami a prohlížeči síťových služeb. Třída NSNetService reprezentuje instanci služby, poskytuje metody pro publikování služby do sítě a připojení k dané službě. Třída NSNetServiceBrowser funguje jako prohlížeč síťových služeb, publikuje dotazy na síti ke zjištění dostupných služeb a sbírá výsledky. [8]

NSNetService

Třída NSNetService reprezentuje jedinou instanci služby. Aplikace může publikovat službu lokální či vzdálenou, avšak jedna služba nemůže být zároveň vzdálená i lokální. Instance NSNetService provádějí veškeré operace asynchronně a navrací výsledky objektu delegáta ke zpracování.

Objekt NSNetService reprezentující vzdálenou službu je inicializován názvem služby, typem a doménou, nikoliv přímo názvem hostitele či IP adresou a číslem portu. Název služby, typ a doména jsou použity až k přeložení instance na IP adresu a číslo portu. Při publikování lokální služby do sítě je objekt NSNetService inicializován podobně a to názvem služby, typem služby, doménou a číslem portu služby.

NSNetServiceBrowser

Objekt NSNetServiceBrowser reprezentuje, buď prohlížeč pro jeden typ služby, a nebo prohlížeč domén. Podobně jako NSNetService, instance NSNetServiceBrowser provádějí veškeré operace asynchronně a předávají výsledky objektu delegáta ke zpracování. V jakémkoliv okamžiku může jediný objekt NSNetServiceBrowser vykonat pouze jednu operaci vyhledávání, pro vyhledávání více služeb najednou je potřeba využít více objektů NSNetServiceBrowser.

Ve většině případů je třeba vyhledat na síti specifický typ služby, napří-

klad se dají vyhledávat dostupné FTP servery na místní síti a následně zobrazit výsledky uživateli a tím mu umožnit výběr na který se aplikace připojí. Výsledky takového vyhledávání jsou instance NSNetService korespondující se službami. Další použití NSNetServiceBrowser je k vyhledání dostupných domén. Při zaslání prázdného řetězce názvu domény objektu NSNetServiceBrowser je vyvoláno vyhledávání všech dostupných domén. Při získání takového seznamu, je rozumnější nechat uživatele vybrat doménu k prohledání dostupných služeb, než je systematicky prohledávat seznam dostupných služeb.

2.4.2 Core Foundation Framework

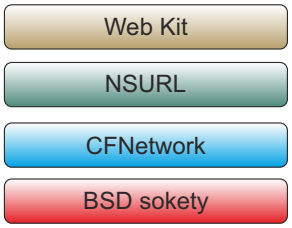
Je knihovnou s rozhraními odvozených z Foundation Frameworku, ale implementovaných v programovacím jazyce C, proto Core Foundation poskytuje jen omezený objektový model. Umožňuje sdílení kódu a dat mezi frameworky a knihovnami, umožňuje určitý stupeň nezávislosti na operačním systému a podporuje lokalizaci. V kontextu síťové komunikace umožňuje vytvářet, mazat a aktualizovat zdroje z URL. [16]

2.4.3 CFNetwork

CFNetwork je nízkoúrovňovým výkonným frameworkem, který poskytuje kontrolu na zásobníkem protokolů. Je rozšířením BSD soketů a standartním soketovým API, které poskytuje objekty ke zjednodušení úkonů, jako je komunikace s FTP nebo HTTP servery, či překlad názvu hostů. Stejným způsobem jako CFNetwork využívá BSD sokety, tak mnoho tříd z frameworku Cocoa využíva CFNetwork (například NSURL). Stejně jako WebKit je sadou tříd Cocoa k zobrazení webového obsahu v okně. Obě tyto třídy jsou vysokoúrovňové a implementují většinu síťových protokolů. Struktura softwarových vrstev znázorněna na obrázku 2.4.

CFNetwork má mnoho výhod oproti BSD soketům, poskytuje integraci do běhových smyček, aplikace tedy může využívat síťové smyčky bez implementace vláken. Také obsahuje objekty, které napomáhají k použití síťových protokolů bez nutnosti jejich implementace. Například lze využívat protokoly FTP bez nutnosti implementace CFFTP API.

CFNetwork má mnoho výhod i v porovnání s síťovým API Foundation



Obrázek 2.4: CFNetwork a softwarové vrstvy na platformě iOS.

frameworku. Je více zaměřen na síťové protokoly, kdežto síťová API Foundation Frameworku jsou více zaměřena na přístupu k datům, jako je třeba přenos za pomoci protokolů HTTP nebo FTP, Foundation API sice poskytuje jistou úroveň konfigurovatelnosti, ale CFNetwork poskytuje mnohem více.

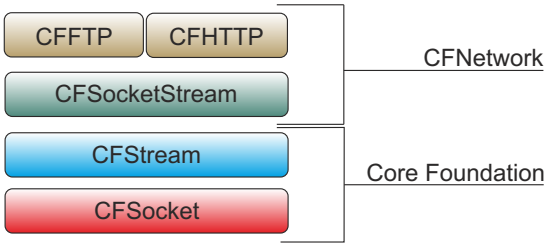
CFStream API

Je součástí CFNetwork API, streamy poskytují jednoduchý způsob pro výměnu dat a nejsou závislé na zařízení. Streamy se dají použít pro data uložené v paměti, souboru, nebo na síti a také se dají používat bez jednorázového načtení dat do paměti.

Stream je sekvencí bajtů sériově přenášených komunikační cestou. Stream je jednosměrnou cestou, proto při obousměrné komunikaci, je třeba vytvořit streamy dva a to vstupní a výstupní. Vyjímkou jsou souborové streamy, v nich se nedá vyhledávat, jakmile byla jednou data poskytnuta streamem a zpracována nelze k nim znovu přistupovat.

CFSteam je API poskytující abstrakci streamům zahrnující dva typy objektů a to: CFReadStream a CFWriteStream, oba typy splňují konvence Core Foundation. CFStream je postaven na třídě CFSocket a je základem CFFTP. (Obrázek 2.5)

Vstupní a výstupní streamy se používají podobně jako popisovače unixových souborů (stdin, stdout, stderr). Nejdříve je vytvořena instance streamu specifikováním zdroje či cíle streamu (paměť, soubor nebo soket). Dále je stream otevřen a čte nebo se do něj zapisuje v libovolném počtu iterací. Po dobu kterou stream existuje lze získávat informace o jeho vlastnostech. Jejich součástí je informace o zdroji a cíli, ale ty se přímo nevztahují k datům které čte či zapisuje. Pokud stream není již potřebný, je uzavřen a uvolněn z paměti.



Obrázek 2.5: Struktura CFStream API

Funkce CFStreamu vykonávající čtení či zápis jsou blokující, to znamená že pozastaví proces dokud nebyl přečten nebo zapsán alespoň jeden bajt dat. Aby bylo zamezeno blokování procesu jsou funkce obsluhující streamy naplánovány v běhové smyčce, poté je volána funkce zpětného volání, pouze pokud je možné nebo číst data bez toho aniž by byl blokován proces.

CFFTP API

CFFTP API zjednodušuje komunikaci s FTP serverem, umožňuje vytvoření FTP streamů pro stahování i upload dat a s jejich pomocí se dají provést následující operace: Stažení souboru z FTP serveru, nahrání souboru na FTP server, stažení seznamu souborů na FTP serveru a vytváření složek na FTP serveru. FTP streamy fungují stejným způsobem jako CFStreamy, s rozdílem že FTP stream je vytvořen s FTP URL, dále mají nastavitelné vlastnosti jako jsou uživatelské jméno a heslo k serveru.

