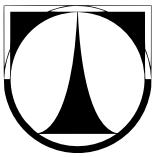
TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií



BAKALÁŘSKÁ PRÁCE

Liberec 2009 **Viktor Bubla**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy

Virtuální měřící a monitorovací přístroj v prostředí GNU/Linux

Virtual measuring and monitoring device under GNU/Linux

Bakalářská práce

Autor: **Viktor Bubla**
Vedoucí práce: Ing. Tomáš Tobiška
Konzultant: Ing. Jan Kraus

V Liberci 24.5.2009

Originál zadání

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Předně bych rád poděkoval celé komunitě lidí, která vyvíjí a stará se o software s otevřeným zdrojovým kódem a zejména o operační systém GNU/Linux[1]. Díky ní jsem mohl já, stejně jako kdokoli na světě, používat legálně a zdarma všechen software, který jsem kdy ke své práci potřeboval i v době, kdy jsem byl jako student bez finančních prostředků.

Právě díky svému zaujetí pro Linux jsem se již v prvním ročníku přes pana Ing. Jana Koprnického, Ph.D., dozvěděl o panu Ing. Janu Krausovi. Ten mě okamžitě nasměroval správným směrem a hned v dalším semestru jsem přešel z jazyka Pascal na mnohem užitečnější jazyk C. Za to jsem mu velice vděčný, protože nepamatuji, že bych se někdy znovu výrazněji setkal s Pascalem jak v souvislosti s Linuxem, tak v dalších oblastech.

Za zmínku jistě stojí i konzultace, při kterých jsem sice většinou zjistil, že velká část předchozího programování byla zbytečná nebo nepoužitelná, na druhou stranu jsem však pokaždé dostal nové nápady, rady, informace a nasměrování, kam se dále ubírat.

Na posledním místě i když v neposlední řadě nemohu nezmínit pohotové a vstřícné rady a updaty zdrojových souborů panem Ing. Tomášem Tobiškou. Jeho přehled a orientace ve všech souvislostech mezi desítkami zdrojových souborů, stovkami struktur a tisíci proměnných na mě pokaždé udělal dojem...

[1] `http://www.linux.cz/`

Abstrakt

Hlavním cílem práce je vytvoření experimentálního virtuálního měřícího přístroje, který je díky platformě GNU/Linux snadno duplikovatelný a umožňuje provádění testů, jenž by byly na fyzických přístrojích obtížně nebo vůbec proveditelné. Autor se snaží chování VMP maximálně přiblížit fyzickému přístroji.

Dalším krokem je provádění konkrétních experimentů na VMP, jejichž výsledky jsou diskutovány a na jejichž základě jsou optimalizovány vybrané parametry. Cíle práce směřují k hledání způsobů vylepšení vlastností stávajících reálných PMD. Velká část práce se zabývá optimalizací ukládání archivních dat.

Klíčová slova: kvalita, virtuální přístroj, Linux, komprese, simulace

Abstract

Main goal of this thesis is developing experimental virtual measuring device, which should be easily duplicateable thanks to GNU/Linux platform. It makes it possible to run tests, that are problematicly or even impossibly executable on real devices. Author mostly tries to make behavior of VMD[2] similar to physical ones.

In next step, exact experiments on VMD are performed. Results are discused and on their basis, chosen parameters are optimized. Findings aim to find out ways to improve properties of present real PMDs. Majority of work is attended to optimalization of archive data storing.

Key words: quality, virtual device, Linux, compression, simulation

[2]VMD Virtual Measuring Device ≈ VMP Virtuální Měřicí Přístroj

Obsah

Seznam obrázků

1 Úvod

PMD neboli Power Monitoring Device je elektronické obvykle vestavné zařízení, sloužící k monitorování kvality dodávek elektrické energie rozvodné sítě v místě instalace. Zařízení provádějí výpočty, zpracovávají a archivují data, umožňují vzdálené monitorování prostřednictvím síťové komunikace a obvykle na displeji zobrazují žádané veličiny. Vývoj firmware a testování s fyzickým přístrojem je možné a běžně používané. Je však nutné mít speciální jak softwarové, tak hardwarové vybavení, nemluvě o fyzickém přístroji. Nabízí se otázka: Proč nevytvořit nezávislý, přenositelný model celého měřicího a monitorovacího řetězce uvnitř naprosto běžného osobního počítače? Řešení této otázky se věnuje převážná část práce. Další část zmiňuje dosavadní provedené praktické experimenty na virtuálním řetězci.

Operační systém GNU/Linux se stává stále oblíbenějším nejen na pracovních stanicích, ale zejména i v nejrůznějších embedded zařízeních. Použití OS místo jednoúčelového kompletního firmware má samozřejmě svoje výhody. Kromě vyšší spolehlivosti a univerzálnosti je jimi jistě i snazší vývoj samotných aplikací, lepší přenositelnost zdrojových kódů a široká podpora hardware. Tato skutečnost je v delším časovém horizontu důvodem vzniku práce, přičemž v budoucnu by mohly být modifikovaný výpočetní blok a další funkce přímo použitelné v PMD založených na OS Linux.

Práce se zabývá myšlenkou vytvoření virtuálního měřicího přístroje tak, aby bez jakýchkoli periferií mohl simulovat běh a chování skutečného přístroje. S tím souvisí potřeba vytvoření programu, který dokáže generovat vzorky virtuálně měřeného signálu a také programu, který zpracuje a ukládá naměřené veličiny. Kromě toho je nutné modifikovat hardwarově závislé funkce výpočetního bloku, aby byl přenositelný a využíval funkcí OS Linux, případně dalších svobodných knihoven s otevřeným zdrojovým kódem. Všechny komponenty je potřeba vytvořit tak, aby měly co nejlépe nastavitelné parametry pro pozdější testování. Jsou používány buď přímo konfigurační soubory čtené při běhu programu, nebo alespoň přepínače parametrů ve zdrojových souborech.

Vzhledem k tomu, že s embedded zařízeními souvisí omezené zdroje výkonu, kapacity paměti a přenosových kanálů, důležitým modulem, kterému je věnována velká pozornost, je kompresní blok. Jeho pomocí jsou testovány různé bezztrátové kompresní algoritmy pro dosažení optimálnějšího využití paměti určené archivním datům, jejich nároky na výkon procesoru a operační paměť. Prověřuje se domněnka o velké redundanci archivních dat a zjišťuje se vliv nejrůznějších struktur vstupních souborů a jejich kódování na účinnost komprese.

V souvislosti s vytvářením modulů komprese a dvou variant generátoru archivů

či vzorků signálu vznikly články na konference EPE[3] [8] a CIRED[4] [9]. Vysvětlení problémů, poznatky a výsledky přeložené do češtiny jsou součástí práce. Jsou zde ilustrovány účinky různých typů datových sad na účinnost komprese a mnoho dalších vlivů. V souvislosti s prováděním benchmarků kompresních metod bylo nutné zvolit způsob měření času. Je uveden způsob použití přesných procesorových časovačů s extrémním rozlišením.

Veškerý vývoj probíhal na platformě GNU/Linux s výhradním použitím volných nástrojů, knihoven, vývojového prostředí, kompilátoru jazyka C a dokonce i WYSIWYM[5] DTP editoru LyX[6]. Nepoužívání placeného software je neoficiálním cílem celé práce a její obsah a forma snad bude reprezentativním příkladem toho, že něco takového je možné.

[3]Electric Power Engineering 2008 `http://www.epe-conference.eu/epe2008/files/partitions2_cz.html`

[4]International Conference on Electricity Distribution 2009 `http://www.cired2009.org/`

[5]WYSIWYM — What You See Is What You Mean

[6]Rychle se vyvíjející kvalitní DTP software, který je GUI frontendem LaTeXu `http://www.lyx.org/`

2 Teoretický úvod

2.1 Optimální metody ukládání dat v PMD

Tato část práce souvisí s tvorbou podkladů a vlastními poznatky při vytváření článku na konferenci Electric Power Engineering 2008 (EPE) [8].

2.1.1 Obecně

Dnešní stav evoluce v oblasti standardů elektromagnetické kompatibility [2, 5] v místě měření definuje kvalitu a její vlastnosti. Během měření jsou zaznamenávány a vyhodnocovány ukazatele, podle kterých se zjišťuje, zda jsou standardy dodržovány. Obvykle se jedná o dlouhodobá měření, jejichž výstupem jsou obrovská množství dat. Jsou specifikovány různé intervaly záznamu dat [4, 3]. Nejkratší interval pro archivaci zde definovaný je 150 period. Pokud to uživatel požaduje, je však možné zaznamenávat i v tak krátkém intervalu, jako 10 cyklů, což odpovídá 200 ms. Pokud se archivují i jen základní hodnoty, není problém, aby během dne archiv dosáhl velikosti stovek MB. Vzhledem k mezinárodním standardům a k obecným vlastnostem energetické sítě se dá očekávat, že mezi měřenými parametry budou pouze malé fluktuace a že v archivech bude vysoký stupeň redundance.

Pro kompaktní měřicí přístroj se slabou konektivitou to znamená mnoho dat, která musí být neustále zpracovávána a ukládána. Médií existuje celá řada, ale v případě PMD jsou data obvykle ukládána do vnitřní paměti v surové binární podobě pomocí suboptimálních kódování pro každou veličinu.

S rostoucím objemem měřených veličin a se zkracováním intervalů archivace se požadavky na vnitřní paměť a propustnost připojení obrovsky zvyšují. Jednoduchým řešením je zvýšení úložné kapacity. Je pravda, že vzhledem k neustálému snižování cen flash disků to není problém, avšak existují i nevýhody, jako třeba pomalé stahování dat nebo dlouhá doba odezvy přístroje a operátorského PC při vzájemné komunikaci, protože se přenáší více bytů, než je nutné.

Redundantní data mohou být také ukládána a odesílána v komprimované podobě pomocí bezztrátové nebo dokonce i ztrátové komprese. Různé komprimační algoritmy v kombinaci s optimálním kódováním veličin snižují nároky na velikost paměti a zrychlují proces získávání dat k dalšímu zpracování nebo vzdálenému monitoringu. Kromě jiného má vliv i různý způsob ukládání dat ve smyslu jejich výhodnějšího uspořádání.

2.1.2 Prováděné simulace

Surová data sledovaných veličin typicky nemají velký rozptyl, což znamená, že obsahují vysoce redundantní blok symbolů a hodnot optimálních pro kompresi. Věnu-

jeme se efektům některých změn ve struktuře zmiňovaného bloku dat vzhledem k účinnosti základních veřejně dostupných kompresních algoritmů. Pro experimenty a srovnání účinnosti komprese jsme vyvinuli nástroj, který umožňuje měnit některé klíčové faktory v archivu komprimovaných dat. Testuje se vliv typu kódování a nastavení archivu, seřazení dat, kvality a intervalu archivace. Variace bloku veličin jsou prováděny na základě reálného PMD. Jsou obsaženy všechny běžně monitorované hodnoty jako napětí, proudy, výkony a energie, harmonické složky a několik dalších v tomto případě nepodstatných záznamů dle funkcí reálného přístroje. Každá veličina je generována v nastavitelném kódování a s parametrizovatelnými rozptyly.

2.1.3 Testované algoritmy

Pro experimentální měření používáme běžně dostupné entropické a slovníkové algoritmy, jejichž zdrojový kód v ANSI C může být lehce získán, je ověřený a může být znovupoužit díky veřejné licenci. Důležitým faktorem při výběru byla skutečnost budoucího použití v embedded zařízení na bázi mikrokontroléru ARM s velmi limitovanými zdroji. Tato skutečnost diskvalifikovala některé pokročilejší ale paměťově a výpočetně náročnější varianty LZ jako gzip nebo bzip2.

