VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SEMI-ANALYTICKÉ VÝPOČTY A SPOJITÁ SIMULACE

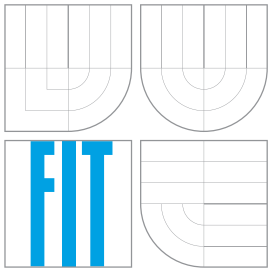
DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. JAN KOPŘIVA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SEMI-ANALYTICKÉ VÝPOČTY A SPOJITÁ SIMULACE

SEMI-ANALYTICAL COMPUTATIONS AND CONTINUOUS SYSTEMS SIMULATION

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. JAN KOPŘIVA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. JIŘÍ KUNOVSKÝ, CSc.

BRNO 2013

Abstrakt

Práce se zabývá urychlením a zpřesněním numerických výpočtů, především pak úloh z oblasti diferenciálního počtu. Zmíněné vlastnosti jsou charakteristické pro skupinu výpočtů nazývaných semi-analytické. Jako jednou z možností urychlení výpočtu obyčejných diferenciálních rovnic se jeví paralelizace. Předkládaná paralelizace je založena na transformaci numerického řešení do aritmetiky zbytkových tříd, která je rozšířena o výpočty s pohyblivou čárkou. Součástí práce je i nový algoritmus pro součin čísel a jeho redukci. Vzhledem k aplikacím v diferenciálním počtu jsou v práci popsány upravené integrační metody - Eulerova, Runge - Kutta a Taylorova s využitím aritmetiky zbytkových tříd. V závěru jsou také nástíněny další možnosti rozšíření a urychlení popsané aritmetiky.

Abstract

My thesis deal with speedup and accurency of numerical computation, especially with solution of differential equation. Algorithms, which are fulling these conditions are named semi-analytical. One posibility how to accelerate computation of ordinary differential equation is paralelization. Introduced paralelization is based on transformation numerical solution to residue number system, which is extended to floating point. A new algorithm for modulo multiplication is also proposed. Because the main goal are applications in differential calculus it is dicussed numeric integration with modified Euler, Runge - Kutta and Taylor methods in residue number system. At the end they are introduced next possibilities and extension for implemented residue number system.

Klíčová slova

metoda Taylorovy řady, obyčejné diferenciální rovnice, paralelní výpočty, aritmetika zbytkových tříd, výpočty diferenciálních rovnic v aritmetice zbytkových tříd, algoritmus pro modulo součin založený na binárních stromech

Keywords

method Taylor's series,ordinally differential equations, parallel computation, resudue number system, computation of differential equation in residue number system, algoritm for modulo multiplication based on binary trees

Citace

Jan Kopřiva: Semi-analytické výpočty a spojitá simulace, disertační práce, Brno, FIT VUT v Brně, 2013

Semi-analytické výpočty a spojitá simulace

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením pana doc. Ing. Jiřího Kunovského CSc. V práci jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Jan Kopřiva
10. června 2014

Poděkování

Chtěl bych poděkovat svému školiteli doc. Ing. Jiřímu Kunovskému, CSc. za jeho podporu a vedení během mého studia, za cenné rady, připomínky a komentáře k mé práci. Také bych chtěl tímto způsobem poděkovat všem, kteří mě během studia podporovali a bez nichž by tato práce nemohla vzniknout.

Obsah

Kapitola 1

Úvod

V současné době naráží počítačové modelování a simulace na dva zásadní problémy.

Prvním problémem je nedostačující přesnost prováděných simulací, která ovlivňuje jejich zpětnou aplikaci do reálného světa . Tato skutečnost se nejčastěji řeší použitím novějších architektur s větším bitovým slovem (např. použitím 64 - bitových architektur místo 32 -bitových) nebo použitím speciálních aritmetik v numerických programech.

Druhým problémem jsou fyzikální vlastnosti používaných technologií. V důsledku to znamená, že možnosti pro zmenšení tranzistorů nebo zvýšení pracovní frekvenci procesoru jsou omezené . V praxi se tento problém řeší paralelizací výpočtu, kdy několik procesorů počítá jednotlivé části celkového problému a na konci výpočtu dojde k jejich syntéze a získání řešení problému (např. vykreslování scény pomocí grafického procesoru).

Ve své práci se soustředím na hledání nových způsobů řešení zmíněných dvou typů problémů v oblasti řešení obyčejných diferenciálních rovnic (ODR). Nejprve naváži na výzkum, zabývající se numerickým řešením ODR pomocí metody Taylorovy řady a následně porovnám numerický přístup s algebraickým řešením označovaným jako symbolické výpočty. Jako možným východiskem pro zpřesnění a urychlení výpočtu ODR jsem určil kombinovaný přístup založený na celočíselné aritmetice zbytkových tříd (RNS) a modifikovaných numerických metodách. Podrobněji lze cíle mé práce charakterizovat následovně:

- shrnutí současného stavu a algoritmů používaných v RNS,
- vytvoření aritmetiky v plovoucí řádové čárce na principech RNS (FP RNS),
- implementace FP RNS v paralelním prostředí a popis vlastností,
- modifikace numerických metod pro dosažení přesnosti bez zaokrouhlovacích chyb ve FP RNS,
- srovnání přesnosti výpočtu ODR v aritmetice FP RNS a standardních aritmetikách.

Výše uvedené cíle jsou součástí prezentované práce.

Kapitola 2

Teorie zbytkových tříd

Paralelní výpočty, které jsou dále popsány staví na poznatcích z oboru teorie čísel. V této kapitole jsou uvedeny nejdůležitější definice.

2.1 Více - modulová aritmetika zbytkových tříd

Při použití čísel v jedno - modulové aritmetice (aritmetika obsahující pouze jeden modul b (číslo, dle kterého se provádí operace *mod*) a k němu přiřazený zbytek a (číslo vzniklé po provedení operace *mod*), zapisuje se jako $|a|_b$) může dojít k tzv. *pseudo* přetečení. Pseudo přetečení je možné zabránit buď použitím vhodného algoritmu (v případě výpočtu multiplikativní inverze pomocí Euklidova rozšířeného algoritmu) nebo použitím více - modulové aritmetiky, kdy mezivýsledek aritmetických operací nebude nikdy větší než vzájemný součin jednotlivých modul.

Definice 2.1.1. Je dána uspořádaná n-tice modul $\beta = (m_1, m_2, ..., m_n)$, kde $m_1, .., m_n \in \mathbb{Z}$, pokud jsou modula vzájemně nesoudělná, označení β se nazývá *bázovým vektorem* systému zbytkových tříd.

Další definice se použije při určení velikosti reprezentovaných bitových čísel.

Definice 2.1.2. Součin modulů bázového vektoru β se bude označovat symbolem M:

$$M = \prod_{i=1}^{n} m_i.$$

Nyní se naváže na jedno - modulové zbytkové třídy a bude uvedena následující definice.

Definice 2.1.3. Nechť $N, m_1, m_{2,}..., m_n \in \mathbb{Z}$ a $\beta = (m_1, m_2, ..., m_n)$ je bázový vektor zbytkových tříd, pak uspořádaná n-tice:

$$|N|_\beta = (|\ N\ |_{m_1}, |\ N\ |_{m_2}, ..., |\ N\ |_{m_n}),$$

se nazývá *reziduální (zbytkovou) reprezentací k bázovému vektoru* β a označuje se jako $|N|_\beta$.

Věta 2.1.4. *Systém zbytkových tříd $\mathbb{Z}_\beta$ je isomorfní s množinou zbytkových tříd $\mathbb{Z}_M$.*

Vzhledem k uvedené větě lze uvést souvislost mezi $\mathbb{Z}_\beta$ a $\mathbb{Z}_M$ na základě zbytku.

Věta 2.1.5. *Čísla $s,t \in \mathbb{Z}$ mají stejnou zbytkovou reprezentaci vzhledem k bázi β systému zbytkových tříd ($| s |_\beta = | t |_\beta$), pokud platí:*

$$s \equiv t \,(mod\, M).$$

Věta tedy říká, že pokud jsou dvě hodnoty sobě rovny, na základě kongruence k modulu M, pak mají stejné vyjádření i vůči vektoru zbytků, získaného pomocí bázového vektoru β a naopak.

Příklad 2.1.6. V následující tabulce je uvedena reprezentace celých čísel pro bázový vektor $\beta = (3,5)$ a modul M

$\mathbb{Z}_M$	0	1	2	3	4	5	6	7
$\mathbb{Z}_\beta$	(0,0)	(1,1)	(2,2)	(0,3)	(1,4)	(2,0)	(0,1)	(1,2)
$\mathbb{Z}_M$	8	9	10	11	12	13	14	
$\mathbb{Z}_\beta$	(2,3)	(0,4)	(1,0)	(2,1)	(0,2)	(1,3)	(2,4)	

Tabulka 2.1.1: Reprezentace čísel ve zbytkových třídách a jejich systémech

Nyní uvedeme několik vět týkajících se aritmetických operací v $\mathbb{Z}_\beta$.

Věta 2.1.7. *Pro čísla $a,b \in \mathbb{Z}$ a bázový vektor modul $\beta = (m_1, m_2, ..., m_n)$ je výsledkem operace $a \pm b$ v $\mathbb{Z}_\beta$ uspořádaná n-tice:*

$$| a \pm b |_\beta = (z_1, z_2, ..., z_n),$$

kde

$$z_i = \left| |a|_{m_i} \pm |b|_{m_i} \right|_{m_i}, i = 1,2,...,n.$$

Následuje podobná věta týkající se operace násobení.

Věta 2.1.8. *Pro čísla $a,b \in \mathbb{Z}$ a bázový vektor modul $\beta = (m_1, m_2, ..., m_n)$ je výsledkem operace $a \cdot b$ v $\mathbb{Z}_\beta$ uspořádaná n-tice:*

$$| a \cdot b |_\beta = (z_1, z_2, ..., z_n),$$

kde

$$z_i = \left| |a|_{m_i} \cdot |b|_{m_i} \right|_{m_i}, i = 1,2,...,n.$$

V případě dělení se vychází z multiplikativní inverze.

Definice 2.1.9. Nechť $a \in \mathbb{Z}$ a bázový vektor modul $\beta = (m_1, m_2, ..., m_n)$. Pokud existují multiplikativní inverze $|a^{-1}|_{m_1}, |a^{-1}|_{m_2}, ..., |a^{-1}|_{m_n}$. Potom n-tice

$$|a^{-1}|_\beta = (|a^{-1}|_{m_1}, |a^{-1}|_{m_2}, ..., |a^{-1}|_{m_n}),$$

je standardní reziduální reprezentace multiplikativní inverze čísla vzhledem k bázovému vektoru β.

Z definice je patrné, že modul M nebude nikdy prvočíslem, proto $(\mathbb{Z}_M, +, \cdot)$ bude pouze komunikativním okruhem. Což má vliv na nalezení multiplikativního inverzního prvku, který pro daný modul bázového vektoru β nemusí existovat. I přes výše popsané nedostatky může být více - modulová aritmetika užitečná, především kvůli rozdělení výpočtu s velkým číslem na nezávislé vektory (procesy).