2.4.4 Vlastní testování vzdáleného přístupu k datům

Při testování vzdáleného přístupu k datům, bylo použito **NSURLConnection** z Foundation frameworku , pro jeho jednodušší použitelnost a a CFFTP API z CFNetwork pro nízkoúrovňový přístup. Měří se rychlost přenosů s využitím protokolu FTP, který je ve CFNetwork implementován. Při přenosech se měří jak celková délka přenosu, i aktuální rychlost jednotlivých přenosů, tímto způsobem je získáno více hodnot ke zpracování.

3 Implementace

Nedílnou součástí bakalářské práce je program, realizující testy vyjmenované v zadání. Je implementován v jazyku Objective-C a vyvinut za pomoci nástroje Xcode, který je poskytován firmou Apple. Cílem aplikace je za pomocí technologií v předchozí kapitole vytvořit funkční celek k testování zařízení na platformě iOS.

3.1 Prerekvizity

3.1.1 Programovací jazyk Objective-C

Objective-C je navrženo jako rozšíření jazyku ANSI C, navíc je založeno na prvním objektově orientovaném programovacím jazyku Smalltalk, je navrženo tak aby poskytovalo plně objektový model v jednoduché podobě. V dnešní době je využíván na dvou platformách a to Mac OSX a iOS, původně však byl Objective-C jazykem primárně používaným v operačním systému NeXTSTEP. Programy napsané v jazyce Objective-C nevyužívající platformně závislé knihovny jako je například Cocoa, mohou být zkompilovány v jakémkoliv operačním systému podporovaným překladačem GCC [1].

Jazyk Objective-C od verze 2.0 podporuje ARC (Automatic Reference Counting). ARC implementuje automatickou správu paměti objektů a bloků jazyka Objective-C.

3.1.2 Xcode

XCode je IDE poskytované firmou Apple, je kompletní sadou nástrojů k vývoji programů na platformách Mac OS X a iOS, obsahuje editor zdrojového kódu, editor uživatelského rozhraní a mnoho dalších funkcí ke správě zdrojového kódu. Xcode také podporuje kontrolu syntaxe a částečně sémantiky programovacího jazyka Objective-C a navrhuje opravy. [13]

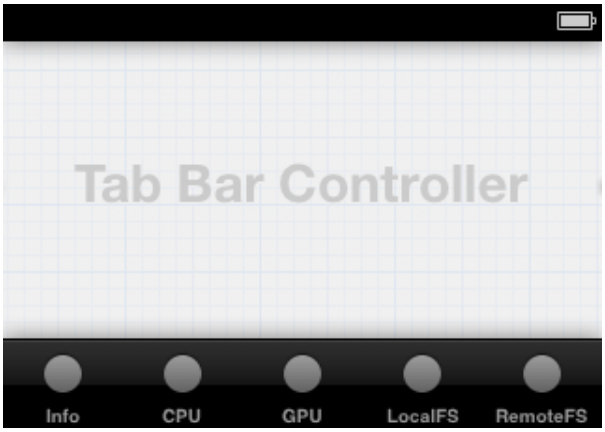
V IDE XCode je zahrnut kompilátor LLVM (Low Level Virtual Machine)

[1] GCC - Multiplatformní kompilátor programovacích jazyků.

verze 2.0 s plnou podporou jazyků Objective-C, C a C++. Toto vývojové prostředí lze využívat pouze na operačním systému Mac OSX.

3.2 Vlastní rozhraní aplikace

Uživatelské rozhraní na mobilních zařízení je limitováno několika faktory, a to velikostí (rozlišením) obrazovky, velikostí jednotlivých ovládacích či zobrazovacích prvků a dotykovým rozhraním. Aplikace se skládá z množství obrazovek obsluhovaných správci zobrazení. Hlavním je **UITabBarController**, ten pomocí záložek umístěních spodku obrazovky, umožňuje přepínání zobrazení vnořených. Záložky samotné jsou pojmenovány podle testu, který bude vykonáván na dané obrazovce. (Obrázek 3.1) [12]



Obrázek 3.1: TabBarController

Každá záložka je obsluhována správcem zobrazení **UINavigationController**, což umožňuje listování mezi jednotlivými obrazovkami (výsledky v tabulce a graf) každého testu.

3.2.1 Informace o zařízení

Pro zjištění systémových a uživatelských informací je implementována třída DevInfo. Systémové informace jsou zjišťovány pomocí třídy **UIDevice**, která je součástí frameworku UIKit. Důležité metody jsou: getValArray a getKeyArray. První vrací hodnoty a druhá klíče, získané z objektu **UIDevice**, ty

jsou využity k zobrazení informací v tabulce obsluhované **InfoTableView-Controllerem**, který je oddděn od **UITableViewControlleru** (Obrázek 3.2).[1]

Obrázek 3.2: Informace o zařízení

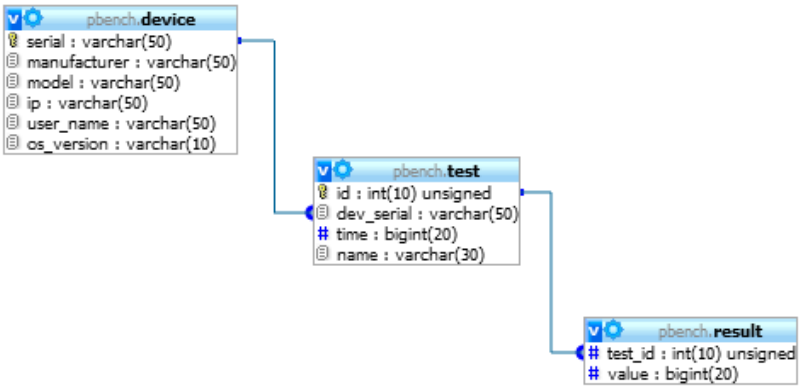
Tlačítkem DB se přechází na další obrazovku obsluhující upload výsledků testů do databáze MYSQL.

3.2.2 Upload výsledků do databáze měření

Výsledky jsou nahrávány do databázového systému MYSQL umístěného na adrese `stevie.heliohost.org/pbench` . ER-model na obrázku(3.3). Databáze je sdílena s aplikací testující výkon zařízení platformy Android.

Backend

Pro jednotlivé transakce mezi databází a zařízením je využito open source knihovny **mysql_connector**. Veškeré transakce jsou obsluhovány třídou Mysql, ta využívá třídy DevInfo k načtení informací o zařízení a následně k jeho identifikaci v databázi měření a třídy ve kterých jsou ukládány výsledky testů (LFSResults, RFSResults, CPUResults, GPUResults).