Run Length Encoding — RLE

RLE je jedním z nejjednodušších algoritmů, jehož princip spočívá v nahrazení opakující se sekvence bytů v souboru vyhrazeným znakem a číslem udávajícím počet opakování. Tento princip přes svou jednoduchost dosahuje zajímavých výsledků u dat s množstvím opakujících se bytů (je obsažen v mnoha grafických formátech).

- jednoduchá a extrémně rychlá komprese s minimálními nároky na paměť
- téměř zbytečné v netriviálních situacích

Shannon-Fano encoding — SF, Huffman Encoding — HUF

Tyto algoritmy jsou představiteli kategorie entropických kodérů. Princip obou metod [7] spočívá v enkódování bytů do bitových streamů. Na základě množství výskytů každého znaku je ušetřena paměť zakódováním znaků s nejvyšším zastoupením pomocí nejkratšího bitového kódu. Oba kódy používají takzvaný histogram pravděpodobnosti, nebo také bitový strom, který je nutné uložit na začátek zkomprimovaného souboru. Existují optimalizace snažící se zjednodušit kompresi a odstranit potřebu více průchodů, kdy se binární strom rekonstruuje za běhu. SF a HUF nedokáží rozlišovat sekvence stejných symbolů.

Základní použité verze algoritmů se liší ve způsobu stavby kódovací tabulky/stromu. Zatímco SF rekurzivně dělí soubor znaků, HUF buduje strom opačně — spojuje

větve s nejnižším výskytem. Oba algoritmy jsou sice podobně složité, HUF má však obvykle lepší výsledky, protože jeho binární kódování je optimální.

- rychlé, optimální komprese jednotlivých znaků
- účinné i u zašuměných dat
- malé paměťové nároky, jednoduchá, jasná implementace
- nepříliš účinné pro malé množství dat

Lempel-Ziv — LZ77

LZ77 je reprezentantem kategorie slovníkových algoritmů, který posouvá dále ideu RLE a HUF pomocí komprese opakujících se sekvencí (různých) symbolů. Pokud narazí na sekvenci, která se již dříve vyskytla, zanechá na místě pouze odkaz na minulý výskyt a délku identické sekvence. Mnoho existujících variant se liší způsobem výstavby slovníku, přičemž celá řada vylepšených a upravených verzí je chráněna patenty.

- LZ algoritmus je univerzální (dobrá účinnost na jakýchkoli datech)
- optimální pro kompresi textu
- umožňuje snadnou a velice rychlou dekompresi
- vyšší nároky na paměť, pomalé

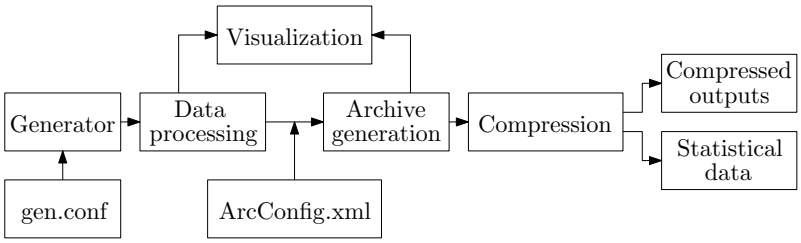
Lempel-Ziv-Oberhumer — LZO, MiniLZO

LZO je vysoce optimalizovaná varianta LZ algoritmu [10], která dosahuje mnohem vyšších rychlostí komprese za cenu mírného snížení účinnosti. Testovaná varianta MiniLZO je binárně kompatibilní algoritmus, avšak bere v úvahu omezené výkonové a paměťové zdroje embedded zařízení.

- velice dobrý kompresní poměr u vysoce redundantních dat
- rychlejší komprese i dekomprese v porovnání s LZ
- střední paměťové nároky, volitelné množství přidělené paměti

2.2 Bezztrátová kódování a kompresní algoritmy aplikované na data kvality elektrické energie

Následující sekce vznikala v souvislosti s prezentací článku na International Conference on Electricity Distribution 2009 v Praze (CIRED) [9].



Obrázek 1: Blokové schema experimentálního systému

2.2.1 Obecně

V další části práce jsme změnili přístup k provádění experimentů, avšak důvody a cíle se neliší od těch popisovaných v sekci 2.1.1.

Opět jsme prováděli testy s cílem prověřit chování různých kompresních algoritmů při kompresi různých typů dat s různým složením a různými vlastnostmi. Pro tento účel jsme nejdříve získali reálný archiv měření. Kromě toho jsme vytvořili modulární systém v prostředí GNU/Linux umožňující generovat virtuální sadu napětí a proudů s velice jemnými možnostmi nastavení tak, abychom se mohli maximálně přiblížit reálnému měření. Generátor byl použit pro přípravu dat určených k vytváření archivů, na nichž můžeme důkladněji studovat efektivitu a stabilitu kompresních algoritmů.

2.2.2 Experimentální generátor měřených dat

Pro účel vytváření dat kvality elektrické energie jsme vyvinuli vlastní modulární systém viz obr. (1), který simuluje funkci PMD v reálném světě. Představený systém se skládá z generátoru vzorků, virtuálního bloku výpočtu kvality, archivace dat a kompresní jednotky. Každý z modulů může být nahrazen jinou implementací tak, aby nabízel data s jinými veličinami a parametry.

Pro generování signálů jsme použili syntézu ze složek, u nichž je možné jednotlivě nastavit napětí, fázi, frekvenci a odchylku, která ovlivňuje jejich náhodné kolísání. Generovaný signál je kromě toho i proměnný v čase. Skládá se z intervalů se stejnými statistickými vlastnostmi s daným počtem opakování. Parametry každého intervalu vzorků mohou být modifikovány jednoduše nastavením v odpovídající části konfiguračního souboru (výchozí je *gen.conf*). Každý kanál (fáze) může být generována naprosto nezávisle na ostatních se svojí sadou parametrů. Celková vzorkovací frekvence, interval generování, datový typ vzorků a množství generovaných dat je opět nastavitelné při každém generování.

Vygenerovaná binární data „naměřených“ vzorků jsou dále zpracována pomocí výpočetního bloku, který simuluje reálný PMD, zpracovává výsledky a jeho výstupem jsou data veličin v různém kódování a různých datových typech. Implementace umožňuje provést simulaci měření kvality elektrické energie ze stejných dat a získat

různé výsledky ve více formátech. Data získaná v tomto kroku jsou výchozím bodem pro provádění testů s bezztrátovými kompresními algoritmy.

Poslední blok je určen ke kompresi vygenerovaných archivů pomocí volných, bez jakéhokoli poplatku dostupných, kompresních metod — RLE, variant LZ, několika entropických algoritmů, aritmetického kódování a blokově třídících gzip a bzip2. Výsledky experimentů jsou porovnávány ve smyslu využití paměti, délky výpočtu, složitosti implementace a zejména dle dosaženého kompresního poměru.

2.2.3 Testované algoritmy

Opět jsme použili skupinu volně a zdarma dostupných algoritmů, popisovaných v (2.1.3). Došlo však k rozšíření této skupiny o několik dalších, pro nás zajímavých, kompresních metod.

Jako další alternativa existuje množství ztrátových metod, které se také prakticky implementují, jako například PQZIP, jenž je v patentovém řízení. V našich experimentech tyto nepoužíváme, protože u nich vždy dochází k určité deformaci dat a ztrátě informace. Porovnávání výsledků by tedy bylo daleko obtížnější, když by se pokaždé mohla výsledná data lišit.

Aritmetická komprese — AC

Aritmetická komprese patří do skupiny entropických metod, která produkuje téměř optimální výstup (lepší, než HUF) na dané skupině symbolů. AR je efektivní při kompresi dat, kde malé množství symbolů množstvím výskytu převažuje nad zbytkem.

- nejoptimálnější testovaný entropický algoritmus
- nižší rychlost oproti SF, HUF, podobná paměťová náročnost

Blokově orientované algoritmy — bzip2, gzip

Bzip2 i gzip jsou relativně moderní a velice efektivní algoritmy. Bzip2 využívá kromě jiného blokové třídění známé jako Burrows-Wheeler transformace. Gzip je pro změnu postaven na DEFLATE algoritmu, používaném v jiné podobě v mnoha patentovaných algoritmech. Gzip patří přímo do projektu GNU. Nejedná se o jednoduché algoritmy, ale o několikavrstvé poměrně složité techniky využívající RLE, BWT[7], MTF[8], HUF, delta kódování či LZ77[9].

- vysoce univerzální a účinné
- vysoce náročné na paměť i výpočetní výkon

[7] Burrows-Wheeler transformace
[8] Move to front
[9] Abraham Lempel a Jacob Ziv v roce 1977

3 Implementace

Pro vývoj všech modulů i jednoúčelových programů byl použit svobodný, nekomerční software. Používaný operační systém je linuxová distribuce Ubuntu, která vychází z jedné z nejstarších a nejrozšířenějších distribučních větví vůbec — Debianu. Zdrojové soubory jsou však přeložitelné na jakémkoli GNU/Linux systému. Binární soubory jsou vytvářeny kompilátorem GCC[10] spouštěným prostřednictvím programu make[11] dle závislostí popsaných v souborech Makefile, které jsou vytvořeny pro jednotlivé projekty. Jako vývojové prostředí bylo zvoleno opensource IDE NetBeans, které původně vzniklo jako vývojové prostředí pro jazyk Java, později však začaly přibývat další podporované jazyky (včetně C/C++). Kromě překládaných souborů jsou využívány i skripty v jazyce BASH[12], které se používají k automatizaci spouštění specifických sad testů. Pouze v jednom případě je použit vlastní formát konfiguračního souboru, pro nějž je součástí programu jednoduchý parser.

Mezi ne úplně běžné knihovny, které jsou nutné pro přeložení některých zdrojových souborů, patří knihovna pro výpočet DFT *libfftw3*[13] a knihovna pro práci s XML libxml2[14].

3.1 Generátor archivů

Generátor archivů je prvním programem vytvářeným za účelem generování dat určených k benchmarku kompresních algoritmů. Jedná se o jednoúčelový program, vytvářející pomocí generování pseudonáhodných čísel s danými rozptyly a parametry archivy dat se strukturou odpovídající archivům vytvářeným reálnými PMD.

3.1.1 Nastavení, spouštění a výstup

Přímo ve zdrojovém kódu je dána struktura generovaných archivů a předdefinované odchylky jednotlivých veličin, při jejichž určování jsme vycházeli z normy ČSN EN 50160 [1]. Volbou parametrů v shellu při spouštění programu *./DataProKompresi* můžeme však volit, zda se mají generovat integer hodnoty, float hodnoty a harmonické, zda se veličiny mají ukládat prokládaně nebo sériově a zda požadujeme delta kódování. Dále je možné zadat globální rozptyly všech veličin a globální pravděpodobnosti všech veličin. V takovém případě se pevně dané hodnoty z programu nahradí zadanými parametry. Poslední volbou je možnost po vygenerování archivu okamžitě spustit kompresi, což je výhodné pro dávkové spouštění desítek testů. Jednotlivé

[10] GNU C Compiler, později GNU Compiler Collection `http://gcc.gnu.org/`
[11] GNU Make `http://www.gnu.org/software/make/`
[12] Bourne Again SHell - BASH `http://www.gnu.org/software/bash/`
[13] `http://www.fftw.org/`
[14] `http://xmlsoft.org/`

Výpis 1 Nutné parametry vypsané generátorem archivů

```
v@r:~/src/komprese$ ./DataProKompresi
Nutne ciselne argumenty oddelene mezerou:
        Pocet celych opakovani
        Pocet vnitrnich opakovani
        Generovat float hodnoty (1/0)
        Generovat integer hodnoty (1/0)
        Generovat harmonicke (1/0)
        Generovat s delta kodovanim (1/0)
        Ihned komprimovat (1/0)
        Spolecna odchylka (>10000 -> neuplatni se)
        Spolecna pravdepodobnost (>10000 -> neuplatni se)
v@r:~/src/komprese$
```

Výpis 2 Příklad spuštění generátoru archivů

```
v@r:~/src/komprese$ ./DataProKompresi 1 300 1 1 1 1 1 500 9500
Generuji data do souboru data-1-300-1-1-1-1-500-9500.bin.
.
Hotovo. Bez chyb.
./komprese data-1-300-1-1-1-1-500-9500.bin log-1-300-1-1-1-1-500-9500.txt BezLZSlow
RLE
            t  0.000903 s
           in  101.180 kB
          out  95.822 kB
        ratio  94.7 %
        speed  112047.610 kB/s
v@r:~/src/komprese$
```

volby tak, jak je vypíše generátor při nesprávném zadání, jsou shrnuty ve výpisu (1).