Shrnutí

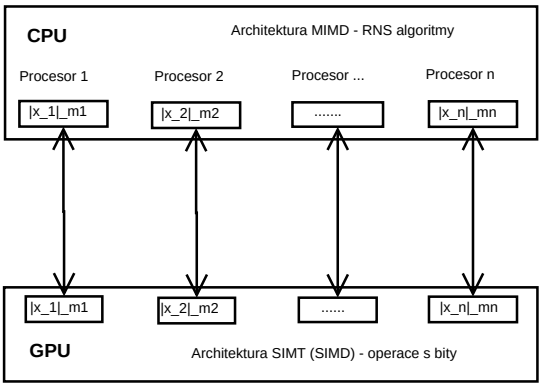
Kapitola objasňuje pojmy z teorie zbytkových tříd, na nichž staví výzkum uvedený v disertační práci. Je zde uvedena více - modulová aritmetika, kdy jednotlivé modula aritmetiky a k ním příslušející zbytky lze pokládat za nezávislé procesory, které jsou součástí paralelního systému.

Kapitola 3

Paralelizace aritmetiky zbytkových tříd

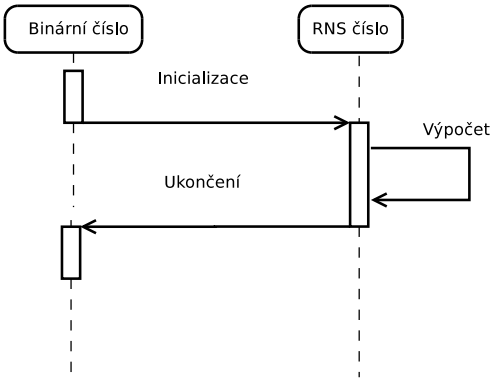
3.1 Paralelní systém FP RNS

Systém, který jsem navrhl a implementoval, lze charakterizovat jako úlohově paralelní (s každým modulem čísla RNS se pracuje zvlášť). Implementovaný systém rozšiřuje aritmetiku RNS o práci s desetinnými čísly a nazval jsem ho FP RNS. Během implementace byla také použita technologie OpenCL, která se využívá pro výpočty na grafických kartách. Na následujícím obrázku je možné vidět, jak se rozmístily zbytky dle modulů m_i na odpovídající procesory v multiprocesorovém systému. Protože stejné operace se provádějí se všemi moduly čísla, lze také využít datový paralelismus, který nabízí OpenCL a využít grafickou kartu (GPU). Tento testovaný přístup však nebyl efektivní, proto je zde umístěn jen jako další možnost urychlení výpočtu RNS v budoucí specifikaci OpenCL.



Obrázek 3.1.1: Paralelní systém RNS - architektura

Průběh výpočtu v implementovaném systému se skládá z několika částí. Nejprve se zvolené číslo převede ze standardní reprezentace do FP RNS a poté se pomocí paralelního systému provádějí výpočty. Nakonec se z modulů v FP RNS sestaví číslo ve standardní aritmetice. Sekvenční diagram těchto činností je zobrazen na následujícím obrázku.

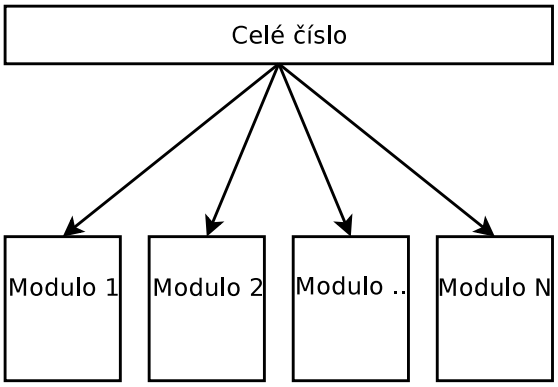


Obrázek 3.1.2: Paralelní systém RNS - komunikace

Jednotlivé fáze výpočtu budou popsány dále.

3.2 Převod celých čísel do RNS (FP RNS) a zpět

Převod celých čísel se provádí v implementovaném systému pomocí operací modulo, jak naznačuje obrázek 3.2.1(na obrázku je znázorněn převod celého čísla do RNS (tento převod se neliší od převodu do FP RNS):



Obrázek 3.2.1: Převod čísla ve standardním vyjádření do RNS

Vzhledem k tomu, že modulované číslo $X \in \mathbb{Z}$ se během převodu nemění, lze číslo modulovat moduly m_i paralelně, tedy v případě použití MIMD (SIMT) architektury umístit modulované číslo do sdílené (konstantní paměti) a jednotlivé procesory (vlákna) nechat číst z těchto pamětí.

Pro převod ze systému RNS do standardní poziční soustavy (desítkové, dvojkové) se používají dva základní principy, jeden je založen na použití transformace do MRS a dále do standardní poziční soustavy, druhý je založen na čínské větě o zbytcích (CRT).

3.3 Sčítání v RNS

Sčítání v RNS je složeno ze součtu jednotlivých složek vektoru RNS. Sčítání složek lze provést paralelně i sériově. Nejprve jsem implementoval sčítání pomocí datového paralelismu (SIMT), pak následovala implementace pomocí výpočetních jader (vláken) v mikroprocesoru (MIMD). Tyto architektury jsou podrobněji popsány v příloze práce C.3 a C.4.

RNS algoritmus pro sčítání

Zobecněný algoritmus pro sčítání čísel v RNS bude vypadá následovně:

Algoritmus 1 RNS sčítání

Require: $A = (|a_1|_{m_1}|a_2|_{m_2}...|a_n|_{m_n}), B = (|b_1|_{m_1}|b_2|_{m_2}...|b_n|_{m_n})$
Ensure: $C = (A+B) \, mod \, M$
  for $i = 1$ to n **do**
    $x_i = (a_i + b_i)$
    $c_i = x_i \, mod \, m_i$
  end for
  return C

Z algoritmu je zřejmé, že sčítání v jednotlivých modulech je paralelní, protože platí $a_i, b_i < m_i$, pak $x_i \leqq 2 \times (m_i - 1)$ a místo složité operace modulo se v případě $x_i \geqq m_i$ použije odčítání modula m_i.

Binární sčítání na architektuře SIMT

Sčítání čísel v RNS jsem prováděl také na bitové úrovni. Zde lze využít několika různých hardwarových návrhů. Vzhledem k časové složitosti bylo zvoleno sčítání, založené na CLA sčítačkách. Vyskytují se čtyři typy těchto sčítaček: Ladner - Fisher, Kogge - Stone, Brent - Kung a Han - Carlson. Specifikaci OpenCL nejlépe odpovídali sčítačky Koggle - Stone a Ladner-Fisher. V implementaci byla dána přednost návrhu založenému na Koggle - Stone vzhledem k možnosti použití skupin proudových procesorů v každém kroku výpočtu a relativně snadnému mapování na jednotlivá výpočetní vlákna architektury SIMT. Binární sčítání založené na tomto návrhu se vyznačuje časovou složitostí $O(log\, n)$. Implementace obsahuje cyklus, jehož délka se na každém stupni logaritmicky sníží.

	500 bitů (ms)	1000 bitů (ms)	5000 bitů (ms)	10000 bitů (ms)
CPU (avg. 10000 exec.)	15	32	123	249
GPU (avg. 10000 exec.)	3916	15102	431621	963999

Tabulka 3.3.1: SIMT (GPU) sčítání

Uvedená tabulka ukazuje sčítání dvou čísel RNS o bitových šířkách (500,1000,5000,10000) a jím odpovídající časy, kdy bylo sčítání (Koggle - Stone) spuštěno na CPU a následně na GPU (SIMT). Z tabulky je patrné, že implementace RNS sčítání na GPU časově značně zaostává za implementací sčítání na CPU. Důvodem je především prodleva globální paměti použité na GPU, která je použita pro ukládání mezivýsledku.

3.4 Odečítání v RNS

Odečítání je v aritmetice RNS složitější oproti odečítání ve standardní reprezentaci. Vzhledem k použití zbytků $X \in \mathbb{N}_0$ nelze využít předností záporných čísel. Popis techniky využívající pouze kladnou poloosu se nazývá *aditivní inverze* .

RNS algoritmus pro odečítání

Zobecněný algoritmus pro odečítání čísel v RNS vypadá následovně:

Algoritmus 2 RNS odečítání

Ensure: $A = (|a_1|_{m_1}, |a_2|_{m_2}, ..., |a_n|_{m_n})_2, B = (|b_1|_{m_1}, |b_2|_{m_2}, ..., |b_n|_{m_n})_2$
Require: $C = (A - B) \, mod \, M$
  for $i = 1$ to n **do**
    if $a_i < b_i$ **then**
      $x_i = (m_i - b_i)$
      $x_i = (a_i + b_i)$
    else
      $x_i = (a_i - b_i)$
    end if
    $c_i = x_i$
  end for
  return C

Z výše uvedeného algoritmu je zřejmé, že odečítání v RNS lze provést paralelně (pokud je číslo, od kterého se odečítá menší, provede se odčítání od báze m_i).

Binární odečítání na architektuře SIMT

Odečítání čísel RNS lze provést stejně jako sčítání, pouze se použijí opačná čísla (v binární soustavě invertují). Výsledky pro RNS odečítání jsou obdobné jako v předcházející tabulce 3.3.1, z nichž je patrná neefektivita výpočtu v případě použití GPU.

3.5 Násobení v RNS

Výsledkem násobení v RNS při použití maximálních čísel $A = M - 1, B = M - 1$ je číslo $C = (A \times B) \, mod \, M$, kde operace modulo M je výpočetně náročná (obvykle 4 - 5 více výpočetních cyklů než sčítání). Proto bylo představeno několik algoritmů pro urychlení modulárního násobení - Montgomeryho algoritmus [12] a Barettův algoritmus [2].