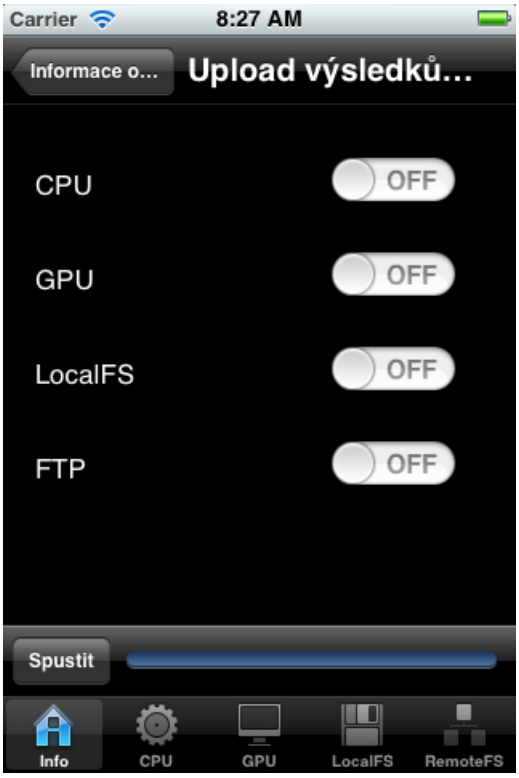
Obrázek 3.3: Struktura databáze

Samotné připojení je inicializováno metodou connectToDatabase, jejíž návratovou hodnotou (BOOL) je zda se připojení zdařilo, vyjímky jsou ošetřeny pomocí bloku try - catch.

Pokud bylo spojení úspěšně navázáno, je spuštěn upload výsledků, obsluhovaný metodami uploadResults a uploadTestResults, první metoda zjišťuje zda jsou dostupné výsledky testů (zda test proběhnul) a druhá je volána v případě, že proběhnul. Při spuštění nahrávání výsledků je nejdříve provedena kontrola, zda se zařízení nachází v databázi za pomocí příkazu SELECT serial FROM Device, pokud zařízení není nalezeno, je vloženo do databáze za pomocí příkazu INSERT a informací získaných ze třídy DevInfo. Jednotlivé testy jsou vkládány do tabulky Test, i při této operaci je prováděna kontrola, zda test není již přítomný v databázi pokud ano, přečte se id z databáze a je použito ke vkládání hodnot testu, v případě že test není v databázi přítomen je vložen a přečteno jeho id a následně použito ke vkládání výsledků testu.

Frontend

Obrazovka zařízení je obsluhována třídou **DBViewController**, tlačítko Spustit zahajuje připojení na databázi a průběh uploadu je zobrazován pomoci **UIProgressBar**, který je aktualizován podle průběhu uploadu. Dále jsou na obrazovce čtyři přepínače **UISwitch**, každý koresponduje s typem testu, který je možné na zařízení spustit a je aktualizován podle skutečnosti, zda test již proběhnul. Tlačítko Informace o zařízení umožňuje přechod zpět na obrazovku s informacemi o zařízení a je obsluhováno správcem zobrazení **UINavigationController** (Obrázek 3.4).



Obrázek 3.4: Upload výsledků do databáze

3.2.3 Lokální přístup k datům

Při spuštění testu je dotázán delegát aplikace zda test již běží pokud ne je vytvořeno nové vlákno a spuštěn test, notifikován delegát o průběhu a po skončení běhu vlákno ukončeno, následně notifikován delegát o ukončení běhu

testu, výsledky testu jsou zpracovávány třídou LFSResults. Veškeré operace se soubory jsou prováděny v dočasné složce /<Aplikace>/tmp. Po skončení testu je vlákno ukončeno pomocí příkazu NSThread Exit.

Backend

Test lokálního souborového systému je vykonáván třídou LFSBenchmark a skládá se ze tří částí a to generování, mazání a kopírování souborů. Generování a mazání je vykonáváno metodou genFile vstupními parametry je velikost souboru v kilobajtech a počet iterací. Soubory jsou vytvářeny pomocí třídy **NSFileManager** a následně je do nich cyklicky prováděn zápis, za pomoci **NSFileHandle** [6], dokud soubor nedosáhne požadované velikosti. Před zahájením operace generování je uložen aktuální systémový čas a po skončení operace je doba běhu zjištěna pomocí rozdílu aktuálního času a předem uloženého. Součástí této metody je i mazání souborů, měření je prováděno stejným způsobem. Testovány jsou soubory o velikosti 1KB, 1MB a 10MB. Návratová hodnota této metody v případě úspěšného běhu je prázdná, v opačném případě je vrácena chyba a notifikován delegát aplikace o neúspěšném průběhu testu.

Kopírování souborů je obstaráno metodou copyFile vstupními parametry jsou počet iterací kopírování a velikost souboru, soubory jsou kopírovány pomocí **NSFileManager**. Opět jsou testovány soubory o velikostech 1KB, 1MB a 10MB.

Frontend

Obrazovka spravující test lokálního souborového systému se skládá z několika prvků a to okna (UITableView) ve kterém bude zobrazována tabulka s výsledky měření ty jsou barevně odlišeny podle typu operace, tlačítka (UIBarButton), stepperu omezeného na interval od tří do deseti určující počet iterací testů a tlačítka Graf (UIBarButton) umožňující přechod na obrazovku s grafem hodnot. Po spuštění testu je schováno tlačítko spustit a místo něj je zobrazen indikátor průběhu testů. Po skončení testu je indikátor schován a opět zobrazeno tlačítko Spustit. (Obrázek 3.5)

Obrázek 3.5: Test lokálního přístupu k datům

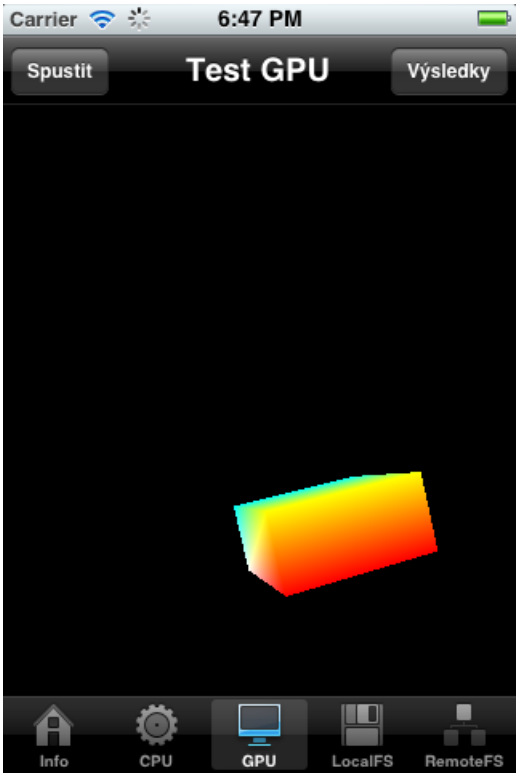
3.2.4 GPU

Podobně jako u testu procesoru je při spuštění testu vytvořeno nové vlákno, notifikován delegát a po úspěšném běhu vlákno ukončeno. Měření je realizováno vždy na skupině deseti vykreslených snímků. Výsledky testu zpracovává třída gpuResults.