Výstupem generátoru je binární soubor pojmenovaný *data* s následným výpisem všech parametrů oddělených pomlčkou. Přípona souboru je *.bin*. Jako příklad můžeme uvést vygenerování přibližně 100 kB souboru se sériovým řazením veličin, ukládáním pomocí delta kódování a obsaženými integer i float hodnotami včetně harmonických. Výstup se bude ihned komprimovat. Všechna generovaná data mají společnou dovolenou odchylku 5 % v 95-ti procentech hodnot.

Z výpisu (2) je zřejmé, že rozptyl a pravděpodobnost je zadána v setinách procenta. Důvodem je zjednodušení výpočtů při použití celočíselné hodnoty a zjemnění možné volby. Ve výpisu je zahrnuta i část výstupu navazující komprese. Ve skutečnosti pokračují informace o běhu dalších algoritmů.

3.1.2 Funkce

Při spuštění programu jsou načteny parametry do vnitřních proměnných, aby následně mohly ovlivnit počty běhů vnitřních a celkových cyklů. *Počet celých opakování* jednoduše vygeneruje za sebe žádaný počet archivů. Naopak *počet vnitřních opakování* ovlivňuje počet zapsaných hodnot stejné veličiny sériově za sebou. Hodnota parametrů *Generovat float hodnoty*, *Generovat integer hodnoty* a *Generovat harmonické* jednoduše povolují nebo zakazují zápis veličin s vybraným datovým typem, respektive přidávají či zakazují harmonické. Pokud nastavíme parametr *Generovat*

s delta kódováním na 1, při prvním zápisu je každá veličina uložena nativně a její aktuální hodnota se zachovává v paměti, v dalším běhu je uložen rozdíl aktuální od předchozí hodnoty a předchozí hodnota je aktualizována. Pokud povolíme *Ihned komprimovat*, vygenerovaný soubor je okamžitě předán pro kompresi i se správně zadanými parametry vstupního a logovacího souboru, které logicky korespondují s názvem generovaného souboru, tedy i se zadanými parametry.

Srdcem generátoru je funkce vytvářející pseudonáhodnou hodnotu dle předaných parametrů viz výpis (3). Jsou ji celočíselně předány parametry ideální hodnota, odchylka, pravděpodobnost a násobek. Ideální hodnota zhruba odpovídá hodnotě konkrétní veličiny měřené v síti reálným PMD. Protože však například proudy či napětí v základních jednotkách nejsou vhodné pro použité datové typy, v PMD se ukládají vynásobené danými konstantami, díky čemuž je kromě optimálnějšího využití datového typu možné vyhnout se výpočetně náročnému používání datových typů s plovoucí řádovou čárkou. Poslední parametr *nasobek* přebírá zmíněnou převodní konstantu. Hned za deklarací lokálních proměnných je s ní vynásobena ideální hodnota tak, aby odpovídala formátu uložení v archivu. Druhý a třetí parametr ovlivňuje statistické rozložení pseudonáhodných čísel. Globální nastavení odchylky a pravděpodobnosti má přednost před hodnotami danými normou. Pokud jsou globální parametry nastaveny (menší než $1000 \approx 100\,\%$), jsou jimi přepsány pevně dané hodnoty a dále se počítá pouze s nimi. Následující část do dočasné proměnné uloží buď ideální hodnotu v případě, že je zadána nulová odchylka, pseudonáhodné číslo v mezích daných odchylkou nebo úplně náhodné číslo, které může nabývat jakékoli hodnoty v rozsahu typu unsigned short. Nakonec je proměnná *nahoda* typově konvertována a vrácena jako návratová hodnota funkce. Pseudonáhodné číslo v daných mezích je v použité implementaci získáno jako

$$\text{nahoda} = \text{ideal} \cdot \left(1 + \frac{x}{10000}\right) \quad , \tag{1}$$

kde pro náhodné x platí

$$-\,\text{odchylka} < x < \text{odchylka} \tag{2}$$

a *odchylka* je parametr v rozsahu

$$0 < \text{odchylka} < 10000 \quad . \tag{3}$$

Výpis 3 Funkce generování pseudonáhodného celého čísla

```
unsigned short RandomInt(int ideal, int odchylka, int pravd, int nasobek)
{
        float nahoda;
        unsigned short hodnota;
        ideal *= nasobek;
        if (OdchylkaGlob <= 10000) odchylka = OdchylkaGlob;
        if (PravdGlob <= 10000) pravd = PravdGlob;
        if (odchylka == 0)
                nahoda = ideal;
        else if ((rand() % 100) < pravd)
                nahoda = ideal + (rand() % (2 * ideal * odchylka) - ideal * odchylka) /
                    10000;
        else
                nahoda = rand() / nasobek;
        hodnota = (unsigned short) nahoda;
        return hodnota;
}
```

3.2 Generátor vzorků signálů

Generátor vzorků signálů je prvním a základním modulem VMP. Vše ostatní už v nějaké podobě existovalo a pro naše použití bylo třeba pouze upravit, případně doplnit či vyměnit některé funkce. S kvalitou a věrohodností generovaných signálů stojí a padá význam a funkčnost celého VMP. Z toho vyplývá, že modul generátoru signálů je klíčovou částí celé práce. Největší důraz byl kladen na univerzálnost a konfigurovatelnost jeho chování. Právě proto byla zvolena cesta vytvoření speciálního konfiguračního souboru, jehož možnosti budou dále vysvětleny. Původní myšlenkou bylo vytváření signálů ze zadaných harmonických (ale možno i neharmonických) složek, čímž je možné vytvořit teoreticky libovolný signál. Později bylo nutné rozšířit signály o náhodné složky, které jsou taktéž zadávány v konfiguračním souboru. Počet proměnných musí být volitelný, signál musí mít možnost být v čase proměnný, musí být možné nastavit množství generovaných dat po celých intervalech, musí být možné nastavit vzorkovací frekvenci, musí být možné nastavit datový typ generovaných vzorků, musí být možné prohlédnout si časový průběh signálu — to vše bylo třeba splnit v zájmu zachování univerzality.

3.2.1 Funkce

Program se spouští s jedním parametrem, kterým je cesta ke konfiguračnímu souboru. Funkce *main()* obsahuje pouze volání několika funkcí:

- nejdříve *test_conf()* projde celý konfigurační soubor a nastaví proměnné *max_intervalu, max_prom, max_harm*, *datatype, delka_inter, opakovani, fv, blok, verbose* a *octave*, jejichž název je většinou samovysvětlující. *fv* obsahuje načtenou vzorkovací frekvenci, *blok* je počet period v jednom bloku dat (jednom souboru), *verbose* nastavuje množství výpisů a informací a *octave* je přepínač pro vypsání prvních několika vzorků do souboru vhodného ke zpracová-

ní pomocí Octave[15]. Při parsování souboru jsou detekovány základní chyby v syntaxi. Při jejich objevení program skončí, vypíše neočekávaný znak a jeho pozici.

- Pokud *verbose* > 0, *vypis_info()* vypíše do terminálu proměnné načtené předchozí funkcí.
- Nyní se pomocí knihovní funkce *malloc()* alokuje paměť pro pole *seznam* a *parametry*. *seznam* je pole struktur *INTERVAL* a *parametry* je pole struktur *HARMONICKA*. Detailní popis později.
- *load_param()* projde podobně jako *test_conf()* znovu celý konfigurační soubor. Tentokrát ale mnohem podrobněji, protože postupně načítá nastavené parametry a plní jimi pole *seznam* a *parametry*.
- Pokud *verbose* > 0, *vypis_param()* vypíše do terminálu všechny načtené parametry. To má smysl pouze při ladění, protože reálný konfigurační soubor obsahuje stovky až tisíce harmonických.
- *generuj()* je nejsložitější ze všech funkcí v generátoru. Podrobnější popis dále.
- Nakonec se pouze korektně uvolní alokované místo a pokud nedošlo k žádné chybě, program skončí s klasickou návratovou hodnotou 0.

Struktury *INTERVAL* a *HARMONICKA* jsou použity pro načtení všech parametrů složek signálů z konfiguračního souboru. Jejich definice je uvedena ve výpisu (4). *INTERVAL* obsahuje informaci o počtu opakování signálu popsaného v konfiguračním souboru jedním blokem parametrů, pointer na první odpovídající strukturu *HARMONICKA* v poli parametry, počet proměnných neboli signálů a největší počet složek jednoho signálu. Struktura *HARMONICKA* potom obsahuje amplitudu, frekvenci, fázi a maximální odchylku od zadané amplitudy pro každou složku každého signálu v každém intervalu. Poslední položka *platnost* obsahuje informaci o tom, zda je konkrétní složka v konfiguračním souboru definována. Pokud ne, při výpočtu vzorku se složka nebere v úvahu.

Funkce *generuj()* kromě polí *seznam* a *parametry*, všech proměnných doposud načtených z konfiguračního souboru a několika iteračních lokálních proměnných, obsahuje i dvourozměrné pole typu double, které je používáno pro ukládání odchylkových koeficientů, které se musejí v zájmu podobnosti s realitou měnit po náhodném počtu vzorků na náhodnou velikost. Každá složka signálu je tedy přenásobena svým odchylkovým koeficientem a až potom je složena spolu s dalšími složkami ve výsledný vzorek odpovídajícího signálu. V tomto místě dochází k drobnému

[15] `http://www.gnu.org/software/octave/`

Výpis 4 Struktury uchovávající načtené parametry signálů

```
typedef struct {
        double amp;
        double frekv;
        double faze;
        double odch;
        int platnost;
} HARMONICKA;

typedef struct {
        int opak;
        HARMONICKA *p_harm;
        int poc_prom;
        int poc_harm;
} INTERVAL;
```

Výpis 5 Naznačení funkce generování vzorků

```
smyčka dle počtu opakování
  smyčka dle počtu intervalů
    smyčka s inkrementací času
      když vzorek % (10 * fv) == 10
        aktualizace odchylkových koeficientů
      smyčka podle počtu proměnných
        smyčka podle počtu složek
          vzorek += odchylka * amplituda * sin(2 * pi * f * t + fi)
        konec smyčky
        zápis vzorku do souboru
      konec smyčky
    konec smyčky
  konec smyčky
konec smyčky
```

omezení univerzality generátoru, protože je pevně dané různé chování při vytváření odchylkových koeficientů signálů napětí a proudů (liché respektive sudé signály v konfiguračním souboru). Důvodem je fakt, že úpravy v souvislosti s vytvářením odchylek byly prováděny až po kompletní realizaci generátoru a s aktuálním principem nebylo zpočátku počítáno. Druhým důvodem, proč nedošlo k celkové změně je fakt, že konfigurační soubor by musel obsahovat další množství parametrů a přišel by o poslední špetku přehlednosti. Ve výpisu (5) je zhruba popsána činnost funkce *generuj()*.