Kop algoritmus pro RNS násobení

Prezentovaný algoritmus jsem navrhl a implementoval na základě znalosti chování binárních stromů. Je určen pro výpočet modulo součinu dvou čísel, kdy každé z čísel je menší než modulované číslo M. Vzhledem k tomu, že používá stejnou modulární redukci v každém řádu binárního stromu, není třeba počítat všechny uzly tohoto řádu. Formální zápis algoritmu je následující:

Algoritmus 3 Kop algoritmus pro RNS

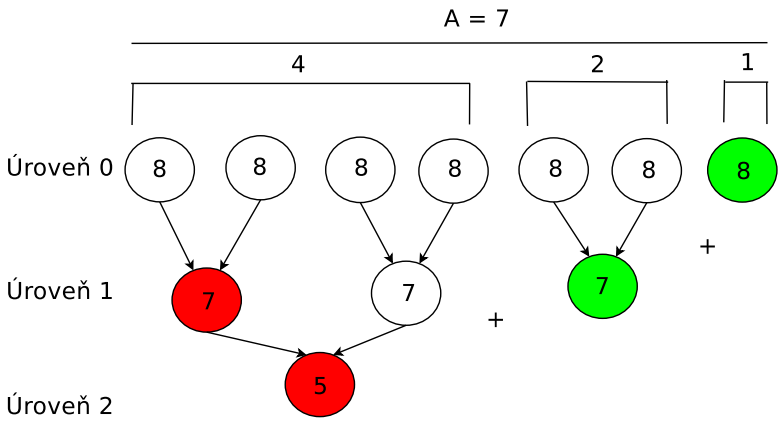
Require: $A = (a_n a_{n-1} ... a_0)_2 < M, B = (b_n b_{n-1} ... b_0)_2 < M$
Ensure: $C = A \times B \, mod \, M$

1: **if** $A = 0$ OR $B = 0$ **then**
2: **return** $C = 0$
3: **end if**
4: **if** $A > B$ **then**
5: SWAP(A, B)
6: **end if**
7: $rtree = B$,$ltree = 0$
8: $level = 0$, $r = 0$
9: **if** $A > 1$ **then**
10: **if** $a_0 = 1$ **then**
11: $A = A - 1$
12: $ltree = B$
13: **end if**
14: $level$ =LENGTH(A)
15: $r = A - 2^{level}$
16: **end if**
17: **for** $i = 0$ to $level$ **do**
18: $rtree = 2 \times rtree$
19: **if** $rtree \geq M$ **then**
20: $rtree = rtree - M$
21: **end if**
22: **if** $i \neq level - 1$ **then**
23: $exponent = 2^{level}$
24: **end if**
25: **if** $r_i = exponent$ **then**
26: $ltree = ltree + rtree$
27: **if** $ltree \geq M$ **then**
28: $ltree = ltree - M$
29: **end if**
30: **end if**
31: **end for**
32: $C = rtree + ltree$
33: **if** $C \geq M$ **then**
34: $C = C - M$
35: **end if**
36: **return** C

Podmínkou správné funkce algoritmu 3 je velikost hodnot $A, B < M$. Pokud $A = 0$ nebo $B = 0$, pak algoritmus skončí (řádek 1 a 2). Protože je výhodnější zkonstruovat binární strom z menšího čísla (menší počet kroků), vezme se menší číslo z čísel A, B (řádek 4 a 5). Binární strom se sestaví pouze pokud je jeho výška větší než 1 (řádek 9). Protože jsou binární stromy založeny na čísle 2, je potřeba aby počet listů byl sudý (řádek 10,11 a 12). Dále se nalezne nejvyšší binární strom v sekvenci čísla B a uloží zbytek (řádek 14 a 15). Potom se začne počítat v cyklu od 0 do $log_2 A$ (řádek 17). Protože uzly ve stejné výšce mají stejné číslo, lze použít bitový posun (řádek 18). Na kořeni stromu lze sečíst jeho obsah se zbytkem, který vznikl po vedlejších stromech o nižší výšce (řádek

26). Nakonec se sečte kořen nejvyššího stromu se zbytkem po stromech nižší výšky (řádek 32). Průběh algoritmu je ukázán na následujícím příkladě:

Příklad 3.5.1. Jsou dány čísla $A = 7, B = 8$ a provede se jejich redukci *mod* 9. Nejprve se sestaví binární strom, jehož listy obsahují větší číslo z násobených čísel (výhodou tohoto přístupu je kratší binární strom a tedy méně stupňů stromu). Potom se provede redukce 2 předchůdců v každém řádu (protože všichni předchůdci obsahují stejnou hodnotu, je nutné provést pouze jednu redukci předchůdců v každém řádu). Binární strom je rozložitelný do binárních podstromů, dle jejich řádu. V každém řádu se provádí také redukce kořene binárního stromu o stejném řádu a předchozího výsledku binárních stromů o nižších řádech, jak naznačuje následující obrázek. Konečný výsledek se získá součtem a redukcí předcházejících stromů o nižším řádu a kořene nejvyššího binárního stromu.



Následník (Úroveň) = Následník (Úroveň - 1) + Následník (Úroveň - 1) % M

Výsledek = ((8 + 7) mod 9) + 5) mod 9 = 2

Obrázek 3.5.1: Příklad násobení modulo čísel Kop algoritmem pro $(8 \cdot 7)$ mod 9

Výhodou navrhnutého a implementovaného algoritmu je jeho časová složitost, kdy je možné deterministicky nalézt výsledek v logaritmickém čase. Tento algoritmus splňuje několik vlastností kladených na urychlení výpočtu redukovaného součinu. Kromě logaritmické časové náročnosti využívá i bitové posuny a neobsahuje operaci násobení a dělení, tyto přednosti jsou vidět v následující tabulce, kde proběhlo srovnání rychlosti výpočtu redukčního součinu pro čísla A, B v rozsahu $\{0, M-1\}$. Velikost čísla M bylo v rozmezí od 5 do 9 bitů a množství čísel v každém měřeném případě M^2:

	M = 31 5 bitů	M = 61 6 bitů	M = 127 7 bitů	M = 509 8 bitů	M = 1021 9 bitů
Barrett: avg.(ms)	101	486	2295	44968	199250
Montgomery: avg.(ms)	101	461	2285	44621	193272
Kop: avg.(ms)	79	375	1846	37532	166885

Tabulka 3.5.1: Časy výpočtu Kop algoritmu pro $(A \cdot B)$ mod M

Z tabulky je zřejmé, že v případě Kop algoritmu je čas výpočtu nižší než při použití jiných algoritmů, používajících operace dělení a hledání multiplikativní inverze. Se zvyšujícím se počtem bitů se však časový rozdíl mezi Montgomeryho algoritmem a Kop algoritmem snižuje. Dochází k převaze rychlosti bitového posunu nad logaritmickou redukcí binárního stromu.

Binární násobení na architektuře SIMT

Binární násobení v RNS lze provést několika způsoby, v případě použití standardního algoritmu obsahuje n^2 násobení. Z tohoto důvodu jsem implementoval techniku založenou na Wallasově stromě [20]. Tato technika se skládá ze tří kroků. V prvním kroku je vynásoben každý bit prvního čísla každým bitem druhého čísla a v závislosti na jejich pozici v násobených číslech jsou vytvořeny skupiny znázorněné na obrázku (4-bitové násobení). Druhým krokem je opakovaná redukce vzniklých produktů pomocí plných sčítaček (FA) a poloviční sčítaček (HA) do stupně, ve kterém existují pouze dvojice přenášených bitů.

Třetím krokem je součet dvojic bitů pomocí CLA sčítaček.

V provedené implementaci lze považovat každou FA a HA sčítačku za mikroprocesor a každý stupeň výpočtu lze provést paralelně.

Srovnání výpočtu na architektuře GPU (SIMT) a CPU je uvedeno v následující tabulce (došlo k 10 000 spuštění násobení čísel, každé číslo mělo 500, 1000, 5000, 10000 bitů):

	500 bitů (ms)	1000 bitů (ms)	5000 bitů (ms)	10000 bitů (ms)
CPU (avg. 10 000 exec.)	1201	4102	31215	87579
GPU (avg. 10 000 exec.)	10265	51636	214562	1 032720

Tabulka 3.5.2: SIMT (GPU) násobení

Z tabulky je patrné, že implementace na GPU je cca 10× pomalejší vzhledem k CPU implementaci (důvodem jsou především pomalé přesuny dat po sběrnici mezi systémovou pamětí a pamětí na grafické kartě).

3.6 Dělení v RNS

Dělení patří v RNS k obtížným operacím, protože nelze získat GF (součin modul m_i nikdy nebude prvočíslo) a tedy ke zbytkům reprezentujícím číslo v systému RNS nemusí existovat multiplikativní inverze. Nejlépe proveditelné dělení je za situace, kdy číslo $A = (|a_1|_{m_1}, |a_2|_{m_2}, ..., |a_n|_{m_n})$ je dělitelné číslem $B = (|b_1|_{m_1}, |b_2|_{m_2}, ..., |b_n|_{m_n})$ tedy $(A \, mod \, B = 0)$, pak se provede inverze čísla B^{-1} a součin čísel $A \times B^{-1}$ je výsledkem dělení. Pokud tomu tak není, používají se následující dva typy dělení:

1. škálování - označuje se tak dělení celým číslem vyjádřeným v poziční soustavě z rozsahu M
2. celočíselné dělení dvou čísel vyjádřených v RNS

3.7 Srovnání RNS (FP RNS) a standardní reprezentace na bitové úrovni

Při použití standardní aritmetiky je nutné se zvětšujícím se bitovým slovem zvýšit i počet logických jednotek vykonávající aritmetické operace, tato situace je uvedena v následujícím obrázku 3.7.1, kdy se počet potřebných bitů po součtu čísel $1,3$ zvýšil.

	Procesor		
Bit	2	1	0
Číslo 1_2		0	1
Číslo 3_2		1	1
Přenos		1	1
Číslo $1_2 + 3_2$	1	0	0
Výsledek	1	0	0

Obrázek 3.7.1: Standardní bitové sčítání s přenosem

V případě použití RNS jsou čísla $1,3$ kódovány do dvou nezávislých binárních vektorů $1_{10} = (|1|_2|1|_3) = (|01_2|_2|01_2|_3)$, $1_{10} = (|1|_2|0|_3) = (|01_2|_2|00_2|_3)$ a oproti sériovému zpracování na jednom procesoru potřebují v tomto případě o $1/3$ menší bitové slovo a nedochází k přenosu mezi výpočetními jednotkami (obrázek 3.7.2, kde je vidět, že třetí bit není potřebný).

	Procesor ($mod\,2$)		
Bit	2	1	0
Číslo $\|1_2\|_2$		0	1
Číslo $\|3_2\|_2$		0	1
Přenos		0	1
Číslo $\|1_2 + 3_2\|_2$		1	0
Redukce $mod\,2$		0	0
Výsledek		0	0

	Procesor ($mod\,3$)		
Bit	2	1	0
Číslo $\|1_2\|_3$		0	1
Číslo $\|3_2\|_3$		0	0
Přenos		0	0
Číslo $\|1_2 + 3_2\|_3$		0	1
Redukce $mod\,3$		0	1
Výsledek		0	1

Obrázek 3.7.2: RNS bitové sčítání

Přestože se časová složitost bitových operací ve standardní aritmetice zvyšuje s délkou slova n pouze nepatrně ($O((log\,n))$, při velkém počtu operací s dlouhými bitovými slovy výhoda paralelizace výpočtu pomocí RNS roste (rychlejší výpočet, jednodušší výpočetní logika pro aritmetické operace a tedy i cena výpočtu).