Backend

Po stisknutí tlačítka Spustit je vytvořeno okno do kterého se bude vykreslovat testovací obraz, to je obsluhováno třídou OpenGLView, ta při inicializaci nastaví a propojí vrstvu okna a CAEGLLayer, dále ji nastaví jako průhlednou, pak je nastavena verze vykreslovacího API (EAGLRenderingAPI) na OpenGLES 2.0 a následně inicializovány a propojeny renderbuffer, framebuffer a depthbuffer s vykreslovacím API. Dalším krokem je nastavení Vertex a Fragment shaderů, jsou napsány v jazyku GLSL a kompilovány za běhu

aplikace. Kompilace je prováděna metodou (GLuint)compileShader, ta ověří zda existuje zdrojový soubor shaderu a následně ho zkompiluje, návratovým typem je shaderHandle. Pro vykreslení 3D objektů ve 2D zobrazení je třeba nastavit projekci objektu, ta je obstarána knihovnou Cocos3DMath. [3]



Obrázek 3.6: Test vykreslování grafického čipu

Frontend

Obrazovka testu (Obrázek 3.6) je obsluhována třídou GpuTestViewController oddděné od UIViewController. Při stisknutí tlačítka Spustit je dotázán delegát aplikace zda test již probíhá, v případě záporné odpovědi je test spuštěn. Při této operaci je vytvořeno vnořené zobrazení UIView propojené s OpenGLES a použité pro vykreslování. Po skončení jsou uvolněny objekty bufferů a je uvolněna instance zobrazení z operační paměti. Tlačítkem výsledky se přechází na obrazovku s tabulkou výsledků použité k zobrazení minimální, průměrné a maximální doby renderování jednoho rámce (frame). Z této obrazovky se přechází na obrazovku s grafem veškerých hodnot stisknutím tlačítka graf. Veškeré hodnoty jsou uváděny v jednotkách mikrosekund.

3.2.5 CPU

Při spuštění testu je vytvořeno nové vlákno, dotázán objekt delegáta zda test již probíhá v případě záporné odpovědi, je notifikován o započetí testu, tím je zamezeno vícenásobnému spuštění. Dále je zobrazen indikátor aktivity, který je viditelný na všech zobrazeních aplikace. Po skončení všech iterací testu procesoru je vlákno ukončeno pomocí příkazu NSThread Exit.

Backend

Výkon procesoru je měřen řešením systému lineárních rovnic, výsledkem je počet operací za vteřinu. Veškeré výpočetní operace použité k měření výkonu procesoru jsou vykonávány třídou CPUbenchmark. Hlavní metodou této třídy je runBenchmark s návratovým typem double výsledek. Při spuštění benchmarku je vygenerována matice o velikost 1000 a vektor řešení. Následně je matice faktorizována metodou factorizeMatrix (faktorizace je operací z oblasti lineární algebry kdy matice je rozdělena na dvě a to horní a dolní trojúhelníkovou). Před i po této operaci je přečten systémový čas, jeho rozdílem je získána délka operace ten je uložen do pole time. Dále je volána metoda solveLinearSystem, před provedením a po skončení této metody je uložen systémový čas do pole time, která počítá s faktorizovanou maticí a vektor pravých stran, ten je postupem výpočtu přepisován výsledky. Výsledná doba běhu testu procesoru je tedy rozdílem časů uložených před a po operacích faktorizace a řešení lineárního systému rovnic.

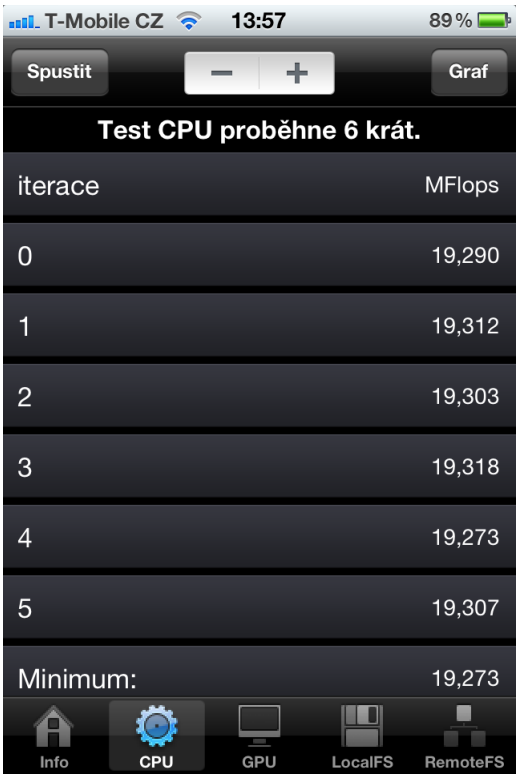
Výsledky testů procesoru jsou spravovány třídou CPUResults, ta provádí výpočet maximální, minimální, průměrné hodnoty a uchovává čas kdy test proběhnul, dále je využívána k předávání dat pro upload do databáze a vykreslování grafem.

Frontend

Úvodní obrazovka testu procesoru (Obrázek 3.7) je obsluhována třídou CPUTableController oddděné od UITableViewController, test je spouštěn tlačítkem Spustit, při stisknutí je nahrazeno progressbarem[2] ukazující postup testů. Uživatel si může zvolit kolikrát test proběhne a to pomocí stepperu[3].

[2]Progressbar je animovaných ukazatelem průběhu operace.

[3]Stepper je tlačítko umožňující inkrementaci a dekrementaci propojené proměnné.



Obrázek 3.7: Test procesoru

Minimálním počtem běhů jsou dva a to z důvodu vykreslování grafu, graf o jedné hodnotě by nemělo smysl vykreslovat a maximálním počtem běhů je deset z důvodu délky testu. Výsledky jsou zobrazeny v tabulce a aktualizovány průběžně s každou iterací testu, po skončení jsou zobrazeny maximální, minimální a průměrná hodnota. Tlačítkem Graf se přechází na obrazovku s grafickou reprezentací hodnot získaných jednotlivými běhy testu.

3.2.6 Vzdálený přístup k datům

Test vzdáleného přístupu k datům je realizován měřením rychlosti uploadu a downloadu souboru. Podobně jako u předchozích testů je při spuštění dotázán delegát aplikace na průběh a následně notifikován o započetí testu, po skončení operace jsou výsledky testu zpracovávány třídou RFSResults. Veškeré datové přenosy jsou naplánovány v běhové smyčce, tedy probíhají asynchronně a neblokují uživatelské rozhraní.

Backend

Operace přenosu směrem k FTP serveru jsou obsluhovány třídou RFSUpload a směrem opačným třídou RFSDownload. Přenášeny jsou soubory o velikosti 10MB, v případě uploadu jsou vygenerovány třídou LFSBenchmark. Pro upload u download jsou v příslušných třídách vytvořeny objekty souborového streamu a síťového streamu. Za pomoci delegáta aplikace je přeložena adresa serveru z řetězce do formátu URL. Dále jsou aktualizovány streamy a nastaveny přihlašovací informace na síťovém streamu. Následně je naplánován přenos v běhové smyčce. Smyčka reaguje na události na síťovém streamu pomocí metody stream:handleEvent.

Měření doby částí přenosu je prováděno po kvantech, protože při měření doby za kterou jsou získána aktuálně přijatá data dochází ke ztrátě přesnosti (jsou děleny přijaté bajty dobou za kterou byly přijaty, ta je získána systémovým voláním a je o dva až tři řády menší než jedna vteřina). Kvantum je rovno deseti iteracím zápisu dat. Dále jsou v této metodě získávány časy před začátkem a po skončení přenosu. Při zápisu do streamu souboru či do síťového streamu je využit buffer. Ten je v případě stahování souboru ze serveru plněn ze síťového streamu a následně jsou zapisována do souborového streamu, při nahrávání souboru na server je plněn daty ze souborového streamu a ta jsou pak zapsána do síťového streamu. Při výskytu chyby přenosu je notifikován delegát a ukončen přenos.

Frontend

Obrazovka testu je navržena stejně jako u testu lokálního souborového systému. (Obrázek 3.8)

Obrázek 3.8: Test vzdáleného přístupu k datům s využitím protokolu FTP

3.3 Reprezentace výsledků

Obecně výsledky všech testů zařízení jsou reprezentovány dvěma způsoby. Tabulka je pro každý test konstruována stejným způsobem načtením hodnot z příslušné třídy obsluhující výsledky a grafem. U všech tabulek jsou vypsány minimální, průměrná a maximální hodnota výsledků. Druhým způsobem je reprezentace vynesením naměřených hodnot do grafu, do grafu jsou vykreslovány hodnoty jednotlivých iterací testů.