Poslední nezmíněnou funkcí je *zapis()*. Má na starost zapsat vygenerovaný vzorek signálu do souboru ve správném datovém typu a případně upozornit na překročení jeho rozsahu. Důležitou činností funkce je také kontrola velikosti „velkého“ výstupního souboru — při přiblížení k limitu 2 GB obdržíme vysvětlení problému a generátor se ukončí. Při nastaveném parametru *octave* dojde k zápisu nejen do binárního souboru, ale i do textového souboru, který je pak možné zpracovat v OO Calc[16], Octave nebo si ho prohlédnout běžným textovým editorem.

[16]Skvělá náhrada MS Excel, samozřejmě zdarma `http://www.openoffice.cz/`

Výpis 6 Popis definice jednoho intervalu signálů

```
# Zpusob zapisu:
# @ pocet_opakovani {
#       [(amplituda,frekvence,faze,odchylka)(amplituda,frekvence,faze,odchylka)...]
#       [(amplituda,frekvence,faze,odchylka)(amplituda,frekvence,faze,odchylka)...]
#       [(amplituda,frekvence,faze,odchylka)(amplituda,frekvence,faze,odchylka)...]
#       }
```

3.2.2 Konfigurační a výstupní soubory

Jak již bylo zmíněno, konfigurační soubor, obvykle *gen.conf*, obsahuje parametry složek signálů, kterými se následně řídí samotný generátor. Mezi společné parametry patří:

- *verbose*, který nastavuje množství vypisovaných informací. Nastavení na 0 znamená „tichý“ režim, kdy není vypisováno téměř nic. Naopak 2 znamená maximální množství informací z průběhu všech funkcí programu.
- *interval* má parametr float hodnotu, která se používá jako násobitel opakování intervalů.
- *fv* určuje vzorkovací frekvenci. Zadává se v počtu samplů na periodu 0,02 s (50 Hz).
- Parametr *blok* je možné použít pro určení počtu sad vzorků ukládaných do číslovaných souborů.
- *datatype* určuje datový typ generovaných vzorků. Může být jedním z *int16, int32, float* a *double*.
- Pokud je uvedeno *octave* (bez parametru), prvních několik period všech signálů se zároveň uloží v textovém formátu do souboru, který skript *graf.m* může použít jako vstupní data pro zobrazení průběhů.
- *opakovani* s celočíselným parametrem nastavuje počet opakování všech dále definovaných intervalů. Při nastavení na nulu se generátor zacyklí dokud ho „zvenčí“ nezastavíme.

Následují definice tzv. intervalů. Každý interval začíná zavináčem následovaným lokálním počtem opakování. Složená závorka otevírá a uzavírá definice signálů v daném intervalu. Hranatá závorka ohraničuje jeden signál. Kulatá závorka obsahuje již parametry jednotlivých (ne)harmonických složek. Parametry jsou amplituda, frekvence, fáze v radiánech a odchylka.

Pro přehlednost je možné doplnit libovolné množství bílých znaků. Je možné používat i komentáře, které musejí být na každém řádku uvozeny křížkem. Ve výpisu (6) je popsána struktura definice intervalu.

Výstupním souborem je obvykle soubor *gen.bin*. Jeho název se však změní, pokud generátoru předáme parametrem jiný než výchozí název konfiguračního souboru. Kromě „velkého“ binárního souboru jsou zároveň data dělena do souborů se stejným názvem, avšak s přidaným pořadovým číslem před příponou *.bin*. Množství vzorků na jeden číslovaný soubor ovlivňuje parametr *blok* v konfiguračním souboru. Posledním výstupním souborem je *gen.m* (ve výchozím případě), který v případě uvedení parametru *octave* obsahuje několik period generovaných signálů pro snadnou kontrolu a vytvoření grafů průběhů.

3.3 Výpočetní blok

Výpočetní blok vychází z firmware reálného PMD.

3.3.1 Funkce

Velice ve zkratce, protože jeho návrh nebyl předmětem této práce, výpočetní blok má na starosti z naměřených vzorků fázových napětí a proudů vypočítat všechny žádané PQ[17] veličiny, uložit je do struktur a generovat archivní data.

3.3.2 Soubory nastavení

Zmíním se pouze o konfiguračním souboru *SmpArcConfig.xml*, kterým nastavujeme chování generování a ukládání archivů (protože s archivy se dále pracuje). Zde je možné nastavit které proměnné a s jakou frekvencí budou ukládány. To je zásadně důležité při porovnávání účinností komprese různých sad dat.

3.3.3 Modifikace portu pro Linux

Na čem naopak většina práce spočívala, byla modifikace firmwaru tak, aby byl použitelný v běžném operačním systému GNU/Linux. Ve všech zdrojových souborech, kde se vyskytovaly hardwarově specifické funkce, byly přidány přepínače preprocesoru, které při kompilaci s přepínačem *-D Linux* změnily chování programu tak, aby na hardware nebyl závislý. Některé klíčové funkce byly vytvořeny speciálně znovu v souboru *Linux.c*. Jedná se například o funkce zápisu archivů do souborů, práce s časem apod. Důležitá je linuxová verze DFT s využitím knihovny fftw3.

3.4 Modul komprese

Kompresní blok umožnil realizaci experimentů zabývajících se vlivem typu a kódování archivních dat na účinnost komprese. Program je však použitelný pro benchmark implementovaných kompresních algoritmů na jakémkoli vstupním souboru. Je

[17] Power Quality

tedy možné jeho pomocí odhadnout ideální typ komprese pro libovolný datový soubor. V současné podobě jsou vstupní data komprimována integrovanými algoritmy RLE, Huffman, LZ, SH a MiniLZO, které doplňují externí programy s metodami AC, Gzip a Bzip2.

3.4.1 Funkce

Funkce programu je velice prostá. Soubor vstupující do komprese, předaný jako první parametr při spuštění, je otevřen a je zjištěna jeho velikost. Po alokaci potřebného množství paměti je soubor načten do bufferu *data* a je alokován další buffer *compressed*, který dále slouží pro ukládání výstupů jednotlivých kompresních algoritmů. Jeho velikost je vypočtena z velikosti vstupního souboru tak, aby v žádném případě nemohlo dojít k jeho přeplnění v extrémních případech, kdy komprese zvětší velikost souboru. Každý algoritmus má jiné požadavky, proto je použit výpočet, který splní všechny.

$$\text{in\_len} + \frac{\text{in\_len}}{16} + 384 \tag{4}$$

Podle poslední verze dokumentace BCL by dokonce mělo stačit menší množství paměti.

$$\frac{(\text{in\_size} \cdot 104 + 50)}{100} + 384 \tag{5}$$

Vzhledem k okolnostem a minimálnímu rozdílu by však byla úprava nepodstatná. Kromě zjištění velikosti souboru a alokace paměti jsou otevřeny i soubory logů, do nichž jsou později ukládány naměřené informace. Jedním z nich je soubor *log.txt*, který se otevírá v režimu přidávání na konec (a), druhým je soubor v režimu „ostrého“ zápisu (w) s názvem předaným jako druhý parametr při spouštění. Význam je zřejmý. Soubor *log.txt* slouží pro ukládání výsledků velkého množství měření, zatímco druhý soubor obsahuje vždy jen data o jednom průběhu komprese, čímž se zjednoduší a zpřehlední dohledávání jednoho konkrétního výsledku.

O zápis naměřených hodnot a informací o průběhu komprese se stará funkce *vypis()*. Jako parametry přebírá dvě struktury, které se v Linuxu používají pro uložení přesné časové informace (podrobněji dále), velikost vstupu a výstupu komprese, název kompresního algoritmu, soubor pro zápis a přepínač výpisu do terminálu. Z parametrů je vypočtena doba výpočtu, kompresní poměr a rychlost výpočtu. Vše je zapsáno do souboru, přičemž je pomocí define možné nastavit, zda se mají ukládat i velikosti vstupního a výstupního souboru. Dle parametru *tisk* mohou být všechny informace taktéž vypsány do terminálu.

Po spuštění jsou do logovacích souborů vypsány záhlaví sloupců zaznamenávaných veličin. Data ze vstupního souboru jsou načtena do paměti a přichází na řadu samotná komprese. Před a po zavolání funkce kompresního algoritmu nebo spuštění

příkazu je změřen čas. Následují funkce *vypis()*. Výsledná data jsou uložena do souborů pojmenovaných *compressed***.dat*, kde hvězdičky nahrazuje zkratka názvu metody.

3.4.2 Měření času

Určení velikosti souboru je hračkou, ale nalezení vhodného způsobu měření času tak, aby výsledky byly stabilní, těžko ovlivnitelné během jiných programů a navíc s dostatečným rozlišením, si vyžádalo detailnější zorientování se v knihovních funkcích libc[18], konkrétně v *time.h*[19]. Popsaný přístup se dobře osvědčil.

Pro ukládání časových údajů jsou využity struktury *timespec*, které obsahují dvě celočíselné hodnoty *tv_sec* pro uložení sekund a *tv_nsec* pro uložení nanosekund. Pokud potřebujeme zjistit rozdíl dvou časových údajů ve strukturách *timespec* jako číslo s plovoucí řádovou čárkou, provedeme výpočet

$$\text{rozdil} = \text{ts2.tv\_sec} - \text{ts1.tv\_sec} + \frac{(\text{ts2.tv\_nsec} - \text{ts1.tv\_nsec})}{10^9}\,[s] \quad . \tag{6}$$

Otázkou zůstává, jak struktury naplnit. K tomu slouží funkce *clock_gettime()*. Má pouze dva parametry. Prvním je makro určující typ použitého časovače a druhým je adresa popisované struktury. Existují 4 druhy hodin. V našem případě má význam *CLOCK_PROCESS_CPUTIME_ID*, což je časovač procesoru s vysokým rozlišením, speciální pro každý proces a *CLOCK_REALTIME*, jímž jsou hodiny reálného času společné pro celý systém (avšak opět s vysokým rozlišením). První zmiňovaný používáme pro měření času výpočtu integrovaných kompresních algoritmů, protože běží ve stejném vlákně. Druhý používáme u externě volaných programů (AC, bzip2, gzip), protože jsou spouštěny pomocí funkce *system()*, jejíž příkaz předaný jako parametr proběhne ve zvláštním vlákně a v případě použití *CLOCK_PROCESS_CPUTIME_ID* tedy logicky dostáváme nesmyslné časy.

3.5 Virtuální měřící přístroj

VMP je spojením vytvořených modulů a snaží se svou funkcí přiblížit reálnému PMD. V současné podobě neobsahuje žádné grafické rozhraní, takže je uživatelsky nezajímavý. Budoucí rozšíření o jakékoli další moduly, například vizualizační, by ale nemělo činit potíže. Funkčním cílem současného stavu bylo dosažení spolupráce všech modulů s možností nastavovat generované signály pomocí konfiguračního souboru generátoru, ovlivňovat činnost výpočetního bloku, respektive ukládání archivů pomocí konfiguračních XML souborů a umožnit spouštění kom-

[18]GNU C Library `http://www.gnu.org/software/libc/`
[19]Hlavičkový soubor funkcí týkajících se práce s časem

presních testů na soubory vznikajících archivů. Nepříliš dokonale implementovaná (rozumějte neuniverzálně, bez možnosti nastavení), nicméně funkční, je schopnost vrácení aktuálních dat v souboru XML přes TCP/IP na požadavek vzdáleného — nyní jednoúčelového — klienta. Tato vlastnost byla doplněna pouze pro jednorázovou potřebu ověření funkčnosti práce s XML soubory. Doplněním jednoduchého přepínače sledujícího parametr příchozího spojení by bylo možné snadno funkčnost rozšířit a vracet v podstatě libovolná data.