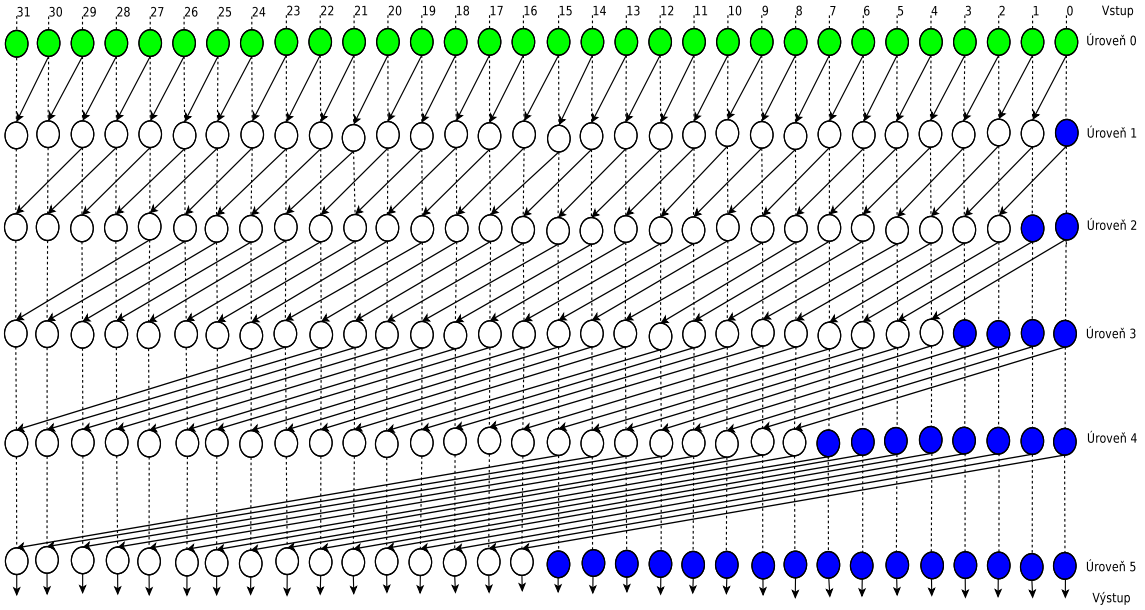
Aby došlo k maximálnímu urychlení pomocí FP RNS je nutné se držet následujícího návrhu:

- rozdělení modulů (složek vektoru RNS) na paralelním systému je rovnoměrné (tedy celý systém je efektivně vytížen),
- číslo součinu bází M je blízké maximálnímu číslo, kterého je během výpočtu dosaženo (nedochází tedy k neefektnímu využití hardwarových prostředků),
- logické obvody (sčítačky, odčítačky, násobičky) nebo algoritmy jednotlivých modulů systému FP RNS odpovídají nejrychlejším dosud používaným návrhům a implementacím.

Výše uvedené zásady jsou použity v následujícím příkladě.

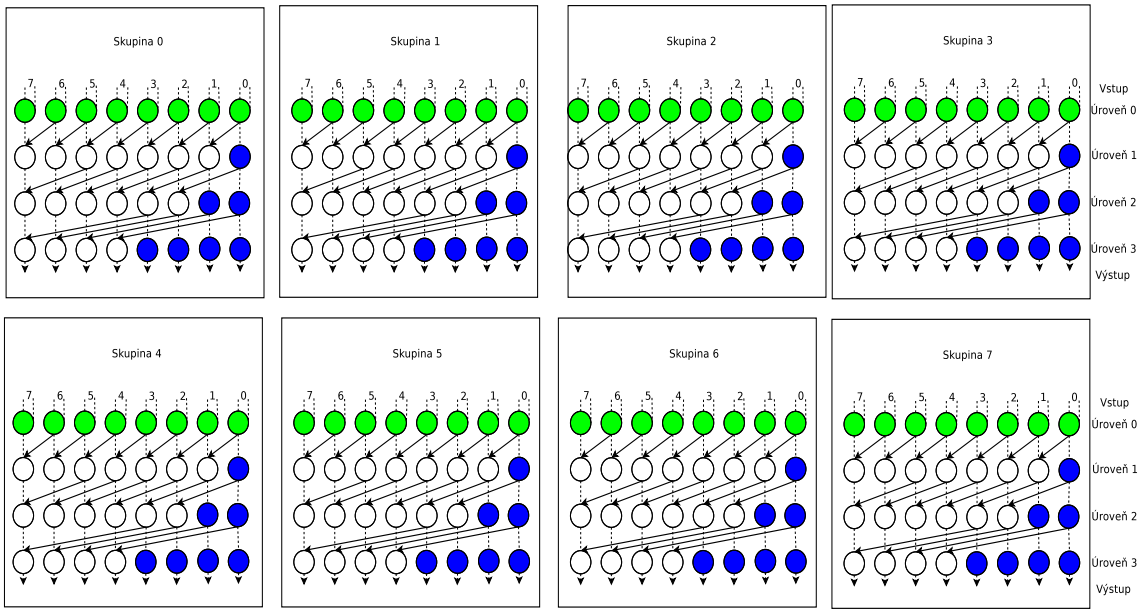
Příklad 3.7.1. Vypočte se součet dvou 32 - bitových čísel, které lze v RNS vyjádřit pomocí báze$\beta = (31,29,23,19,17,13,11,7)$, což odpovídá 8 samostatným funkčním celkům, které se budou považovat za procesory. V úvahu se bere nejlepší čas výpočtu, kdy v případě použití RNS nebude zapotřebí obvod pro modulo operaci (součty v modulech budou menší než použitá báze každého funkčního celku). Bitový součet se provede pomocí CLA sčítaček, počet stupňů sčítaček je určen funkcí $log_2 n$ (lze považovat za zpoždění), kde n představuje délku vstupu v bitech. Důležitou roli v návrhu hraje Eulerova funkce s jejíž pomocí je možno určit dostatečný počet prvočísel

požadované bitové délky (v tomto konkrétním případě jsou použity prvočísla s bitovou šířkou max 5 bitů). Při návrhu logických obvodů se vychází z Koggle - Stone sčítačky, tato sčítačka pro 32 bitů (odpovídá maximálnímu prvočíslu v bázi β) bude vypadat následovně:



Obrázek 3.7.3: 32 - bitová sčítačka Koggle - Stone

Při použití aritmetiky FP RNS se využije sčítaček o maximální délce 8 bitů, jejichž počet bude 8 (počet bází) a sčítačky pro jednotlivé moduly báze vypadají následovně:



Obrázek 3.7.4: RNS architektura složená z 8 bitových sčítaček Koggle - Stone

Porovnáním obrázku 3.7.3 a obrázku je možno vyčíst, že počet členů logického obvodu klesl o dvě úrovně, což značí urychlení 33% na každé aritmetické operaci v RNS, kdy není zapotřebí obvodu

pro modulo operaci (vzhledem ke kombinaci možných vstupů je minimálně v 50% použit zobrazený návrh) a při volbě vhodných bází lze RNS systém dále urychlit.

3.8 Urychlení výpočtu FP RNS na více - procesorovém systému

Výpočet v RNS (FP RNS) lze rozdělit do tří fází. Každá z těchto fází může být vypočtena v logaritmickém čase velikosti vstupu n. Fáze sestavení FP RNS (označení FS) je závislá na propustnosti používané paměti, neboť z čísla ve standardní reprezentaci jsou vytvořeny složky vektoru FP RNS, tedy dochází k mnohonásobnému čtení stejné adresy v paměti a lze využít vyrovnávací paměť. Fáze výpočtu (FV) není v paralelním kontextu omezena, stejné operace jsou použity pro všechny složky vektoru FP RNS a synchronizace je potřebná pouze při závěrečné fázi, kdy se z vektoru FP RNS sestaví číslo ve standardní reprezentaci. Fáze zpětné transformace (FZ) je závislá na jednotlivých složkách vektoru FP RNS a synchronizaci mezi nimi. Z těchto důvodů (třebaže ji lze provést v logaritmickém čase) je časově nejnáročnější. V následující tabulce je uvedeno srovnání výpočtu na jedno a více - procesorovém systému:

	50x FZ / 50x FV avg. (ms)	50x FZ / 100x FV avg. (ms)	50x FZ / 150x FV avg. (ms)	50x FZ / 200x FV avg. (ms)	50x FZ / 250x FV avg. (ms)
1 procesor	5761	9232	12698	16159	19733
4 procesory	5029	7734	10452	13095	15839
(1 proc. / 4 proc.)	1.146	1.194	1.215	1.234	1.246

Tabulka 3.8.1: Urychlení aritmetických operací na více-procesorovém systému RNS

Z tabulky 3.8.1 je patrné, že při zvětšujícím se počtu aritmetických operací, roste i celkové zrychlení (pro výsledky bylo použito sčítání čísel vyjádřených v FP RNS o délce 512 bitů každé báze, každé číslo se skládalo z 32 bází, testováno na AMD QUAD - Core A6 - 3420M). Tato skutečnost je dána zvyšujícím se rozdílem mezi FZ a FV operacemi, kdy začínají převažovat aritmetické operace (FV) a vliv paralelního zpracování. Pokud bude platit $FV >> FZ$, lze dosáhnout maximálního zrychlení. Tuto skutečnost splňují výpočty diferenciálních rovnic v FP RNS, které budou probírány v dalších kapitolách.

Obecně se dá říci, že se zvyšujícím se počtem procesorů řešících problém v systému FP RNS dochází k urychlení výpočtu, neboť bitová šířka (potřebná pro výpočet) jednotlivých procesorů se zkracuje a tedy roste urychlení výpočtu. Více informací o paralelních výpočtech lze najít v příloze C.

Shrnutí

Kapitola se zabývá možnostmi paralelizace RNS a implementací systému FP RNS, z tohoto důvodu jsou zde na začátku zmíněny všechny tři fáze, jimiž výpočet ve FP RNS prochází. Dále následuje samotný výpočet ve FP RNS na architekturách SIMT a MIMD. Také je zde uveden nový algoritmus pro modulo násobení založený na binárních stromech. Nakonec je zde uvedeno srovnání FP RNS a standardní reprezentace na bitové úrovni a urychlení výpočtu FP RNS na více procesorovém systému.

Kapitola 4

Přesnost aritmetiky zbytkových tříd

Dnešní výpočetní technika je založena na elektronických obvodech, jejichž podstatou je tranzistor, který může pracovat ve stavu zapnuto (ON) nebo vypnuto (OFF). Proto se z důvodů snadné přenositelnosti a škálovatelnosti používají na počítačích soustavy, které jsou násobkem zmíněných stavů (binární soustava pro ukládání čísel a výpočtů s nimi, případně hexadecimální soustava pro uložení dat). Obvykle se systémy označují podle množství bitů, se kterými jsou schopny optimálně pracovat. Nejčastěji se jedná o 32-bitové, 64-bitové a 128-bitové systémy. V této kapitole budou shrnuty současné používané systémy a soustavy společně s jejich přednostmi a nevýhodami v návaznosti na vlastnosti aritmetiky zbytkových tříd.

4.1 Reprezentace celých čísel v RNS

Aritmetika zbytkových tříd je založena na celočíselných zbytcích, proto i reprezentace čísel pomocí zbytků bude celočíselná. Výhodou této skutečnosti je to, že absolutní i relativní chyba prováděných výpočtů je rovna nule.

Rozsah intervalů hodnot RNS

Chybnost výpočtu také záleží na volbě intervalu hodnot popisujících zkoumaný problém. Při použití jedno-modulové aritmetiky je relativní chyba nulová, pokud jsou použity čísla $X \in \mathbb{N}$ z rozsahu $\{0, m-1\}$. Jsou-li čísla nebo výpočet větší než tento rozsah, je výsledek provedených operací kongruentní se správným výsledkem (avšak pro další použití nepoužitelný při přesném řešení ODR). Při jiných typech výpočtu je toto řešení jediné správné - například sčítání času.

Chceme-li použít záporná čísla, upravuje se rozsah hodnot na symetrickou množinu zbytkových tříd, kde záleží na dělitelnosti použitého modula.

- Pro lichá modula je číslo $X \in \mathbb{S}_m$ v intervalu:

$$-\frac{m-1}{2} \leq X \leq \frac{m-1}{2}.$$

- Pro sudá modula je číslo $X \in \mathbb{S}_m$v intervalu:

$$-\frac{m}{2} \leq X \leq \frac{m}{2} - 1.$$

Většinou se volí jako modulo m liché číslo vzhledem k symetričnosti intervalu k počátku souřadnic.