3.3.1 Tabulky

Z důvodu jednotného formátování tabulek byla vytvořen vlastní prototyp řádky tabulky UITableCell s názvem CustomCell, skládající se z popisku (UILabel) využívaného k popisu hodnot a textového pole (UITextfield) využívaného k zobrazování hodnot. Pozadí řádky je vyplněno gradientem pro estetický vzhled. [10]

3.3.2 Grafy

Veškeré grafy jsou vykreslovány jediným správcem zobrazení pojmenovaným GraphController, ten je oděděn od UIViewController, data k zobrazení a jednotky jsou předávána pomocí delegáta aplikace. Při vykreslování více křivek je využito třídy Colors v níž jsou nadefinovány barvy použité k vykreslo-

vání. Tato třída je využita pro reprezentaci výsledků testu lokálního souborového systému a vzdáleného souborového systému, kde barvy jednotlivých nadpisů testů korespondují s barvami křivek grafu. Na ose x grafu jsou vždy vyneseny iterace a na ose y korespondující hodnoty iterací. (Obrázek 3.9)

Obrázek 3.9: Grafická reprezentace výsledků na zařízení.

3.4 Uživatelský manuál

3.4.1 Instalace

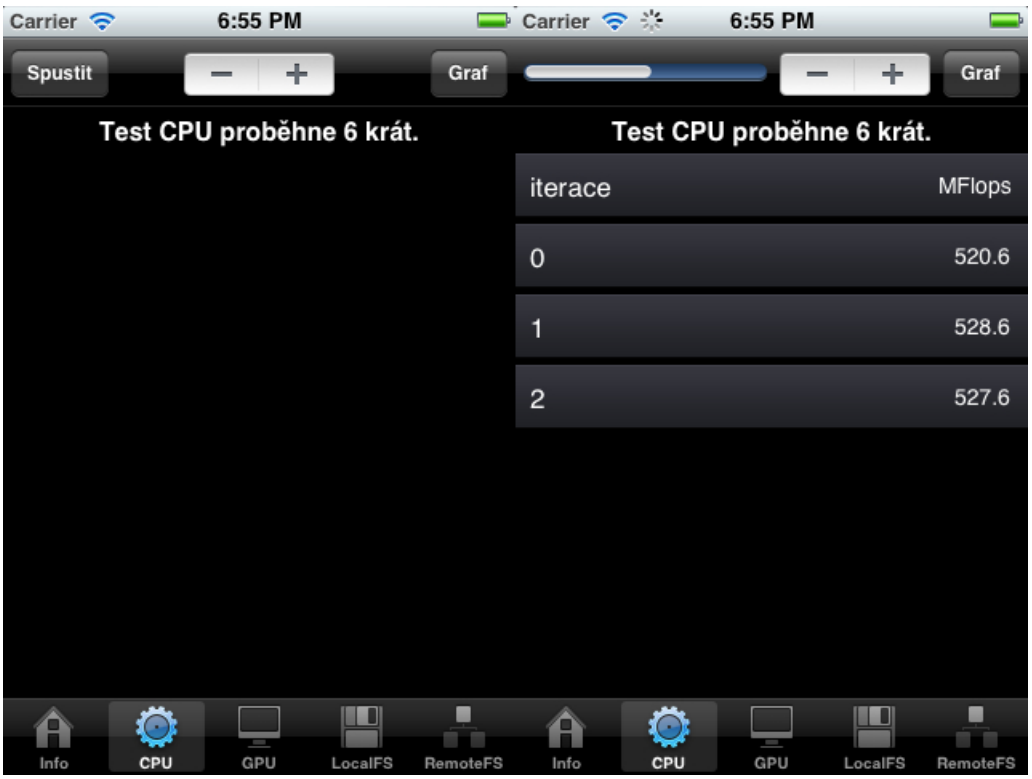
Aplikace může být obecně nainstalována dvěma způsoby. Prvním je získání aplikace přes distribuční aplikaci poskytovanou firmou Apple (AppStore), avšak vývojář pro uveřejnění aplikace musí aplikaci nechat schválit firmou Apple. Druhým způsobem ,využitým v bakalářské práci, je instalace na vlastní zařízení, ta je možná pouze za předpokladu, že vlastníte počítač s operačním systémem MacOSX, IDE Xcode a vývojářský účet u firmy Apple, vygenerujete profil a certifikát nutný k přeložení a instalaci aplikace do zařízení.

3.4.2 Ovládání aplikace

Aplikace se spouští stisknutím ikony (Obrázek 3.10) na displeji zařízení.

Obrázek 3.10: Ikona aplikace

Dále je zobrazena úvodní obrazovka s informacemi o zařízení. Přepínání mezi jednotlivými testy je prováděno stisknutím ikony testu na spodní části obrazovky. Listování mezi testem, tabulkou a grafem je realizováno stisknutím navigačních prvků na horní části obrazovky. Testy se spouští tlačítkem spustit a počet iterací testu je upravován stiskem tlačítek plus a mínus. Průběh testů je indikován postupem ukazatele průběhu v případech testu CPU, lokálního souborového systému (LocalFS) a vzdáleného souborového systému (RemoteFS). V případě testu GPU je při průběhu zobrazována kostka pohybující se na displeji. Informace o průběhu testů je také zobrazována ve liště ukazující status telefonu, z důvodu stále viditelnosti. (Obrázek 3.11)



Obrázek 3.11: Připravený a spuštěný test.

4 Analýza získaných dat

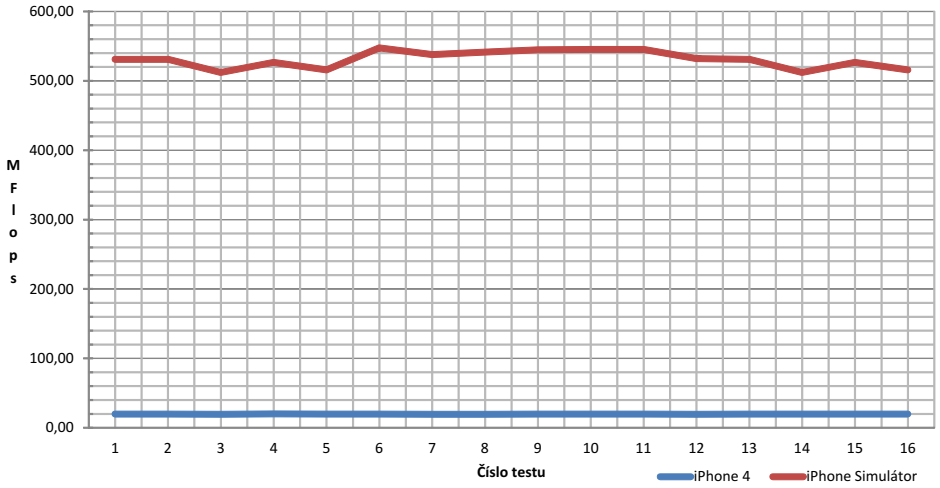
4.1 Test CPU

Naměřené hodnoty jsou uvedeny v tabulce 4.1, jednotkou naměřených hodnot je MFlops (počet milionů operací za jednu vteřinu). a vyneseny v grafu 4.1.