3.5.1 Součinnost všech modulů

Funkce *main()*, volající všechny další funkce a spouštějící generátor, je umístěna v souboru *main.c*, patřícího mezi soubory, které byly vytvořeny „od nuly“. Nejedná se o modifikovanou funkci *main()* z originálního firmware. Hned po definici vkládaných hlavičkových souborů je několik přepínačů preprocesoru, které ovlivňují způsob běhu programu.

- *#define VYPISOVAT_VSE* asi není nutné dlouze vysvětlovat. Použití obvykle při ladění, kdy může být užitečné vypsání hodnot prvního sta vzorků signálu, všech vypočtených aktuálních hodnot, doby výpočtu, data a času používaného výpočetním blokem.

- *#define GENEROVAT* spustí před spuštěním výpočtů generátor, který vytvoří (nebo přepíše) binární soubory virtuálních vzorků signálů.

- *#define ZACYKLIT* zajistí spuštění dalšího běhu smyčky programu po zpracování všech vstupních dat. Pokud je zároveň povoleno generování, před dalšími výpočty dojde opět ke spuštění generátoru tak, aby se počítalo z „nových dat“.

- Volba *#define REALNA_SIMULACE* má velice prostou funkci — pozdrží běh programu o necelých 200 ms mezi běhy výpočtů tak, aby program běžel přibližně v reálném čase. Bez této volby náš VMP běžící na rychlém PC zanalyzuje až několik minut vzorků signálu za sekundu, což je skvělé, pokud generujeme data pro testování kompresních algoritmů, ale má hodně daleko k práci reálného PMD co se časové reality týká. Nepříjemné je i zatuhnutí počítače při běhu plnou rychlostí, pokud má pouze jedno jádro (100 % využití CPU).

Volba reálné simulace je jednou z věcí, které by bylo možné a vhodné v budoucnu vylepšit. Místo jednoduchého čekání pomocí *usleep()* by bylo vhodné použít smyčku, která by kontrolovala čas a další výpočet by spustila až při dosažení dalšího násobku 200 ms reálného času. Tím by se zároveň odstranily možné rozdíly v rychlosti běhu na různě výkonném hardware. Druhým vylepšením by mohlo být spuštění generátoru

buď v časech čekání, nebo v jiném vlákně. V současné podobě totiž měření zastaví až do chvíle, kdy jsou vygenerována všechna data (je-li povoleno).

Funkce *termination_handler()* slouží ke korektnímu dokončení právě probíhajících výpočtů v případě, že běh VMP „násilně“ zastavíme (například příkazem *Ctrl+C*). Při prvním pokusu o ukončení je vypsána hláška o akceptování požadavku ukončení. Při opakování požadavku je program ukončen okamžitě.

Následuje funkce *sig_io()*. Její asynchronní spuštění zapříčiní příchozí požadavek vzdáleného klienta o soubor s aktuálními daty. Soubor je tedy vytvořen a jako odpověď odeslán zpět klientovi. Používá se server s neblokujícím TCP socketem, který je otevřen a nastaven hned po spuštění VMP. Byl zvolen port 4343, protože je pro autora snadno zapamatovatelný…

Hlavní smyčka celého systému běží ve funkci *main()*. Po spuštění jsou deklarovány potřebné proměnné, nastaveny reakce na signály (přerušení programu a příchod komunikace po socketu), inicializovány struktury *RTC_TIME* a *RTC_DATE* dle reálného času, ze souboru *SmpArcConfig.xml* je načtena konfigurace archivu do odpovídající struktury, jsou spuštěny funkce inicializace výpočetního bloku, je nastavena struktura *SmpPQSettings* určující sledované limity PQ, je spuštěn generátor dat a vlastní výpočetní smyčka. Zde jsou čteny bloky dat z binárního souboru vzorků signálů a zpracovávají se výpočetním blokem tak dlouho, dokud je s čím počítat. Další činnost závisí na dříve zmíněném nastavení pomocí definic preprocesoru. V případě volby *ZACYKLIT* nedojde k ukončení programu, ale ke skoku na místo spouštění generátoru, který se buď spustí, nebo přeskočí podle *GENEROVAT*. V závislosti na definici *VYPISOVAT_VSE* mohou být neustále vypisovány informace o průběhu programu, hodnotách veličin, délce trvání výpočtů atd. Doba výpočtu na poli záznamů odpovídajících intervalu 200 ms je měřena způsobem pracujícím na stejném principu, jako měření rychlosti komprese v kompresním bloku. Pokud je definována *REALNA_SIMULACE*, je po každém vypočteném intervalu vloženo čekání *usleep(t)* s parametrem vypočteným dle

$$t\,[\mu s] = 2 \cdot 10^5 - (\mathrm{t2.tv\_sec} - \mathrm{t1.tv\_sec}) \cdot 10^6 - \frac{\mathrm{t2.tv\_nsec} - \mathrm{t1.tv\_nsec}}{10^3} \quad , \qquad (7)$$

kde *t1* je struktura typu *timespec* naplněná před spuštěním výpočtů a *t2* je stejná struktura naplněná těsně po ukončení výpočtů. Tímto způsobem je alespoň částečně eliminována různá odchylka od doby běhu intervalu na různých PC či při různém zatížení. Zůstává zde vliv instrukcí mimo výpočty, ty jsou však provedeny alespoň řádově rychleji, než je doba výpočtu na intervalu vzorků signálu.

Výpis 7 Základní použití „one-dimensional DFT of size N“ dle dokumentace libfftw3

```
#include <fftw3.h>
...
{
    fftw_complex *in, *out;
    fftw_plan p;
    ...
    in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
    ...
    fftw_execute(p); /* repeat as needed */
    ...
    fftw_destroy_plan(p);
    fftw_free(in); fftw_free(out);
}
```

3.5.2 Archivy

Výstupem výpočetního bloku je množství archivních dat, která jsou ukládána do několika souborů. Jsou jimi hlavní archiv PQ dat, archiv PQ událostí, elektroměr, profil, log a pro naše dosavadní experimenty nejdůležitější — hlavní archiv. V souboru *Linux.c* jsou umístěny funkce, které se starají o zápis dat do odpovídajících souborů ve chvíli, kdy o to výpočetní blok požádá (ovlivněno nastavením, generovanými signály a konfiguračními soubory). Zajímavou funkcí je *Write_Record()*, která zapisuje hlavní archiv VMP do souboru *archiv.bin*. Obsahuje totiž přepínač, umožňující ukládání dat s delta kódováním. Při definování makra *DELTA_KODOVANI* je vytvořena globální proměnná typu *ArchivniData* s názvem *MinuleHodnoty*, kam se při každém zápisu uloží aktuální hodnoty všech archivovaných veličin, aby se s jejich pomocí mohl v dalším běhu vypočítat rozdíl od aktuálních hodnot, který je teprve ukládán. Při prvním běhu je uložen celý záznam v nativním kódování. Funkce obsahuje i možnost povolit výpis vybraných hodnot pro účel kontroly správné funkce výpočtů.

3.5.3 Linuxové varianty funkcí

Krom zápisových funkcí *Linux.c* obsahuje i specificky řešené funkce využívající standardních knihoven OS Linux. Patří mezi ně *RTC_GetTime()* a *RTC_GetDate()*, které pomocí funkce *clock_gettime()* a struktur typu *time_t*, *tm* a *timespec* z knihovny *time.h* naplní globální instance struktur typu *Time* a *Date*, které jsou již definovány ve zdrojových souborech výpočetního bloku. Úsporně, s využitím krátkých celočíselných typů, udržují aktuální reálný čas potřebný při práci s archivy, elektroměrem atp.

Další naprosto zásadní funkcí je DFT, řešená pomocí knihovny *libfftw3*. Vychází z příkladu, který je uveden v dokumentaci [6] přiložené k balíku zdrojových souborů knihovny. Jedná se o použití jednorozměrné DFT. Ve výpisu (7) je příklad citován.

Struktury *fftw_complex* slouží v našem případě pro uložení vstupních dat, kterými jsou pole N vzorků jednotlivých signálů, a výstupních dat, což jsou komplexní výstupy z DFT. Z nich jsou dále vypočteny amplitudy a fáze jednotlivých složek, jsou vynásobeny různými koeficienty podle toho, zda se jedná o signál proudu či napětí a jsou také vypočteny harmonické a meziharmonické složky. Výsledky jsou ukládány do struktury *AktualniHodnoty* typu *CSMPData*.

Funkce *fftw_plan_dft_1d()* vytvoří tzv. plán, což je objekt, který obsahuje všechna data potřebná pro výpočet DFT. Knihovna umí počítat i dvou, tří a vícerozměrné transformace, jejichž plán připravují funkce *fftw_plan_dft_2d*, *fftw_plan_dft_3d* a *fftw_plan_dft*. Poslední zmíněná přebírá navíc jeden parametr určující dimenzi. Společnými parametry jsou počet vzorků, vstupní a výstupní pole komplexních čísel *in* a *out*, parametr *FFTW_FORWARD* určující směr transformace (při inverzní *FFTW_BACKWARD*) a poslední flag parametr, který rozhoduje o způsobu vytvoření plánu. *FFTW_ESTIMATE* vytvoří suboptimální plán bez jakýchkoli měření, na rozdíl od *FFTW_MEASURE*, jehož volbou bychom povolili provedení rychlostních testů několika možností výpočtů, na jejichž základě by knihovna vytvořila plán s nejrychlejším z nich. Měření dle dokumentace může trvat několik sekund a jeho použití je vhodné v situacích, kdy s jedním plánem budeme provádět velké množství náročných transformací. Zde by bylo pravděpodobně nejlepší začít v případě rychlostních optimalizací VMP, protože výpočet DFT je jednou z časově nejnáročnějších částí výpočtů.

Funkce *fftw_execute()* už pouze spustí vlastní výpočet. Jediným parametrem je předem vytvořený plán, na jehož základě se vybírá metoda výpočtu a v němž jsou obsažena všechna vstupní data i připravený výstupní buffer. Po dokončení výpočtu DFT a zpracování výstupního pole dat je možné „zničit“ plán funkcí *fftw_destroy_plan()* a případně i uvolnit alokované buffery. V našem případě jsme použili pole jako lokální proměnné funkce, *fftw_free()* jsme tedy nepoužili.

3.5.4 Makefile

Program *make* je utilita určená k automatizaci překladu zdrojových kódů do binárních souborů. Na základě definic v souboru *Makefile* dokáže řešit jejich závislosti, zpřehledňuje překlad a zejména šetří čas. Považuji za vhodné uvést zde alespoň význam společných flagů použitých při překladu.

- *-lrt* je přepínač, bez jehož použití bychom nemohli přesně měřit čas
- *-lfftw3* umožňuje použití *libfftw3*
- *-lm* přilinkuje matematické knihovny
- *-lxml2* je důležité kvůli práci s XML

- *-lz* přilinkuje knihovnu *libz*
- *-g* je přepínač ladění
- *-D Linux*, *-D SMPQ* a *-D TALEMA* mají stejnou funkci jako *#define* ve zdrojovém kódu, ale zde je možné je definovat hromadně
- zbývá několik parametrů uvedených za *-I*, určujících adresáře, kde se mají vyhledávat hlavičkové soubory

3.6 Pomocné utility

Zde jsou zařazeny stručné komentáře některých programů, které vznikaly v souvislosti s testováním, používáním nebo jako vedlejší efekt některých experimentů.