Protože rozsah čísel jedno-modulové aritmetiky je relativně malý, používá se více-modulová aritmetika (RNS). Tato aritmetika zaručuje izomorfismus pro všechna čísla z množiny $\{0, M-1\}$, kde číslo M je součin použitých modul m_i. Aby bylo možné provádět dělení volí se jako číslo m_i prvočíslo. V případě použití systému RNS v rekonfigurovatelném hardwaru se jeví jako užitečné upravení intervalu na zvolený rozsah hodnot.

Bitová efektivnost RNS

Vzhledem k tomu, že paměťová kapacita výpočetního zařízení bývá vždy omezená, je vhodné také porovnávat efektivnost vyjádření čísel skrze RNS se standardním binárním vyjádřením.

Příklad 4.1.1. Je dán modul $RNS(7|6|5|3)$, rozsah tohoto vektoru bude:

$$M=7 \times 6 \times 5 \times 3 = 630.$$

Intervaly pro číslo X budou:

- pokud bude $X \in \mathbb{Z}_M$, pak číslo X bude nabývat hodnot z rozsahu $< 0, 629 >$
- pokud bude $X \in \mathbb{S}_M$, pak číslo X bude nabývat hodnot z rozsahu $< -314, 314 >$

Cílem příkladu je efektivní vyjádření 100 hodnot ve dvojkové soustavě ($log_2(100)$=6.6 $\doteq$ 7 cifer) za pomocí RNS na dvouprocesorovém systému (tedy každé jádro provede 2 výpočty vzhledem k vektoru délky 4). Pro náš zvolený RNS systém platí:

modula	mod 7			mod 6			mod 5			mod 3		celkem 4 modula
bity	0	1	2	0	1	2	0	1	2	0	1	celkem 11 bitů

Tabulka 4.1.1: Bitová efektivnost RNS ($m_1 = 7$, $m_2 = 6$, $m_3 = 5$, $m_4 = 3$) pro 100 hodnot

Na 11-ti paměťových buňkách je možno uložit 2^{11}=2048 hodnot v binárním vyjádření. Paměťová efektivnost tedy bude $630/2048$=31%. Můžeme také napsat, že jsme zmařili 2.7 bitu, protože platí $log_2(630) = 9.3$, tedy $9.3 - 6.6 = 2.7$. Pokud se experimentálně upraví rozsah vektoru RNS s cílem dosáhnout co největší efektivnosti při vyjádření 100 hodnot v RNS získá se:

$$M=16 \times 7=112.$$

Tedy bitový vektor bude vypadat následovně:

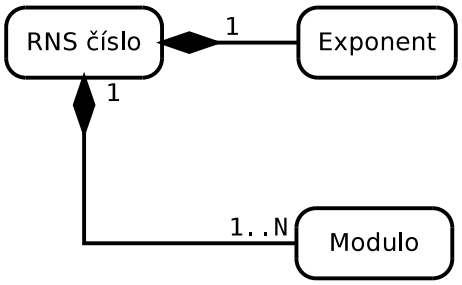
mod 16				mod 7			modula
0	1	2	3	0	1	2	celkem 7 bitů

Tabulka 4.1.2: Bitová efektivnost RNS ($m_1 = 16$, $m_2 = 7$) pro 100 hodnot

Na 8 paměťových buňkách jsme schopni uložit 2^7=128 hodnot v binárním vyjádření. Paměťová efektivnost je $112/128 = 86\%$. Tedy tento vektor je vzhledem k bitové šířce a vyjadření 100 hodnot na dvouprocesorovém systému optimální.

4.2 Reprezentace čísel s pohyblivou čárkou v FP RNS

Systém, který jsem navrhl a implementoval pracuje s pohyblivou desetinnou čárkou, tento systém jsem nazval RNS s plovoucí desetinnou čárkou (dále FP RNS). Původní návrh vycházel z článku [**?**], avšak při implementaci došlo k značnému přepracování původních myšlenek (objevily se problémy, se kterými se v teoretickém návrhu nepočítalo). Zásadním problémem bylo vyjádření čísla v plovoucí desetinné pomocí RNS. Jednou z možností řešení (ke kterému jsem se přiklonil) bylo uchovat číslo v mantise a pozici desetinné čárky v exponentu (podobné vyjádření používá IEEE-754). Grafický návrh mého vyjádření desetinného čísla je následovné:



Obrázek 4.2.1: Tvar FP RNS čísla

Při použití tohoto zápisu čísla s pohyblivou řádovou čárkou se objevuje problém s přesunem nul mezi exponentem (celé číslo) a mantisou (celé číslo vyjádřené pomocí RNS). Z těchto důvodů jsem navrhl a implementoval techniku hlídání nul v mantise, tato technika využívá rozšíření báze, která je popsána dále v kapitole. Nejprve však budou ještě uvedeny příklady aritmetických operací prováděných pomocí RNS v plovoucí čárce.

Sčítání v plovoucí čárce systému RNS

Tato operace má dvě fáze. Nejprve se získá stejný exponent u obou čísel a poté se provede samotné sčítání v RNS. Postup si ukážeme na následujícím příkladě:

Příklad 4.2.1. Jsou dána dvě čísla, číslo $A = (|2|_5|2|_7|2|_{11}) = (2)_{10}$, $EXP(A) = 0$
a číslo $B = (|4|_5|4|_7|1|_1 1) = (14,4)_{10}$, $EXP(B) = -1$. Výpočtem bude číslo $C = A + B$. Nejprve se zjistí rozdíl exponentů mezi čísly A, B :

$$EXP(C) = 0 - 1 = -1.$$

Výpočet exponentu značí, že je potřeba rozšířit číslo A, aby se získala stejná velikost mantisy pro obě čísla. Rozšíření se provede vynásobením čísla A číslem $X = (|0|_5|3|_7|10|_{11}) = (10^1)_{10}$, poté se změní exponent na $EXP(A) = -1$.

$$\begin{aligned} A &= A \times X, \\ &= (|2|_5|2|_7|2|_{11}) \times (|0|_5|3|_7|10|_{11}), \\ &= (|0|_5|6|_7|9|_{11}). \end{aligned}$$

Poté lze provést součet čísel v mantise:

$$\begin{aligned} C &= A + B, \\ &= (|0|_5|6|_7|9|_{11}) + (|4|_5|4|_7|1|_{11}), \\ &= (|4|_5|3|_7|10|_{11}). \end{aligned}$$

Výsledkem je číslo $C = (16,4)_{10}$ s exponentem $EXP(C) = -1$.

Odčítání v plovoucí čárce systému RNS

Odčítání se skládá z více kroků. Nejprve se získají stejné exponenty, dále se naleze inverzní číslo pro sčítání a následně provede součet. Celý výpočet je uveden v následujícím příkladě:

Příklad 4.2.2. Jsou dány čísla $A = (|4|_5|4|_7|1|_{11}) = (14,4)_{10}$, $EXP(A) = -1$ a $B = (|2|_5|2|_7|2|_{11}) = (2)_{10}$, $EXP(B) = 0$ a provede se výpočet $C = A - B$. V první fázi výpočtu se zjistí rozdíl exponentů:

$$EXP(C) = 0 - 1 = -1.$$

Číslo B se rozšíří číslem $X = (10^1)_{10}$:

$$\begin{aligned} B &= B \times X, \\ &= (|2|_5|2|_7|2|_{11}) \times (|0|_5|3|_7|10|_{11}), \\ &= (|0|_5|6|_7|9|_{11}). \end{aligned}$$

Poté se vypočte aditivní inverze čísla B, tedy $\bar{B}$:

$$\begin{aligned} \bar{B} &= (|m_1|_{m_1}|m_2|_{m_2}|m_3|_{m_3}) - (|b_1|_{m_1}|b_2|_{m_2}|b_3|_{m_3}), \\ &= (|5|_5|7|_7|11|_{11}) - (|0|_5|6|_7|9|_{11}), \\ &= (|0|_5|1|_7|2|_{11}). \end{aligned}$$

Poslední částí výpočtu je přičtení čísla A s číslem $\bar{B}$:

$$\begin{aligned} C &= A + \bar{B}, \\ &= (|4|_5|4|_7|1|_{11}) + (|0|_5|1|_7|2|_{11}), \\ &= (|4|_5|5|_7|3|_{11}). \end{aligned}$$

Výsledkem je tedy číslo $C = (12,4)_{10}$ s exponentem $EXP(C) = -1$.

Násobení v plovoucí čárce systému RNS

Násobení je z aritmetických operací v RNS nejjednodušší. Exponenty čísel jsou pouze sečteny a mantisy čísel vynásobeny.

Příklad 4.2.3. Jsou dány čísla $A = (|4|_5|4|_7|1|_{11}) = (14,4)_{10}$, $EXP = 0$ a $B = (|2|_5|2|_7|2|_{11}) = (2)_{10}$, $EXP = -1$. Výsledkem je číslo $C = A \times B$ a pro výpočet exponentu platí:

$$EXP(C) = 0 - 1 = -1.$$

Součin mantis čísel A a B je následující:

$$\begin{aligned} C &= A \times B, \\ &= (|4|_5|4|_7|1|_{11}) \times (|2|_5|2|_7|2|_{11}), \\ &= (|3|_5|1|_7|2|_{11}). \end{aligned}$$

Tedy výsledek samotný lze získat ve dvou krocích.

Dělení čísel v plovoucí čárce systému RNS

Tato kapitola je věnována mojí metodice, která umožňuje krátit čísla ve vyjádření RNS a převést násobky soustavy do exponentu.

Rozšiřování a krácení číslem 10^x v plovoucí čárce systému RNS

Každé číslo, kromě modul které ho reprezentují, obsahuje také redundantní modul m_E, jenž má podobu násobku čísla soustavy (v našem případě číslo 10). Tato vlastnost umožňuje říci, zda-li je dané číslo dělitelné násobkem čísla 10 tedy: $x \equiv 0 mod\, 10^i$, pokud ano lze dané číslo vydělit 10^i (čísla modul m_i a číslo m_E jsou vzájemně nesoudělná, tedy platí GCD(m_i, m_E)=1 pro všechny i) a i převést do exponentu, nakonec vydělené číslo x rozšířit o modul $m_E = 10^i$ a jeho zbytek.