Tabulka 4.1: Měření rychlosti CPU

Měření provedeno na: [MFlops]	
iPhone 4	iPhone Simulátor
19,65	530,88
19,71	530,88
19,53	511,99
19,98	526,60
19,87	515,66
19,87	547,58
19,58	537,77
19,59	541,27
19,69	544,78
19,70	545,36
19,70	545,26
19,52	532,10
19,87	530,88
19,70	511,99
19,75	526,60
19,81	515,66

Rozdíl mezi naměřenými hodnotami na zařízení a simulátoru je způsoben různou hardwarovou konfigurací. Veškeré operace na simulátoru jsou vykonávány procesorem hostovacího systému, není omezena rychlost zprostředkovaných operací, kterým v tomto případě byl MacOS X 10.7 na zařízení MacBook s procesorem Core i5 o frekvenci 2.4 GHz. Rozdíly mezi jednotlivými měření na simulátoru (v řádech desítek MFlops) i na zařízení (v řádech desetiny MFlops) jsou způsobeny různým vytížením systému v době měření.



Obrázek 4.1: Graf měření rychlosti CPU.

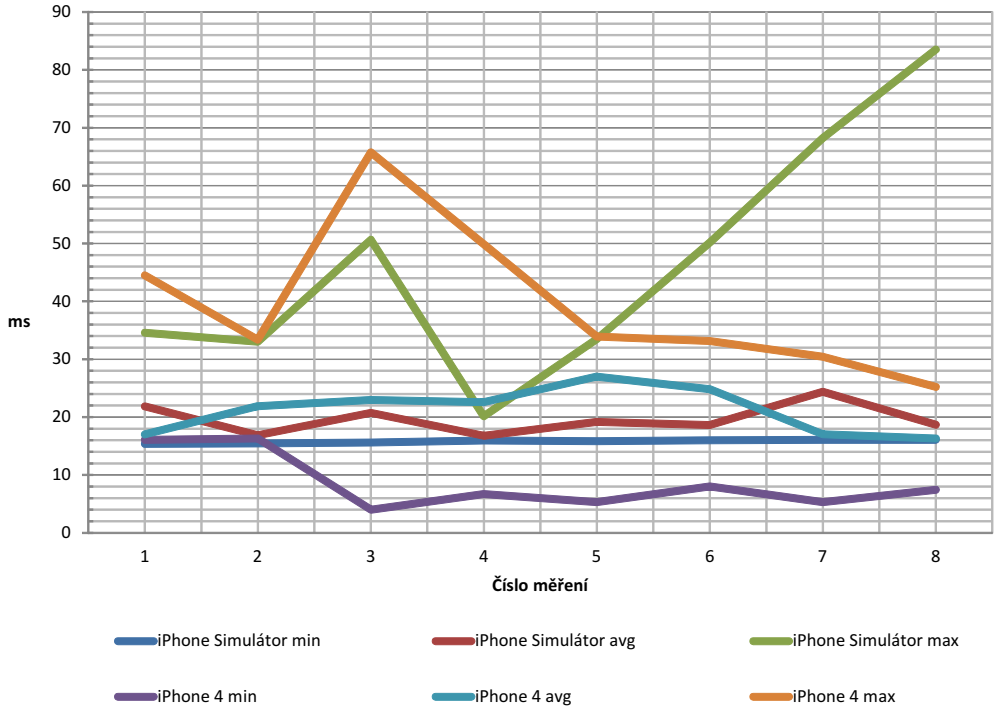
4.2 Test GPU

V tabulce 4.2 a grafu 4.2 jsou uvedeny hodnoty minimální, průměrné a maximální doby vykreslování snímku jednotlivých testů.

Tabulka 4.2: Měření doby renderování rámce

Měření provedeno na: [μs]					
iPhone Simulátor			iPhone 4		
min	avg	max	min	avg	max
15371	21824	34590	16084	17011	44503
15465	16829	33016	16319	21868	33366
15616	20734	50664	3990	22958	65742
15939	16781	20136	6700	22566	49864
15828	19183	33457	5293	26980	33912
16006	18616	50133	7997	24838	33161
16055	24341	68208	5288	17046	30445
16075	18648	83508	7419	16307	25197

Při přepočtu změřených hodnot na počet rámců vykreslených za vteřinu se veškeré naměřené hodnoty budou pohybovat za hranicí 15 FPS a to je hodnotou plně dostačující k plynulému zobrazení vykreslovaného obrazu. Hodnoty vykreslené v grafu byly převedeny na jednotky milisekund pro větší přehlednost.



Obrázek 4.2: Graf rychlosti vykreslování rámců.

Výkyvy naměřených hodnot jsou způsobeny tím, že operační systém iOS stejně jako MacOS X akceleruje vykreslování všech prvků uživatelského rozhraní pomocí grafického čipu zařízeni.

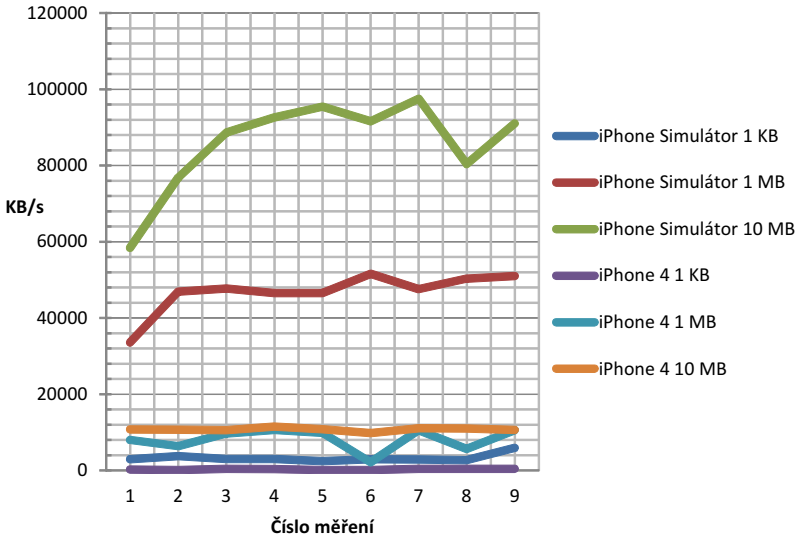
4.3 Test lokálního souborového systému

Hodnoty získané měřením rychlosti generování souborů jsou uvedeny v tabulce 4.3 a grafu 4.3, kopírování souborů v tabulce 4.4 a grafu 4.4 a mazání souborů v tabulce 4.5 a grafu 4.5, jednotkou rychlosti je KB/s.

Tabulka 4.3: Tabulka rychlosti vytváření souborů různých velkostí

Měření provedeno na: [KB/s]					
iPhone Simulátor			iPhone 4		
1 KB	1 MB	10 MB	1 KB	1 MB	10 MB
2907	33561	58373	239	7982	10743
3732	46925	76832	75	6319	10707
3003	47750	88570	421	9698	10589
3003	46507	92557	378	10593	11514
2445	46497	95407	106	9885	10834
2958	51571	91542	110	2118	9822
2874	47537	97489	443	10548	11073
2688	50302	80341	432	5641	11028
5881	50996	90983	420	10559	10647

Hodnoty naměřené při testování rychlosti generování souboru, v případě iPhone simulátoru přibližně odpovídají rychlosti operací zápisů na pevný disk počítače. Variace hodnot u simulátoru je způsobena zpožděním předávání instrukcí mezi simulátorem a hostujícím operačním systémem. Rychlosti zápisu na zařízení jsou podobny rychlostí zápisu na paměti typu flash.