StahniActData

Klientská aplikace, která z běžícího VMP na zadané IP adrese a portu stáhne XML soubor obsahující aktuální hodnoty všech veličin.

csv2conf

Pomůcka, která vznikla při vytváření konfiguračních souborů generátoru dat. V tabulkovém editoru (OO Calc) byly zapsány hodnoty složek dle vypočtených reálných grafů, exportovány v textovém formátu *.csv* a následně pomocí *csv2conf* převedeny do syntaxe konfiguračního souboru. Parametrem byly předávány společné odchylky a fáze.

graf.m

Skript programu *Octave*, který načte soubor *gen.m* vytvořený při generování signálů a zobrazí prvních několik period všech proudů a napětí v grafu závislosti hodnoty proměnné na čase.

dekomprese

Program používaný pro ověření korektní funkce kompresních algoritmů a bezztrátovosti. Postupně dekomprimuje všechny soubory *Compressed***.dat* v pracovní složce do *Decompressed***.dat*, kde hvězdičky opět nahrazují zkratky kompresních algoritmů.

Výpis 8 Nápověda vypsaná při chybném spuštění rozdelData

```
v@r:~/src/komprese$ ./rozdelData
Spusteni programu:
        ./rozdelData RADA KONSTANTA BLOK MAXIMUM SOUBOR KOMPRESE
        RADA            ...1 = aritmeticka, 0 = geometricka
        KONSTANTA       ...celociselna konstanta rady
        BLOK            ...celociselna velikost bloku dat v bytech
        MAXIMUM         ...velikost nejvetsiho generovaneho souboru
        KOMPRESE        ...1 = komprimovat, 0 = nekomprimovat
        SOUBOR          ...cesta k souboru dat
v@r:~/src/komprese$
```

rozdelData

Program používaný při přípravě dat pro experimenty (4.1.1). Funkci nejlépe osvětlí výpis (8). Ze vstupního binárního souboru jsou vytvořeny části s velikostí zvyšující se podle aritmetické nebo geometrické řady.

dot2comma

Jednoduchý prográmek, který z textového souboru zadaného jako první parametr vytvoří soubor se jménem dle druhého parametru tak, že všechny tečky nahradí čárkami. Tato změna je zásadně důležitá při importu dat do tabulkového editoru, který obvykle v českém systému používá desetinné čárky místo teček. Pokud provedeme nahrazení teček čárkami až v editoru, obvykle se mnoho hodnot nesmyslně nahradí kalendářními daty a dohledání důvodu „proč ten graf vypadá tak strašně“ nebývá jednoduché. Nakonec poznámka pro každého, kdo s Linuxem někdy pracoval: Ano, stejnou práci vykoná i obyčejný *sed*[20]...

[20]Stream EDitor — sed `http://www.gnu.org/software/sed/`

4 Praktické výsledky

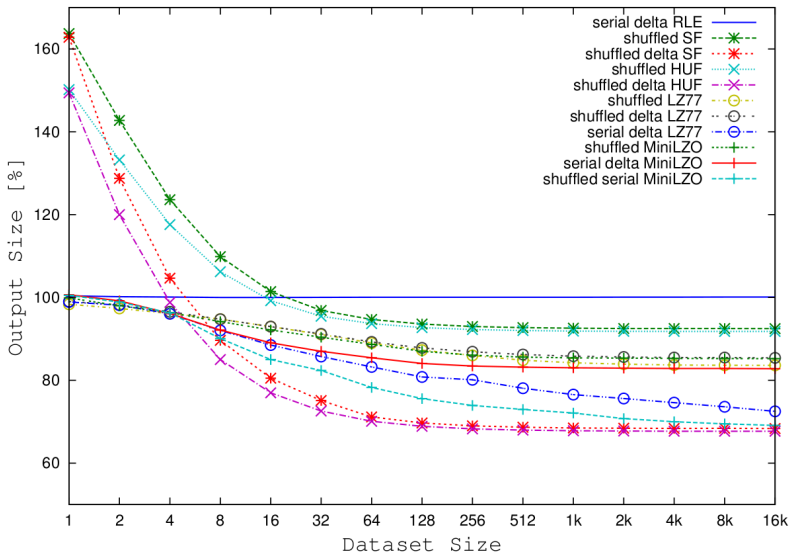
4.1 Optimální metody ukládání dat v PMD

4.1.1 Účinnost komprese

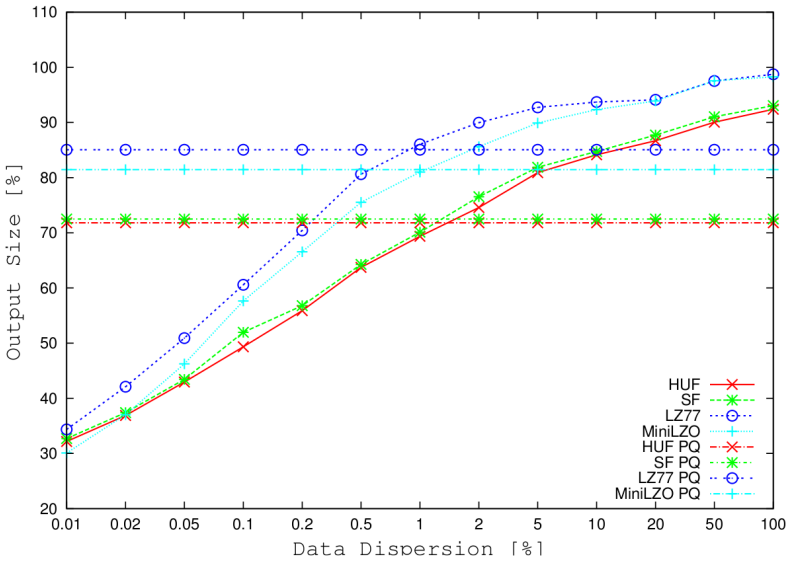
Při praktickém testování byly nejdříve zmíněným generátorem (2.1.2) vytvořeny obrovské spousty archivů s různě nastavovanými parametry. Ve všech případech obsahují vygenerované soubory sdružená a fázová napětí, proudy, 25 harmonických a zanedbatelné množství dalších veličin. Jeden blok všech hodnot má velikost 256 B a ve vygenerovaných souborech se opakuje N-krát. Všechny binární datové soubory byly následně komprimovány pomocí dalšího testovacího programu, který kromě komprese pomocí všech použitých algoritmů zároveň ukládá do logu rychlosti a účinnosti komprese na jednotlivých souborech.

V první sérii experimentů byly měřeny kompresní poměry jednotlivých algoritmů u dat se surovým binárním a delta kódováním. Dalším parametrem byla možnost ukládat data sériově (serial) nebo prokládaně (shuffled). Chování jednotlivých algoritmů je zachyceno na obr. (2), odkud můžeme jasně vyčíst různé chování dvou rozdílných skupin.

Algoritmy založené na entropii (HUF, SF) platí daň za potřebu umístění svého binárního stromu do výstupního souboru, což pro malé vstupní soubory znamená relativní velikost daleko přes 100 %. Při větším množství dat se již tento efekt vytrácí a okolo záznamu 64 bloků dosahuje svého optima, jenž je dále stabilní pro ještě větší soubory. Zatímco rozdíl mezi SF a HUF je minimální a dle očekávání, poměrně zásadně účinnost komprese ovlivňuje způsob kódování dat. Delta kódování evi-



Obrázek 2: Účinnost komprese pro různé velikosti souboru dat



Obrázek 3: Účinnost komprese pro různé hodnoty rozptylových limitů

dentně a razantně zvyšuje účinnost komprese. U entropických algoritmů v grafu nenalezneme sériové řazení, což vyplývá z jejich funkce — nepoznali bychom rozdíl...

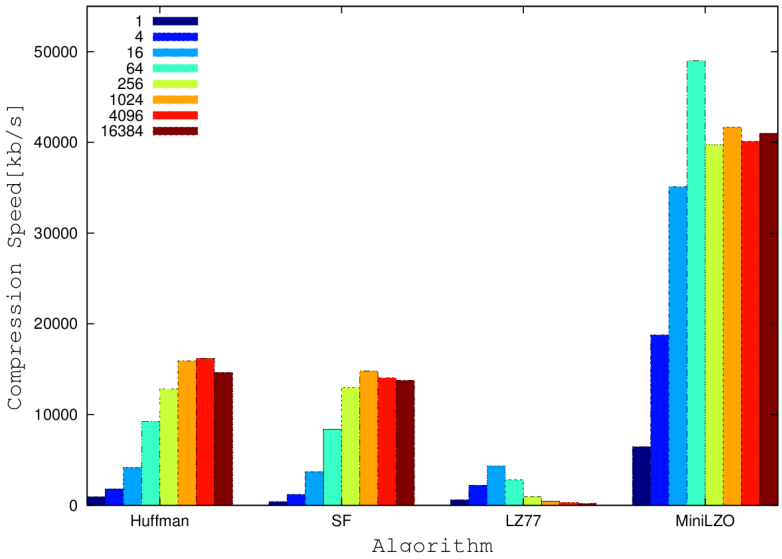
Skupina slovníkových algoritmů netrpí neduhem „nafouknutí“ malých souborů. Kompresní účinnost se ustálí obdobně jako u minulé skupiny, nedosahuje však tak dobrých výsledků. I zde je zřejmá výhoda použití delta kódování veličin.

V dalším kroku dokumentovaném na obr. (3) jsme testovali závislost změny účinnosti komprese na rozptylu veličin v generovaných datech, jenž byl nastavovaným parametrem. Měření ukazuje jak se chovají kompresní algoritmy při nízké a vysoké entropii. V reálném světě jsou si naměřené hodnoty vzhledem k intervalu záznamu v poměrně dlouhém časovém úseku velice blízké a dochází pouze k malým odchylkám, čehož důsledkem je účinnější komprese. Entropické algoritmy opět dosahují lepších výsledků.

Vodorovné čáry, v legendě označené jako PQ, zobrazují kompresní poměr dosažený při rozptylu odpovídajícím limitům EN 50160. Tento případ popisuje situaci, kterou bychom mohli očekávat v síti nízkého napětí. Měření fázového napětí v intervalu jedné minuty během celého týdne pomocí reálného měřícího přístroje však ukazuje maximální odchylku méně než 1,2 V. To znamená, že při nominálním napětí sítě 230 V dochází k největší odchylce 0,22 % okolo klouzající střední hodnoty po relativně dlouhou dobu.

4.1.2 Rychlost komprese

V dalším experimentu jsme provedli opakovaná měření rychlosti jednotlivých kompresních knihoven na stejném datovém souboru. Obr. (4) ukazuje, jak se rychlosti liší. Pro minimalizaci chyb měření jsme každý test spouštěli opakovaně a použili



Obrázek 4: Vliv množství záznamů v archivu na rychlost komprese

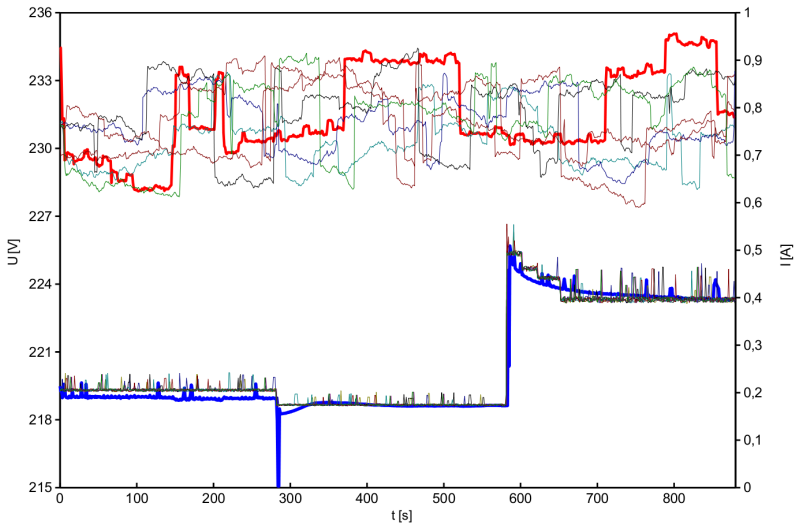
průměrné hodnoty. Výsledky se však lišily minimálně vzhledem k použití časovače měřícího čas běhu programu s rozlišením 1 ns. Je potřeba zmínit, že jsme pro testy použili veřejně dostupné knihovny s různou úrovní optimalizace, což znamená, že výsledky je velice těžké přesně porovnávat. Pro hrubý odhad výpočetní náročnosti jednotlivých algoritmů však graf postačuje.