Uvedenou techniku lze ukázat na následujícím příkladě:

Příklad 4.2.4. Je dáno číslo $A = (|2|_7|1|_{11}|9|_{13}) = (100)_{10}$ s exponentem $EXP(A) = 0$, dále číslo $B = (|3|_7|10|_{11}|10|_{13}) = (10)_{10}$, s exponentem $EXP(B) = 0$, nakonec redundantní modul $R = |0|_{10}$ pro každé z čísel A, B. Číslo A je tedy zapsáno jako:

$$A = (|2|_7|1|_{11}|9|_{13}|0|_{10}).$$

Protože redundantní modulo $R = 0$, zřejmě platí $A mod\, 10 = 0$ a proto je možné redukovat číslo A číslem 10, protože jsou jednotlivé moduly m_i vzájemně nesoudělné, vypočte se multiplikativní inverzi:

$$\begin{array}{rcccl} b_1^{-1} & = & |3^{-1}|_7 & = & |5|_7, \\ b_2^{-1} & = & |10^{-1}|_{11} & = & |10|_{11}, \\ b_3^{-1} & = & |10^{-1}|_{13} & = & |4|_{13}. \end{array}$$

Výsledkem dělení je číslo C s moduly :

$$\begin{array}{rcccl} c_1 & = & |2 \times 5|_7 & = & |3|_7, \\ c_2 & = & |1 \times 10|_{11} & = & |10|_{11}, \\ c_3 & = & |9 \times 4|_{13} & = & |10|_{13}. \end{array}$$

V desítkové soustavě je $C = (10)_{10}$ a exponent se zvýší $EXP(C) = 0 + 1 = 1$. Nyní je potřeba použít rozšíření báze pro zisk modulu R. Následně se vypočte dynamický rozsah M a jeho inverze M^{-1}:

$$\begin{array}{rcccl} M & = & 7 \times 11 \times 13 & = & 1001, \\ |M^{-1}|_{10} & = & |1001^{-1}|_{10} & = & |1|_{10}. \end{array}$$

Pro každé M_i^{-1} platí:

$$\begin{array}{rcl} M_1^{-1} & = & |5|_7, \\ M_2^{-1} & = & |4|_{11}, \\ M_3^{-1} & = & |12|_{13}. \end{array}$$

Výpočet báze za použití rovnice je:

$$\begin{array}{rcl} \tilde{X} & = & |3 \times 143 \times 5|_{1001} + |10 \times 91 \times 4|_{1001} + |10 \times 77 \times 12|_{1001}, \\ & = & |2145 + 637 + 231|_{1001}, \\ & = & |3013|_{1001}, \\ & = & |10|_{1001}. \end{array}$$

Použitím rovnice se získá:

$$q = |1 \times (|10|_{10} - 0)| \ .$$

Po dosazení vznikne:

$$\begin{aligned} |X|_{10} &= |10 - 0 \times 1001|_{10}, \\ &= 0. \end{aligned}$$

Výsledkem úprav je tedy vektor $C = (|3|_7|10|_{11}|10|_{13}|0|_{10})$, s exponentem $EXP(C) = 1$, protože $R = |0|_{10}$, proběhne redukce ještě jednou a výsledkem bude $C = (|1||1|_{11}|1|_{13}|1|_{10}) = (1)_{10}$, $EXP(C) = 2$.

Dělení v plovoucí čárce RNS.

Dělení ve zbytkových třídách je ovlivněno použitím multiplikativní inverze. Ta je závislá na použití prvočísel (případně nesoudělných čísel) ve všech modulech systému RNS. Princip dělení v plovoucí řádce spočívá v získání stejného řádu mantisy a následného provedení dělení.

Shrnutí

Kapitola se zabývá popisem systému FP RNS, který jsem implementoval a využil pro řešení ODR. Součástí kapitoly je i popis aritmetických operací (sčítání, odčítání, násobení) provedených v systému FP RNS. Vzhledem k efektivnosti výpočtu je vhodné odstranit nuly na konci čísla. Tyto nuly se transformují z mantisy do exponentu, protože však systém FP RNS je nepoziční, není tato operace triviální a využil jsem k tomu metodiku založenou na rozšíření čísla RNS, kdy je přidáno k číslům RNS redundantní modulo. Tato metodika je popsána v poslední části kapitoly.

Kapitola 5

Využití aritmetiky zbytkových tříd pro výpočet ODR

V této kapitole navazuji na semi - analytické výpočty ODR pomocí metody Taylorovy řady a uvádím svůj výzkum výpočtu ODR v RNS.

5.1 Volba integračního kroku h

Při numerických výpočtech na počítačích se pracuje s konečnými čísly, vyskytne-li se během výpočtu číslo s nekonečným desetinným rozvojem, je toto číslo zaokrouhleno dle implementované normy (většinou IEEE - 754). V následujících odstavcích jsou uvedeny definice, objasňujícími pojem konečné číslo.

Definice 5.1.1. Je dána množina M, která je podmnožinou $\mathbb{Q}$, tato množina je uspořádaná a prvky množiny jsou čísla s maximální cifernou délkou n. Tato množina se bude označovat jako FIN.

Důsledek. *Množina FIN je uzavřená vzhledem k aritmetickým operacím sčítání, odčítání. Prvky množiny jsou i čísla s maximální délkou n, které se získaly aritmetickými operacemi násobení a dělení*

Definice 5.1.2. Konečná podmnožina $Fin \subseteq FIN$, obsahuje prvky x množiny FIN, pro něž platí $A \leq x \leq B$, kde prvek A je supremem množiny Fin a prvek B infimem množiny Fin.

Ve výzkumu jsem se zabýval velikostí kroku h, který je prvkem množiny Fin. Odpovídá - li kroku h numerické metody číslo $x_1 \in Fin$, pak výsledkem aritmetické operace bude číslo $x_2 \in Fin$. Vzhledem k vlastnostem množiny bude zaokrouhlovací chyba provedených aritmetických operací nulová. K výpočtu velikosti kroku h, jsem sestavil následující algoritmus, vycházející z určení nejmenšího společného násobku (LCM):

Algoritmus 4 Výpočet velikosti kroku h

Require: všechna čísla x ve výrazech x/h zvolené numerické metody, jejíž počet je n, krok h je velikosti10^{-p}, maximální číslo aritmetiky L

Ensure: krok h

1: **for all** x **do**
2: $f \leftarrow 1/x$
3: **if** $f > L$ **then**
4: $field[] \leftarrow x$
5: **end if**
6: **end for**
7: $lcm \leftarrow LCM(field[])$
8: $h \leftarrow lcm \times 10^{-p}$

Tento algoritmus pracuje následujícím způsobem: nejprve se určí všechna čísla x, jimiž je krok metody h dělen (řádek 1), a provede se naznačené dělení (řádek 2), pokud je výsledkem dělení číslo f (které má nekonečný rozvoj), tak se číslo zařadí do pole $field[]$ (řádek 3,4). Poté se vypočte LCM čísel obsažených v poli $field[]$ (řádek 7). Nakonec se získá velikost kroku h stanoveného řádu (řádek 8).

Důvodem pro návrh tohoto algoritmu bylo zajištění dělitelnosti kroku h v numerické metodě, což v důsledku znamená konečný výsledek kroku numerické metody bez zaokrouhlovacích chyb. Výsledky výpočtů v FP RNS, které obsahují takto upravený krok budou uvedeny v následujících odstavcích.

Řešení ODR metodou Taylorovy řady v aritmetice FP RNS

V následujícím příkladě bude řešena ODR, kde řešení je ve tvaru $y = e^t$.

Příklad 5.1.3. Zadání rovnice je ve tvaru:

$$y' = y, y(0) = 1, \tag{5.1.1}$$

a MTR je použita pro interval $t \in <0, 0.945>$(číslo 0.945 je násobek kroku h vypočteného pomocí algoritmu 4). Použije se např. řešení pomocí 10. řádu MTR obsahující 10 členů Taylorovy řady, vůči kterým je potřeba zajistit dělitelnost:

$$Y_1 = Y_0 + \sum_{i=1}^{i=10} DY_i.$$

Jednotlivé členy DY_i se vypočítají rekurentním způsobem:

$$DY_i = \frac{h}{i} DY_{i-1}.$$

Podle uvedeného algoritmu 4 se vypočte číslo 630, které je LCM pro všech 10 členů Taylorovy řady obsahující podíly h/i. Na základě volby řádu $p = -3$, které bylo zvoleno záměrně z důvodu demonstrace přesnosti výpočtu FP RNS, se zvolí velikost kroku $h = 0.063$. Důsledkem volby tohoto kroku je, že se odstranění zaokrouhlovací chyby při použití dostatečně velké aritmetiky. Řešením ODR je $y = e^t$. Hodnota této funkce v jednotlivých uzlech numerické metody je vypočtena pomocí programu MAPLE s přesností na 20 desetinných míst. Takto vypočtená hodnota funkce je považována za analytickou, tedy relativní chyba výpočtu $\delta(y(\tilde{t})) = 0$ a následně porovnána s numerickým

řešením získaným mojí implementací MTR v aritmetice FP RNS a standardním řešením získaném v programovacím jazyce C++ s přesností *double* (pro program MATLAB platilo také $\delta(y(\tilde{t})) = 0$). Výsledky jsou uvedeny v tabulce 5.1.1. Tabulka je složena ze třech sloupců, v prvním sloupci je uvedena hodnota uzlu t, pro který byla použita MTR, v druhém sloupci je uvedena aritmetika programu použitá pro výpočet a ve třetím sloupci je uvedena relativní chyba výpočtu vzhledem k referenčnímu řešení. Z tabulky je zřejmé, že v případě použití FP RNS aritmetiky nedochází k relativním chybám vůči aritmetice používané programem MAPLE (20 desetinných míst). V případě použití normy IEEE -754 (C++) je možné si všimnout zvyšující se relativní chyby výpočtu, což je dáno zaokrouhlením aritmetiky založené na IEEE -754 . Např. Pro čas (uzel) $t = 0.063$ je relativní chyba hodnoty vypočtené pomocí programu MAPLE (dosazením do řešení $y = e^t$) a hodnoty vypočtené numericky (MTR 10. řádu) v FP RNS nulová. V případě C++ je relativní chyba výpočtu $6.037407E-17$.

t	program	$\delta(y(\tilde{t}))$	t	program	$\delta(y(\tilde{t}))$
0	FP RNS (num.)	0	0.504	MAPLE (anal.)	0
	C++ (double) (num.)	0		C++ (double) (num.)	3.744874E-16
0.063	FP RNS (num.)	0	0.567	FP RNS (num.)	0
	C++ (double) (num.)	6.037407E-17		C++ (double) (num.)	3.339818E-16
0.126	FP RNS (num.)	0	0.630	FP RNS (num.)	0
	C++ (double) (num.)	1.0929379E-16		C++ (double) (num.)	1.957807E-16
0.189	FP RNS (num.)	0	0.693	FP RNS (num.)	0
	C++ (double) (num.)	3.403527E-16		C++ (double) (num.)	3.493014E-16
0.252	FP RNS (num.)	0	0.756	FP RNS (num.)	0
	C++ (double) (num.)	4.58574E-16		C++ (double) (num.)	4.519331E-16
0.315	FP RNS (num.)	0	0.819	FP RNS (num.)	0
	C++ (double) (num.)	2.268914E-16		C++ (double) (num.)	4.790959E-16
0.378	FP RNS (num.)	0	0.882	FP RNS (num.)	0
	C++ (double) (num.)	2.95334E-16		C++ (double) (num.)	6.028829E-16
0.441	FP RNS (num.)	0	0.945	FP RNS (num.)	0
	C++ (double) (num.)	3.083138E-16		C++ (double) (num.)	5.39876E-16

Tabulka 5.1.1: Srovanani chyb aritmetiky IEEE - 754 a FP RNS, řešení: $y = y'$, $y(0) = 1$, $h = 0.063$, MTR 10. řádu

Při výpočtu pomocí MTR ve FP RNS bylo pro řešení v posledním řádku tabulky ($t = 0.945$) použito 578 cifer a 12 prvočísel s délkou 256 bitů. Celkového počtu 12-ti prvočísel se užilo záměrně vzhledem k snadnému přerozdělení výpočtu na sudý počet jader procesoru (12-ti jádrový procesor, dva 6-ti jádrové procesory, atd.).