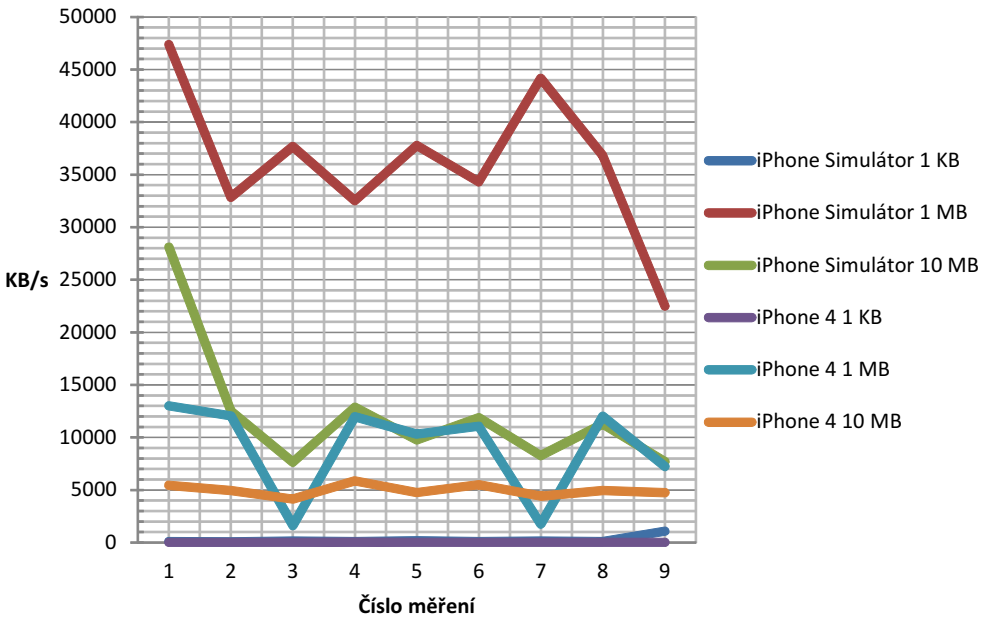


Obrázek 4.3: Graf rychlosti generování souborů různých velkostí.

Tabulka 4.4: Rychlost kopírování souborů různých velkostí

Měření provedeno na: [KB/s]					
iPhone Simulátor			iPhone 4		
1 KB	1 MB	10 MB	1 KB	1 MB	10 MB
82	47361	28109	19	13002	5449
63	32833	12511	19	12040	4958
136	37676	7628	21	1617	4115
87	32536	12859	24	11958	5865
176	37785	9762	22	10312	4755
92	34315	11892	22	11066	5492
151	44155	8252	22	1732	4414
90	36820	11258	19	12021	4949
1082	22478	7692	21	7229	4749

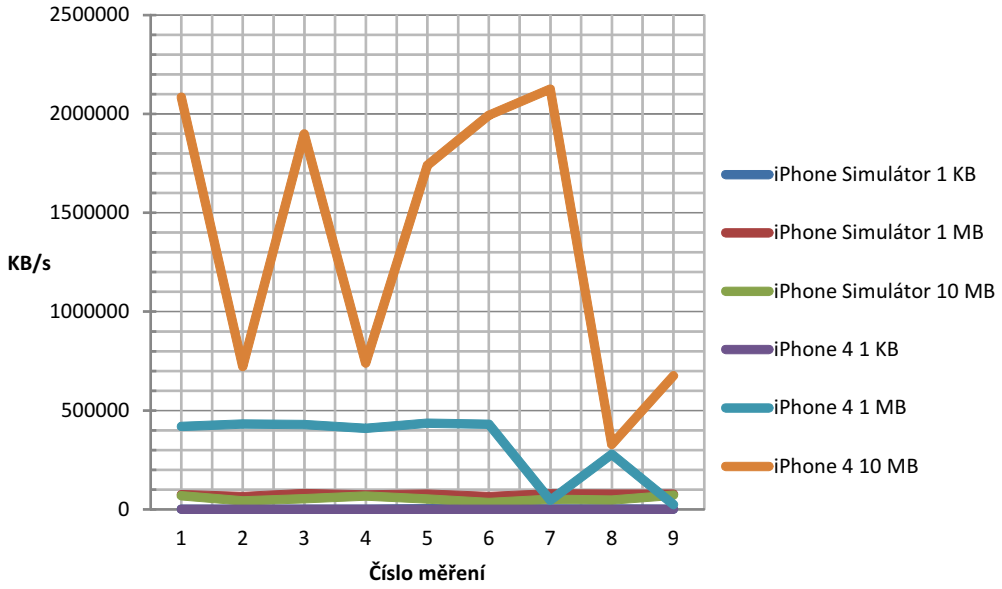
Z naměřených hodnot lze usoudit, že rychlost kopírování souborů v zařízení přibližně odpovídá rychlosti kopírování souborů na flash discích.



Obrázek 4.4: Graf rychlosti kopírování souborů různých velikostí.

Tabulka 4.5: Rychlost mazání souborů různých velkostí

Měření provedeno na: [KB/s]					
iPhone Simulátor			iPhone 4		
1 KB	1 MB	10 MB	1 KB	1 MB	10 MB
61	74478	69471	279	419154	2083848
124	64447	43628	282	431698	722147
26	79435	54017	376	428982	1897343
177	73908	67128	160	410089	739724
3677	77588	52570	201	435927	1738536
5376	64561	35007	173	430616	1992608
1701	79128	50752	199	46484	2124907
1961	77231	47074	489	278487	327219
899	77752	71405	20	24099	675684



Obrázek 4.5: Graf rychlosti mazání souborů různých velikostí.

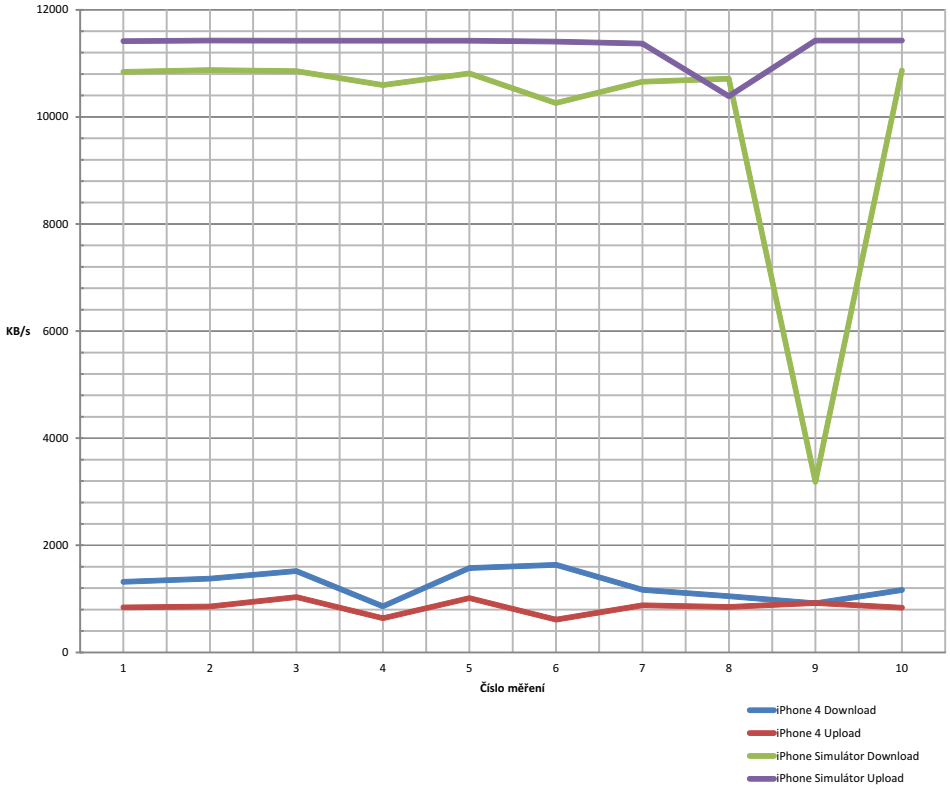
Naměřené rychlosti mazání souborů jsou dle očekávání vyšší než rychlosti zápisu (generování), nebo kopírování souborů. To je způsobeno tím, že operace mazání není fyzickým smazáním souboru z paměťového úložiště, ale pouze operací označení daného prostoru na úložiště za volné pro zápis.