Je zřejmé, že rychlost komprese se liší velice zásadně — i o několik řádů. Použitá implementace LZ77 byla extrémně pomalá, protože nevyužívala žádnou optimalizaci, zato však byla optimální ve své podstatě vůči kompresnímu poměru. HUF i LZ mají opět dle očekávání velice podobnou, dostatečnou rychlost i pro použití v embedded zařízení. Rychlostně optimalizované MiniLZO je evidentně a nepopiratelně nejrychlejší v celém testu.

4.1.3 Shrnutí Optimálních metod ukládání dat v PMD

Vysvětlili a předvedli jsme výhody a nevýhody různých kompresních algoritmů použitých na měřená data virtuálního PMD. Pro smysluplné velikosti datových souborů s vhodně zvoleným způsobem ukládání a kódování jsme dosáhli značného snížení velikostí výstupu. Typicky bylo pomocí nemodifikovaných víceúčelových kompresních algoritmů dosaženo 80-ti až 50-ti procentní relativní velikosti výstupního souboru. Nejlepšími se jeví ve vybraných oblastech delta kódovaný stream komprimovaný MiniLZO a Huffman.

MiniLZO bylo jednoznačně nejrychlejší kompresní knihovnou, zatímco Huffman podával nejlepší výsledky v účinnosti komprese. Použili jsme BCL implementaci HUF kompresního algoritmu, který podle jeho autora není optimalizován na rychlost. Věřím, že dalšími optimalizacemi by bylo možné rychlost ještě zvýšit.



Obrázek 5: Reálné a generované vzorky napětí a proudu

Dále jsme ukázali, že výběrem správného předzpracování vstupních dat můžeme dosáhnout zajímavé redukce velikosti výstupu. Sériové řazení a delta kódovaní u nejrychlejšího MiniLZO a delta kódování u HUF zvýšilo účinnost komprese o dalších 20 % oproti surovým datům.

V budoucnu plánujeme implementaci vybraných kompresních technik do vyvíjeného reálného měřícího zařízení na platformě ARM.

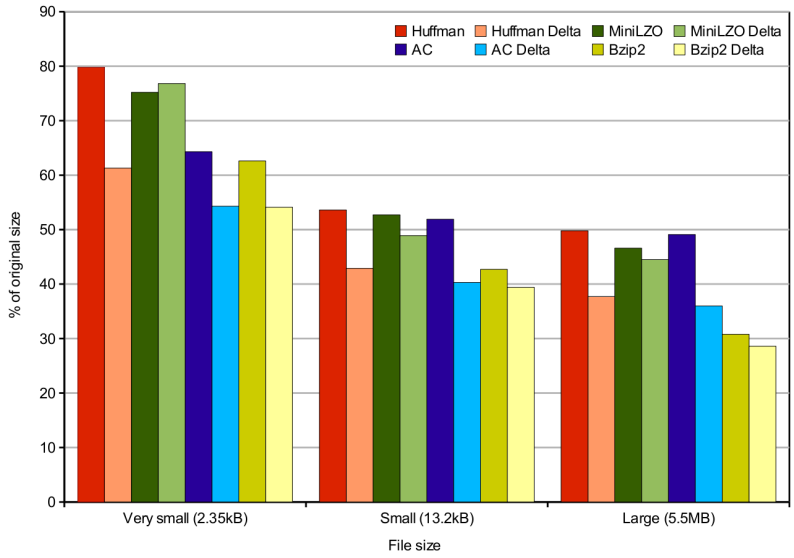
4.2 Bezztrátová kódování a kompresní algoritmy aplikované na data kvality elektrické energie

4.2.1 Srovnání generovaných a reálných dat

V prvním experimentu jsme testovali parametry generovaných dat ve srovnání s hodnotami reálně naměřenými. Na obr. (5) je příklad, který ukazuje několik vygenerovaných průběhů (slabě) a jeden průběh naměřený reálným PMD (silně). Červená silná čára je v praxi naměřené napětí, modrá proud. Na základě těchto experimentů jsme vylepšili možnosti nastavení a vyladili generátor tak, aby virtuálně generované průběhy napětí a proudů měly stejné statistické parametry, jako reálná data. Toto je zásadní vylepšení oproti sekci (2.1) v naší minulé práci, kde pouze náhodně generované archivy zapřičinily zásadně horší výkonnost komprese ve srovnání s reálnými měřeními.

4.2.2 Stabilita komprese

S více náhodně generovanými virtuálně naměřenými daty jsme při stejném experimentu mohli testovat celkovou stabilitu účinnosti komprese. Pro rozumně veliké



Obrázek 6: Účinnost komprese při normálním a delta kódování

bloky archivních dat algoritmy podávaly překvapivě stabilní výkon a rozptyl kompresních poměrů byl obvykle pod 3 %.

4.2.3 Vliv kódování

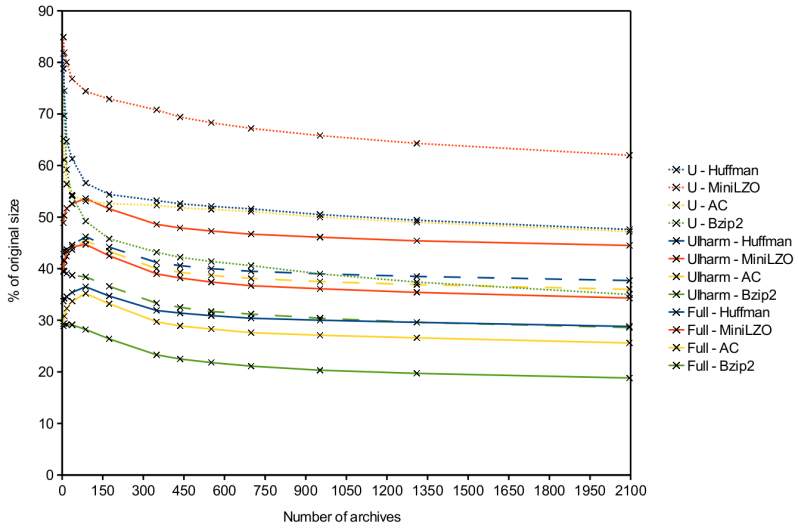
Dalším významným faktorem, ovlivňujícím účinnost komprese, je způsob ukládání veličin při tvorbě archivu. V následujícím experimentu jsme použili běžné a delta kódování při třech různých velikostech archivního souboru. Jak je zřejmé z obr. (6), většinou delta kódování zvýšilo účinnost komprese. V některých případech poměrně zásadně — o více než 10 %. Z grafu můžeme také vyčíst, že mnohem citlivější na způsob ukládání jsou entropické metody.

4.2.4 Vliv velikosti souboru

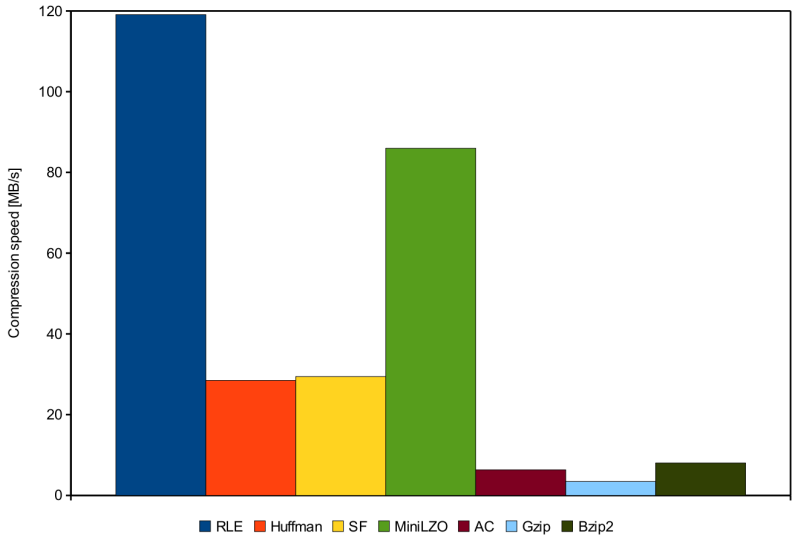
Na obr. (7) je prezentován vliv velikosti archivu na dosažené kompresní poměry testovaných algoritmů. Byly použity tři různé varianty nastavení obsahu archivních záznamů — pouze napětí, napětí, proudy a harmonické a nakonec plné záznamy všech veličin, které PMD monitoruje včetně všech typů výkonů apod.

Jak je vidět, nejlepších výsledků dosahuje blokový algoritmus bzip2, u něhož kompresní poměr klesá až na 20 % původní velikosti. AC a HUF metody s poměry okolo 30ti % mohou být stále zajímavé v situacích, kde je potřeba jednoduchý a přitom výkonný algoritmus. Nakonec varianty LZ kompresních algoritmů reprezentované MiniLZO stále nabízejí smysluplné kompresní poměry, ale zdá se, že jsou nejméně vhodné v aplikacích pro kompresi databází kvality elektrické energie.

Dále můžeme vyčíst i závislost mezi efektivitou a velikostí archivu — výkon většiny algoritmů se v našem případě ustálí okolo 500 záznamů v archivu.



Obrázek 7: Ilustrace vlivu velikosti vstupního souboru na účinnost komprese



Obrázek 8: Srovnání rychlosti komprese včetně Gzip a Bzip2

4.2.5 Rychlost komprese

Posledním experimentem bylo měření průměrné rychlosti komprese. Je třeba zmínit, že měření probíhalo na běžném desktopu, avšak z výsledků můžeme usuzovat na relativní rychlosti v embedded zařízeních. Výsledek je na obr. (8). V reálném zařízení se může výsledek lišit na základě použité architektury (ARM, Intel Atom, x86 apod.) a množství operační paměti. Dalším pro srovnání nepříznivým faktem je různá úroveň optimalizace jednotlivých kompresních knihoven a programů, takže je velmi obtížné srovnávat jednotlivé výkony. Co je však zřejmé, i nejpomalejší algoritmus bzip2, použitý v našem testu, je stále dostatečně rychlý v porovnání s množstvím dat typicky generovaných PMD za celý den.

4.2.6 Shrnutí bezztrátového kódování a kompresních algoritmů aplikovaných na data kvality elektrické energie

S ohledem na naše předchozí experimenty s daty kvality elektrické energie můžeme říci, že použitím běžných kompresních algoritmů dosáhneme zásadního zlepšení využití úložiště dat v moderním PMD. Vytvořili jsme modulární systém schopný generovat pseudonáhodná data pro automatizované a detailnější zkoumání chování těchto algoritmů.

Pro ověření správnosti jsme provedli několik simulací s použitím generátoru a porovnali výsledky s reálnými daty získanými přímo z měřícího přístroje. Jak pro PMD, tak pro výpočetní modul generátoru jsme použili stejný ANSI C zdrojový kód, takže jsme schopni generovat binární pole naprosto stejné struktury. Navíc byl blok generátoru doplněn o zvláštní blok umožňující změnu kódování a vytváření různých podmnožin výstupních dat.