Srovnáním relativních chyb z tabulky 5.1.1 je patrné, že typ *double* postrádá potřebnou přesnost pro tento typ výpočtu (přesnost výpočtu v typu *double* je 14 desetinných míst). V případě použití FP RNS a metody Taylorovy řady 10. řádu se získá přesné řešení (bez zaokrouhlování) na 20 desetinných míst. Přesné výsledky lze nalézt v příloze práce.

Při použití MTR 20. řádu je situace více názornější. Následující tabulka 5.1.2 zobrazuje výsledek je pro uzel $t = 1.01846745$ a stejnou ODR (5.1.1):

t	program	$y(t)$	$\delta(y(\tilde{t}))$
1.01846745	MAPLE	2.76894795938017628745610642465885822093502076194 48...	0
	FP RNS	2.76894795938017628745610642465885822093502076194 48...	0
	TKSL C	2.76894795938017628745610642465885822092**97062936546**...	1.91930E-39
	MATLAB	2.76894795938017**6015997676586266607046127319335937 5**...	9.80366E-17
	C++	2.76894795938017**340000000000000000000000000000000000**...	1.04279E-15

Tabulka 5.1.2: Srovnání chyb aritmetik v bodě t=1.0184674, $y' = y$, $y(0) = 1$, $h = 0.02909907$, MTR 20. řádu

Výsledek ODR je získán znovu analyticky ($y = e^t$) pomocí programu MAPLE a shoduje se s numerickým řešením vypočítaného pomocí MTR v aritmetice FP RNS až do 51. desetinného čísla (relativní chyba výpočtu je do 51 místa nulová). Pro výpočet jsem použil 50 prvočísel, každé o délce 384 bitů a výsledek pro t zabírá celkem 6231 desetinných cifer a při numerickém řešení se vycházelo z bodu $t = 0$.

V tabulce se také nachází numerické řešení pomocí TKSL C, kdy byla použita přesnost mantisy 1000 bitů a proměnný řád metody $ORD = 50$. Výsledek byl získán na 60 desetinných míst a shoduje se s programem MAPLE do 38 desetinného místa.

Jednoznačně výsledek s horší přesností generuje program MATLAB, kde pro výpočet byla zvolena interní funkce programu a výsledek je správný pouze do 15 desetinného místa (důvodem tohoto stavu je špatná vnitřní implementace funkce *exp* v programu MATLAB 15).

V případě použití standardního typu *double* (C++) se získá přesné numerické řešení na 14 desetinných místech. Relativní chyba výpočtu je tedy příliš vysoká a pro většinu přesných výpočtů nedostačuje.

Výsledky v tabulce 5.1.2 tedy ukazují převahu MTR implementované v aritmetice FP RNS, především z důvodu odstranění zaokrouhlovacích chyb aritmetiky, které se postupně šíří numerickým výpočtem v jiných aritmetikách.

Řešení ODR Eulerovou metodou v aritmetice FP RNS

Při použití této metody není potřeba úprav kroku h pro výpočet bez zaokrouhlovacích chyb ve vhodné aritmetice, protože krok h není během iteračního výpočtu dělen a tedy lze použít libovolnou velikost kroku v celé oblasti stability metody.

Příklad 5.1.4. Nalezení řešení diferenciální rovnice $y' = t + 2y$, kde $y(0) = 0, t = 0$, h=0.25. Nejprve se iterační výpočet přepíše do tvaru:

$$y_{i+1} = y_i + h \cdot (t_i + 2 \cdot y_i),$$

kde $t_i = t_0 + i \cdot h$. Vzhledem k celistvosti je zde uvedeno také analytické řešení ODR:

$$y = -\frac{1}{4} - \frac{1}{2}t + \frac{1}{4}e^{2t}.$$

Numerické řešení Eulerovou metodou bez zaokrouhlovacích chyb ve standardní aritmetice (IEEE - 754) je uvedeno v následující tabulce (první sloupec značí kroky n, druhý uzly t, třetí hodnoty získané iteračním výpočtem $y(t)$):

n	t	$y(t)$
0	0	0
1	0.25	0
2	0.5	0.0625
3	0.75	0.21875
4	1	0.515625

Tabulka 5.1.3: Aritmetika IEEE - 754, řešení: $y' = t + 2y$,$y(0) = 0$, $h = 0.25$, Eulerova met.

Protože se při výpočtu v IEEE - 754 použili čísla, kde jmenovatel je mocninou čísla 2, nedošlo k žádnému zaokrouhlení výpočtu. Dále je uvedeno řešení rovnice v FP RNS s vektorem modul $< 47,53,59,61 >$, výsledek $y(1) = 0.515625$ je totožný s výpočtem ve IEEE - 754 (první sloupec značí kroky n, druhý sloupec značí vektor modul aritmetiky FP RNS, třetí sloupec značí exponent v FP RNS, ve čtvrtém sloupci je hodnota ve standardním vyjádření, v pátém sloupci je výpočet numerické metody v FP RNS, v šestém exponent výpočtu v FP RNS a v sedmém výpočet ve standardním vyjádření).

n	$(t)_{RNS}$	$(t^n)_{RNS}$	$(t)_{double}$	$(y(t))_{RNS}$	$(y(t^n))_{RNS}$	$(y(t))_{double}$
0	<0,0,0,0>	0	0	<0,0,0,0>	0	0
1	<25,25,25,25>	-2	0.25	<0,0,0,0>	0	0
2	<5,5,5,5>	-1	0.5	<14,42,35,15>	-4	0.0625
3	<28,22,16,14>	-2	0.75	<20,39,45,37>	-5	0.21875
4	<1,1,1,1>	0	1	<35,41,24,53>	-6	0.515625

Tabulka 5.1.4: Aritmetika FP RNS, řešení: $y' = t + 2y$,$y(0) = 0$, $h = 0.25$, Eulerova met.

Nepřesnost výpočtu (IEEE - 754) nastává při použití čísla, které nemá v binárním vyjádření ekvivalent. Tato situace je demonstrována v následujícím příkladě, kdy je použita velikost kroku h=0,3 (Po převodu čísla do IEEE - 754 se při využití 64 bitů získá číslo 0.29999999999999999). Bohužel s tímto číslem se interně dále pokračuje a chyba vlivem špatné reprezentace čísla narůstá (obvykle dochází k zaokrouhlení k nejvyššímu číslu při tisku výsledku, to ale nemá vliv na interní reprezentaci čísla). Následující tabulka ukazuje vzrůstající zaokrouhlovací chybu a skládá se z pěti sloupců, v prvním sloupci je krok n, ve druhém uzel t, ve třetím relativní chyba uzlu, ve čtvrtém vypočtená hodnota a v pátém její relativní chyba.

n	t	$\delta(\tilde{t})$	$y(t)$	$\delta(\tilde{y(t)})$
0	0.00000000000000000	0	0.000000000000000000	0
1	0.29999999999999999	3.3330E-17	0.089999999999999997	3.3333E-17
2	0.59999999999999998	3.3330E-17	0.323999999999999950	1.5432E-16
3	0.89999999999999991	1.0000E-16	0.788399999999999880	1.5221E-16

Tabulka 5.1.5: Chyby aritmetiky IEEE - 754, řešení: $y' = t + 2y$, $y(0) = 0$, h =0.3, Eulerova met.

Při výpočtech v FP RNS se využívá celočíselné aritmetiky a k těmto zaokrouhlovacím chybám nedochází, což je vidět v následující tabulce (první sloupec obsahuje krok, druhý hodnotu v FP RNS, třetí exponent v FP RNS, čtvrtý standardní reprezentaci, pátý ukazuje relativní chybu zaokrouhlení výpočtu, šestý vyjádření v FP RNS, sedmý exponent v FP RNS, osmý standardní representaci a devátý relativní chybu zaokrouhlení).

n	$(t)_{RNS}$	$(t^n)_{RNS}$	$(t)_{std}$	$\delta(\tilde{t})$	$(y(t))_{RNS}$	$(y(t^n))_{RNS}$	$(y(t))_{std}$	$\delta(\tilde{y(t)})$
0	<0,0,0,0>	0	0	0	<0,0,0,0>	0	0	0
1	<3,3,3,3>	-1	0.3	0	<9,9,9,9>	-2	0.09	0
2	<6,6,6,6>	-1	0.6	0	<42,6,29,19>	-3	0.324	0
3	<9,9,9,9>	-1	0.9	0	<35,40,37,15>	-4	0.7884	0

Tabulka 5.1.6: Chyby aritmetiky FP RNS, řešení: $y' = t + 2y$, $y(0) = 0$, h =0.3, Eulerova met.

V tabulce výše jsou uvedeny přesné hodnoty a jim odpovídající reprezentace v RNS vektoru $< 47,53,59,61 >$, je možné si všimnout že k žádné zaokrouhlovací chybě při výpočtu numerickou metodou nedochází.

Řešení ODR Heunovou metodou v aritmetice FP RNS

Tato metoda, podobně jako Eulerova, nepotřebuje vzhledem ke své jednoduchosti úpravu kroku h (každé číslo dělené číslem 2 má konečný rozvoj). Dříve než budou uvedeny výpočty s Heunovou metodou, se definuje následující funkce.

Definice 5.1.5. Nechť funkce $\triangle(x)$ označuje počet použitých cifer za desetinnou čárkou čísla $x \in Fin$ v desetinném vyjádření.

Definovaná funkce se použije v další části kapitoly a slouží k určení rozsahu aritmetiky FP RNS použité pro výpočet řešené ODR.

Příklad 5.1.6. Rovnice 5.1.4 se bude řešit pomocí Heunovy metody. Pro potřeby numerického výpočtu se upraví rovnice do tvaru:

$$\begin{aligned} \tilde{y}_{i+1} &= y_i + h \cdot (t_i + 2 \cdot y_i), \\ y_i &= y_i + \tfrac{h}{2}((t_i + 2 \cdot y_i) + ((t_i + 2 \cdot \tilde{y}_{i+1})), \\ t_i &= t_0 + i \cdot h. \end{aligned}$$

Takto upravená rovnice se použije pro jednotlivé kroky výpočtu. V následující tabulce jsou uvedeny výsledky (tabulka se skládá ze čtyř sloupců, v prvním sloupci je krok metody, ve druhém hodnota uzlu, ve třetím hodnota výpočtu kroku numerickou metodou, ve čtvrtém počet použitých cifer):

n	t	$y(t)$	$\triangle(y(t))$
0	0.00	0.00000000000000	0
1	0.25	0.03125000000000	5
2	0.50	0.16015625000000	8
3	0.75	0.44775390625000	11
4	1.00	0.99322509765625	14

Tabulka 5.1.7: Aritmetika IEEE - 754, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.25$, Heunova met.