4.4 Test vzdáleného souborového systému

Připojení do sítě internet v případě měření rychlosti přenosů při využití iPhone simulátoru bylo realizováno za pomoci kabelového připojení do univerzitní sítě (rychlost 100Mb/s). V případě testování rychlosti přenosů na zařízení iPhone bylo využito připojení do univerzitní sítě využitím WiFi sítě eduroam. Hodnoty získané měřením jsou uvedeny v tabulce 4.6 a vyneseny v grafu 4.6, jednotkou rychlosti je KB/s.

Tabulka 4.6: Rychlost přenosů za pomoci protokolu FTP

Měření provedeno na: [KB/s]			
iPhone 4		iPhone Simulátor	
Download	Upload	Download	Upload
1319	841	10841	11415
1378	858	10875	11427
1518	1032	10855	11423
860	636	10596	11422
1574	1012	10813	11422
1635	610	10257	11406
1167	879	10658	11369
1048	848	10714	10386
917	923	3186	11425
1166	834	10867	11426



Obrázek 4.6: Graf měření rychlosti přenosů.

4.5 Možná rozšíření

Sekce testování procesoru by mohla být rozšířena o další testy jako například test výpočtu čísla π v rozsahu N desetinných míst, či hledání prvočísel. Dalším možným rozšířením aplikace by mohlo být testování grafického čipu vykreslováním s využitím knihovny CoreGraphics a CoreImage (poskytované firmou Apple). Jedním z možných rozšíření testování lokálního souborového systému by bylo o operace s adresáři, například generování struktury adresářů o předepsané hloubce. Test vzdáleného souborového systému bylo možné implementovat pro další protokoly používané k přenosu dat například SFTP, SSH, Samba, nebo pro stále vyvíjený souborový systém KIVFS.

5 Závěr

Mobilní zařízení se v dnešní době využívají k celé řadě činností jako zpracování dokumentů a fotografií, což klade nároky na výpočetní výkon procesoru. Tato zařízení jsou také využívána k přehrávání videa ve vysokém rozlišení nebo hraní 3D her a to zvyšuje nároky na výkon grafického čipu. Při operacích s většími objemy dat, jako jsou fotografie, mapy nebo při nahrávání her do operační paměti, záleží na rychlosti lokálního souborového systému. V dnešní době je velké množství dat, jako dokumenty a obrázky uloženo na síťovém úložišti (cloud), proto je kladen důraz na rychlost síťového připojení.

Cílem bakalářské práce bylo seznámit čtenáře s dostupnými a použitými technologiemi k měření výkonu mobilních zařízení platformy iOS. Dalším cílem práce bylo naimplementovat aplikaci, která realizuje měření rychlosti jednotlivých komponent zařízení a umožní jednoduché porovnání jejich výkonu mezi různými zařízeními.

Při testování procesoru byl měřen počet miliónů operací za jednu vteřinu (MFlops). U testování výkonu grafického čipu byla měřena doba vykreslování jednoho rámce a ta je snadno přepočitatelná na počet vykreslených rámců za jednu vteřinu. Byla testována rychlost souborového systému na množině operací se soubory. U každého měření lze nastavit počet opakování měření, zobrazit graf a nahrát výsledky na server pro další zpracování.

Aplikace byla otestována na simulátoru zařízení iOS verze 5.0 a 5.1 i na zařízení samotném (iPhone 4 verze operačního systému iOS 5.1.1) a je připravena na další rozšíření. Naměřené hodnoty splnily očekávání. Aplikace byla úspěšně naimplementována a splňuje požadavky, které byly v zadání práce.

Literatura

[1] J. Conway. *iPhone Programming: The Big Nerd Ranch Guide*. Addison-Wesley Professional, 2010.

[2] J. J. Dongarra. *Numerical Linear Algebra on High-Performance Computers*. Society for Industrial Mathematics, 1998.

[3] B. Hollings. Cocos3d programming guide. URL: <`http://brenwill.com/2011/cocos3d-programming-guide/`>, [online], cit. 2012-05-08.

[4] L. K. John. *Performance Evaluation and Benchmarking*. CRC Press, 2005.

[5] D. Kaeli. *Computer Performance Evaluation and Benchmarking*. Springer, 2009.

[6] S. G. Kochan. *Objective-C 2.0*. Cpress, Brno, 2010.

[7] A. D. Library. Hfs plus volume format. URL: <`http://developer.apple.com/legacy/mac/library/#technotes/tn/tn1150.html`>, [online], cit. 2012-05-08.

[8] A. D. Library. Networking and internet starting point. URL: <`https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/GS_Networking_iPhone/`>, [online], cit. 2012-05-08.

[9] A. Munshi. *OpenGL ES 2.0 Programming Guide*. Addison-Wesley Professional, 2008.

[10] V. Nahavandipoor. *iOS 5 Programming Cookbook: Solutions and Examples*. O'Reilly Media, 2012.

[11] P. Rideout. *iPhone 3D Programming: Developing Graphical Applications with OpenGL ES*. O'Reilly Media, 2010.

[12] M. K. Rob Napier. *iOS 5 Programming Pushing the Limits*. Wiley, 2011.

[13] E. Sadun. *The iOS 5 Developer's Cookbook: Core Concepts and Essential Recipes for iOS Programmers*. Addison-Wesley Professional, 2012.

[14] D. Salomon. *Data compression : the complete reference*. Springer, 1998.

[15] R. Warren. *Creating iOS 5 Apps: Develop and Design*. Peachpit Press, 2011.

[16] S. Welch. *iOS 5 Core Frameworks: Develop and Design*. Peachpit Pressl, 2011.

[17] Zimerman. How to find prime numbers. URL: <`http://umumble.com/blogs/algorithm/334/`>, [online], cit. 2012-05-08.

A Přílohy

A.1 Struktura přiloženého CD

1. /doc/ - Složka obsahující dokument bakalářské práce ve formátu PDF.
2. /src/bp_iphone_test - Projekt aplikace pro IDE XCode k testování výkonu zařízení iPhone.
   /src/bp_iphone_test/bp_iphone_test/bp_iphone_test.xcodeproj - Hlavní soubor projektu.
3. /tex/ - Zdrojový kód dokumentu se seznamem literatury.
   /tex/images/ - Obrázky použité v dokumentu bakalářské práce.