S pomocí generátoru jsme experimentálně ověřili, že použití univerzálních kompresních algoritmů může významně zlepšit využití paměti v embedded PMD. Vliv vstupního kódování a stabilita dosažených výsledků byly ilustrovány na různých typech dat. Nejvýkonnějším algoritmem v našich testech je bzip2 ze skupiny blokových kompresních metod. S delta kódováním byl algoritmus schopen snížit velikost souboru na pouhých 20 % jeho původní velikosti.

Naše budoucí práce by se mohla zaměřit na další vylepšování implementovaného generátoru, případně doplnění dalších parametrů tak, aby se signály generovaných napětí a proudů ještě těsněji přiblížily modelované realitě. Kromě toho chceme použít existující jádro VMP pro vytvoření skutečného měřícího přístroje, který by nativně podporoval kompresi a měl by doplněny komunikační a vizualizační moduly. S takovým zařízením by bylo možné ještě precizněji analyzovat požadavky reálného systému.

4.3 Dílčí poznatky

Měření času

Ze zkušeností s měřením času pomocí funkcí z *time.h* vyplývá, že funkce mohou mít různé nejmenší rozlišení v různých operačních systémech v rámci jednoho hardwaru. Jedná se tedy o rozdíl v implementaci. Je však třeba říci, že nejnižší řády prvku *tv_nsec* ve struktuře *timespec* nejsou ve většině případů důležité, protože i reálné rozlišení ve stovkách či tisících nanosekund přesahuje potřeby přesnosti měření (pokud měříme časy v řádu milisekund až jednotek sekund).

Nabízí se také otázka, zda nejsou „diskriminovány“ algoritmy, jejichž čas výpočtu je měřen pomocí reálného času a ne času vlákna procesu — důvod viz (3.4.2). Ano,

jsou znevýhodněny, protože spouštění příkazu v shellu a start dalšího programu má jistě nezanedbatelnou režii. Záleží však na okolnostech. Je jasné, že při testech rychlosti komprese jsme neměřili s malým množstvím dat tak, aby se tento handicap mohl uplatnit. V případě velmi malých komprimovaných souborů je však zanášena značná chyba. K jejímu přesnému určení bychom museli porovnat časy výpočtu při volání jako externí program s integrovaným řešením.

Využití GOTO

Tento odstavec může docela jistě na mnoho programátorů vyšších jazyků působit jako provokace, ale poprvé v životě (respektive ve svojí krátké praxi s jazykem C) jsem narazil na situaci, kdy je použití *goto* ideálním řešením problému. Do fungujícího *main.c* VMP jsem se rozhodl doplnit přepínač *ZACYKLIT* viz (3.5.1). Použitím standardních postupů, tedy nějakého flagu a funkce *while*, *for* nebo *do–while*, by se kód nejen znepřehlednil, ale i dosti zkomplikoval. Pomocí *#ifdef ZACYKLIT* jsem jednoduše vložil značku *ZNOVU:* na místo žádaného skoku a *goto ZNOVU;* spolu s podmínkou ukončení programu na místo, odkud se má skákat. Řešení funguje a zásah do kódu je minimální.

Závěry a doporučení

S využitím zdrojových kódů skutečného přístroje jsme implementovali modulární VMP. K jeho vzniku vedla poměrně spletitá cesta, vedoucí přes experimenty s kompresí, způsoby generování archivů a vzorků signálů. Na přístroji jsme provedli řadu experimentů, jejichž výsledků v budoucnu využijeme při volbě algoritmu optimalizujícího využití paměti v PMD. Dílčí výsledky jsme úspěšně prezentovali na dvou mezinárodních konferencích EPE [8] a CIRED [9].

Ze studia reálných dat jsme zjistili, že kolísání napětí v síti je obvykle daleko pod limitem daným normou a naše měření v souvislosti s částí (2.1) byla tedy pro hodnocení účinnosti komprese zbytečně náročná. Mnohem pokročilejší „příprava“ dat pro komprimaci pomocí speciálního generátoru vzorků signálu v součinnosti s jádrem reálného PMD (2.2) přinesla svoje ovoce a po pečlivém nastavení algoritmu generování a vyladění konfiguračního souboru jsme získali data, jejichž velikost dokázaly kompresní algoritmy až pětinásobně zmenšit. Co je však důležitější, takto získané archivy mnohem více korespondují s těmi z reálného světa a budoucí využití komprese ve skutečném embedded zařízení je naším cílem.

Ze všech provedených testů vystupují jako vítězové blokové kompresní algoritmy bzip2 a gzip, přičemž v těsném závěsu sekundují entropické algoritmy SF a Huffman, ale zajímavé výsledky podávaly při vhodném kódování i slovníkové algoritmy LZ77 a rychlostně optimalizované MiniLZO. V účinnosti komprese dle očekávání naprosto nedostatečný algoritmus RLE díky své extrémní jednoduchosti naopak všechny poráží svou rychlostí. Jeho účinnost je však natolik zanedbatelná, že by se jeho implementace v žádném případě nevyplatila. Zajímavější situace je v případě MiniLZO, který je druhým nejrychlejším algoritmem v testech a přitom podává smysluplné výsledky. Pokud by nebyly zásadní požadavky na rychlost komprese, mohli bychom uvažovat o použití LZ77, Huffman, SF nebo AC, které podávaly velice dobré výsledky kompresní účinnosti. V ideálním případě, při přebytku výkonu i paměti, bychom jistě vybírali mezi komplexními algoritmy bzip2 a gzip.

Nezajímali jsme se pouze o různé kompresní algoritmy, ale sledovali jsme i vliv různých kódování a řazení dat v archivech. Použití sériového řazení nebo delta kódování bylo na základě jejich funkce u jednotlivých algoritmů a v korespondenci s předpoklady více či méně vhodné. Je logické, že sériové řazení entropický algoritmus vlastně ani nepozná, zatímco slovníkový algoritmus nebo dokonce RLE bude vykazovat zásadní zlepšení kompresní účinnosti. Téměř univerzálního zlepšení dosahujeme použitím delta kódování, které zásadně sníží množství různých znaků ve všech záznamech v archivu (kromě prvního). Ukázalo se také, že blokově třídící kombinované algoritmy zastoupené bzip2 a gzip jsou na změnu kódování nejméně citlivé, protože podobné transformace s daty samy interně provádějí. Pokud však dopředu

přesně známe strukturu dat a již při ukládání použijeme delta kódování, následná komprese jednodušším algoritmem bude vykazovat srovnatelné výsledky a dokonce bude celý proces rychlejší, než při použití složitého (= pomalého) algoritmu.

Jednotlivé běhy generátorů dávaly po kompresi velice blízké výsledky jak v kompresním poměru, tak v rychlosti. Dle tohoto zjištění jsme opět později usoudili, že postupy měření používané při prvních experimentech s generátorem archivů byly zbytečně precizní a pro pouhé srovnání algoritmů přímo megalomanské. Každé měření jsme opakovali se stejným nastavením desetkrát a prezentovali průměrné hodnoty. Pokud někdy docházelo ke zřetelnějším odchylkám, bylo to v případě velice malých archivů, které se však v praxi vyskytují minimálně a pokud ano, u tak malého archivu komprese stejně nemá rozumný smysl. Poznatky jsme se řídili při experimentech s generátorem vzorků signálu. Zde jsme již měření neopakovali vždy, pouze jsme ověřovali namátkou, zda nedochází k popsané nestabilitě.

V budoucnu je možné vydat se mnoha směry. Jedním z nich je další vylepšování generátoru vzorků signálů (generátor archivů je dnes uzavřenou záležitostí). Na generátoru je bez nadsázky možné pracovat donekonečna, protože je neustále možné přidávat nové funkce, upravovat algoritmus nebo možnosti konfiguračního souboru. Dalším směrem je „dotažení“ VMP do stavu, kdy by dokázal nejen provádět uživateli skryté výpočty, ale i vizualizovat měřená data a veličiny, zdokonalit a rozšířit síťovou komunikaci, doplnit možnost ovládat běh nebo vytvořit GUI atd. Posledním možným směrem je příprava a praktická implementace na skutečný hardware, respektive fyzické PMD s implementovanou schopností komprese dat.

Jak bylo zmíněno v úvodu, celá práce skutečně vznikla s využitím pouze volného software, knihoven a zdrojových kódů. I když bylo třeba překonat mnoho překážek, nyní mohu s klidným srdcem říci, že vytvoření jakékoli technické zprávy, vývoj software i obecné používání linuxového operačního systému je pružnější, pohodlnější, příjemnější, rychlejší a přitom zdarma. Je škoda, že stále ještě žijeme uprostřed společnosti zarytě bránící monopoly[21] a odmítající změnu...

[21] *„I am the Linux/UNIX user only!!! It is because of Linux/UNIX is like teepee. No Windows, no Gates, Apache inside!!!“*
[podpis nadšeného uživatele CIJOML na serveru `www.abclinuxu.cz`]

Použitá literatura a prameny

[1] ČSN EN 50160 Charakteristiky napětí elektrické energie dodávané z veřejné distribuční sítě.

[2] Voltage Characteristics in Public Distribution Systems. 2000.

[3] Testing and measurement techniques - General guide on harmonics and interharmonics measurements and instrumentation for power supply systems and equipment connected thereto. 2002.

[4] Testing and measurement techniques – Power quality measurement methods. 2003.

[5] Pravidla provozování distribučních soustav, Příloha 3: Kvalita elektřiny v distribuční soustavě a způsoby jejího zjišťování. 2006.

[6] FRIGO, M.; JOHNSON, S. G.: Fastest Fourier Transform in the West. 2009.
URL `http://www.fftw.org/`

[7] GEELNARD, B.: Basic Compression Library Manual API version 1.2. 2006.
URL `http://bcl.comli.eu/`

[8] KRAUS, J.; BUBLA, V.: Optimal Methods for Data Storage in Performance Measuring and Monitoring Devices. 2008.

[9] KRAUS, J.; TOBIŠKA, T.; BUBLA, V.: Looseless Encodings and Compression Algorithms Applied on Power Quality Datasets. 2009.

[10] OBERHUMER, M.: LZO Real-time data compression library, Version 2.02. 2005.
URL `http://www.oberhumer.com/`

Nomenklatura

ANSI C Verze jazyka C standardizovaná v roce 1990.

BEZZTRÁTOVÁ KOMPRESE Zmenšení velikosti souboru bez ztráty informace.

BINÁRNÍ STROM Tabulka symbolů uspořádaných dle pravděpodobnosti výskytu v datovém souboru.

DELTA KÓDOVÁNÍ Ukládá se pouze rozdíl aktuální a minulé hodnoty veličiny.

EMBEDDED ZAŘÍZENÍ Zabudovaný, jednoúčelový, specializovaný přístroj.

ENTROPIE Míra neuspořádanosti symbolů v souboru.

FLUKTUACE Náhodné kolísání veličiny kolem střední hodnoty.

GNU/LINUX Jádro operačního systému, knihovny a nástroje z projektu GNU.

KÓDOVÁNÍ Transformace informace z jednoho systému znaků do jiného.

MONITORING Sledování, ovládání a získávání dat ze vzdáleného přístroje.

PMD Performance Measuring and Monitoring Device.

PROKLÁDANÉ UKLÁDÁNÍ Zapisují se celé bloky všech veličin — obvyklé při práci v reálném čase.

PSEUDONÁHODNÁ ČÍSLA Vytvářejí posloupnost, která se zdá být náhodná. Jsou však vytvářena deterministickým algoritmem.

REDUNDANCE DAT Nadbytečnost. Více dat, než je nutné.

SHELL Interpret pro vytvoření příkazového řádku.

SLOVNÍK V našem případě tabulka opakujících se sekvencí znaků.

SÉRIOVÉ UKLÁDÁNÍ Hodnoty každé veličiny jsou v archivu v souvislém bloku.

VMP Autonomní Virtuální Měřící Přístroj určený k simulacím.

Rejstřík