Z uvedené tabulky je patrné, že přesnost typu *double* nedostačuje, v n=5 bude $\triangle(y(t)) = 17$ a dojde k zaokrouhlení výsledné hodnoty, přestože jsou použita čísla o základu 2. Při použití aritmetiky FP RNS a zvolení vhodného rozsahu k zaokrouhlení nedochází (tabulka se skládá ze čtyř sloupců, které jsou značeny obdobně, jako v předcházející tabulce):

n	t	$y(t)$	$\triangle(y(t))$
0	0.00	0.0	0
1	0.25	0.03125	5
2	0.50	0.16015625	8
3	0.75	0.44775390625	11
4	1.00	0.99322509765625	14
5	1.25	1.95774078369140625	17
6	1.50	3.60320377349853515625	20
7	1.75	6.3552061319351196289 0625	23
8	2.00	10.90533496439456939697265625	26
9	2.25	18.377419317141175270080 56640625	29
10	2.50	30.597681390354409813880 92041015625	32
11	2.70	50.533732259325915947556 49566650390625	35
12	3.00	83.007939921404613414779305 45806884765625	38

Tabulka 5.1.8: Aritmetika FP RNS, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.25$, Heunova met.

Z tabulky je patrný zvyšující se lineární počet použitých cifer $\triangle(x)$, který bude diskutován dále v této kapitole.

Řešení ODR metodou Runge-Kutta 4. řádu v aritmetice FP RNS

Tuto metodu je již nutné upravit na základě vhodné dělitelnosti algoritmem prezentovaným na začátku kapitoly 4. Zjednodušeně lze říci, že se bude hledat h takové, aby bylo dělitelné číslem 3. V tomto případě je určení vhodného h, možné i bez použití algoritmu, neboť pro číslo dělitelné číslem 3 platí, že pokud jeho číselný součet je dělitelný číslem 3, je dané číslo dělitelné 3.

Příklad 5.1.7. Bude se řešit ODR z předešlého příkladu 5.1.4. Dle metody Runge - Kutta se výpočet upraví do tvaru:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{1}{6}h \cdot (k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= t_i + 2 \cdot y_i, \\ k_2 &= (t_i + \tfrac{h}{2}) + 2 \cdot (y_i + \tfrac{h}{2}k_1), \\ k_3 &= (t_i + \tfrac{h}{2}) + 2 \cdot (y_i + \tfrac{h}{2}k_2), \\ k_4 &= (t_i + h) + 2 \cdot (y_i + hk_3), \\ t_i &= t_{i-1} + h. \end{aligned}$$

Krok dělitelný číslem 3, který vede k výpočtu bez vzniku zaokrouhlovacích chyb je např. $h = 0,15$ (násobek čísla 3 a řádu $p = -2$) a byl použit pro výpočet znázorněný v následující tabulce (tabulka se skládá ze třech sloupců, první označuje krok n, druhý čas uzlu t a třetí samostatný výpočet $y(t)$):

n	t	$y(t)$
0	0.00	0.000000000000000000000000
1	0.15	0.012459375000000000000000
2	0.30	0.055515319101562495000000
3	0.45	0.139871659547755380000000
4	0.60	0.279976823744793210000000
5	0.75	0.495333840821612300000000
6	0.90	0.812268630860043080000000
7	1.05	1.266316908008543500000000
8	1.20	1.905446111813982300000000
9	1.35	2.794404490955706400000000
10	1.50	4.020591659560423600000000

Tabulka 5.1.9: Aritmetika IEEE - 754 v C++, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.15$, Runge - Kutta met.

Z tabulky je však patrné, že při použití aritmetiky založené na IEEE - 754 dochází k překročení přesnosti typu *double* ve třetím řádku ($n = 3$) a dále se se výpočtem šíří zaokrouhlovací chyba.

Při použití FP RNS k zaokrouhlovacím chybám při výpočtu numerickou metodou nedochází a výsledek výpočtu je zobrazen v následující tabulce (první sloupec značí uzlový bod t, druhý vypočtenou hodnotu $y(t)$ a třetí počet platných cifer $\triangle(y(t))$):

t	$y(t)$	$\triangle(y(t))$
0.00	0.0	0
0.15	0.012459375	9
0.30	0.0555153191015625	16
0.45	0.13987165954775537109375	23
0.60	0.279976823744793240728759765625	30
0.75	0.49533384082161234608220726013183593750	37
0.90	0.81226863086004315520474144249820709228515625	44
1.05	1.26631690800854350251368017688817361593246459960937	51
1.20	1.90544611181398234007430976577029005329623818397521972656250	58
1.35	2.79440449095570638697005610845295389981626090966165065765380859375	65
1.50	4.02059165956042332012169311229386415974323208564540836960077285766601562500	72

Tabulka 5.1.10: Aritmetika FP RNS, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.15$, Runge - Kutta met.

Z výsledku (jak pro Runge - Kutta tak i pro Heunovu metodu) je patrná pravidelnost ve zvyšování počtu platných cifer výsledku $\triangle(y(t))$, linearita tohoto nárůstu se využije pro určení dynamického rozsahu čísla M.

5.2 ODR s řešením ve tvaru polynomiální funkce v aritmetice FP RNS

Rovnice lze pomocí Taylorova rozvoje v FP RNS řešit velice snadno, což lze ukázat na následujícím příkladě:

Příklad 5.2.1. Hledá se numerické řešení diferenciální rovnice: $y' = t^4$, $y(0) = 0$. Analytické řešení je ve tvaru:

$$y = \frac{t^5}{5}.$$

Při numerickém řešení se postupuje následujícím způsobem. Nejprve se daná diferenciální rovnici derivuje:

$$\begin{aligned} y' &= t^4, \\ y'' &= 4t^3, \\ y''' &= 12t^2, \\ y^{IV} &= 24t, \\ y^V &= 24. \end{aligned}$$

Poté se rekurentně zapíše následujícím způsobem:

$$y_1 = y_0 + h \cdot t^4 + h \cdot 2 \cdot t^3 + h \cdot 2 \cdot t^2 + h \cdot t + \frac{h}{5}.$$

Po dosazení hodnot $h = 1$, $y(0) = 0$, $t = 0$, se získá analytické řešení.

Tedy pomocí numerického výpočtu (při použití FP RNS a metody Taylorovy řady) lze získat výsledek ekvivalentní s analytickým řešením.

5.3 Určení velikosti dynamického rozsahu *M* pro výpočet ODR

Z předcházejících příkladů je patrné, že v případě výpočtů uvedených jednokrokových numerických metod se příbytek cifer (funkce $\triangle(y(t))$) ve výsledku vyznačuje pravidelností. Z této vlastnosti jsem vycházel při návrhu a implementaci následujícího algoritmu:

Algoritmus 5 Algoritmus pro určení přesnosti FP RNS

Require: počet cifer řešení ODR $\triangle(y_0(t)), \triangle(y_1(t)), \triangle(y_2(t))$, počet kroků n numerické metody

Ensure: počet číslic X potřebných pro určení velikosti M systému FP RNS

$w \leftarrow |\triangle(y_2(t)) - \triangle(y_1(t))|$
$v \leftarrow |\triangle(y_1(t)) - \triangle(y_0(t))|$
if $v < w$ **then**
  $t \leftarrow w$
else
  $t \leftarrow v$
end if
$X \leftarrow t \times n$

Algoritmus tedy slouží pro výpočet dynamického rozsahu FP RNS čísla M pro řešení ODR. Z počtu potřebných číslic (proměnná X) se dá určit číslo M. Algoritmus je odvozen od ciferné délky prvních třech kroků výpočtu ODR, kdy výsledkem algoritmu je rozdíl dvou následujících kroků s největší změnou počtu cifer (změny počtu cifer jsou označeny jako $\triangle(y(t))$ v tabulkách 5.1.8, 5.1.10). Tento rozdíl je dále násoben celkovým počtem kroků použité numerické metody.

Pokud se stanoví potřebná velikost přesnosti pro celkový výpočet po prvních třech krocích, odpadá testování překročení přesnosti a tedy celkový výpočet s potřebnou přesností může proběhnout deterministicky na základě výpočtu tohoto algoritmu.

5.4 Stanovení libovolné velikosti kroku h

Systém FP RNS umožňuje získat výsledek ODR bez zaokrouhlovací chyb. Tohoto stavu se dosáhne použitím kroku h, jehož velikost je stanovena algoritmem 4.

Obecně však není možné při vysokém řádu metody (např. metoda Taylorovy řady) získat přesnou hodnotu $y_i(t)$ pro stanovený uzel t_i, což je dáno dělením ve členech MTR, kde výsledkem jsou čísla s nekonečným ciferným rozvojem. Tuto skutečnost lze napravit použitím vysokého řádu metody až do určitého uzlu t_{i-1} a pro výpočet v uzlu t_i použít metodu s nižším řádem, která umožňuje dosáhnout uzlu t_i s jiným krokem h. Druhou možností je volba uzlu t řešené ODR tak, aby ležel uprostřed intervalu tvořeného hodnotami t_{i-1}a t_{i+1}, bude-li rozdíl těchto bodu dostatečně malý, lze získat hodnotu $y(t)$ z hodnot $y(t_{i-1})$ a $y(t_{i+1})$.

Shrnutí

Kapitola pojednává o přesnosti výpočtu v FP RNS. Nejprve jsem sestrojil algoritmus pro odstranění mezivýsledků, jejichž součástí je číslo s nekonečným rozvojem. Následně bylo srovnáno analytické řešení a řešení vypočtené numericky pomocí MTR, která obsahovala členy vypočtené představeným algoritmem. V další části byla konfrontována přesnost numerického výpočtu v aritmetice IEEE - 754 a přesnost dosažená pomocí upravených metod (Eulerova, Heunova, Kunge Kutta) v FP RNS. V závěru kapitoly byl popsán algoritmus, který slouží pro výpočet dynamického rozsahu FP RNS a tedy dosažení požadované přesnosti bez použití zaokrouhlených čísel.

Kapitola 6

Závěr

Předložená disertační práce se zabývá semi - analytickými výpočty obyčejných diferenciálních rovnic (ODR). Tyto výpočty se vyznačují urychlením a zpřesněním řešení vůči standardně používaným metodám a postupům výpočtu ODR. Urychlení a zpřesnění výpočtu ODR je závislé na volbě výpočetního algoritmu a také na zvolené aritmetice. Cílem mého výzkumu je aplikace semi - analytických výpočtů v aritmetice zbytkových tříd (RNS). Závěry tohoto výzkumu lze charakterizovat níže uvedenými odstavci.

6.1 Zvolený přístup k řešení

Na začátku výzkumu jsem analyzoval současné metody pro řešení ODR. Jako velmi efektivní se ukázaly symbolické výpočty, používané v programech Maple a Mathematica, založené na algebraických úpravách. Vzhledem k uzavřenosti těchto výpočetních postupů (firemní tajemství), je velmi obtížné získat jejich přesné znění a to dále modifikovat a rozšířit. Dalším omezením symbolických výpočtů je komplikovanost analytického řešení v případě složitých systémů.

Z výše popsaných důvodů jsem se při výzkumu zvýšení přesnosti výpočtu ODR soustředil na řešení pomocí numerických metod. Stanovil jsme pravidla, které zvyšují přesnost výpočtu klasickými numerickými metodami (Eulerova, Heunova, Runge - Kutta) a metodou Taylorovy řady.

Pro odstranění zaokrouhlovacích chyb (a jejich šíření vlivem iterací numerických metod) během numerických výpočtů je nutné zvolit vhodnou (bezchybnou) aritmetiku. Jako bezchybné aritmetiky lze označit aritmetiky založené na celých číslech. Z důvodů dobré paralelizace byla místo klasické celočíselné aritmetiky s velkým bitovým slovem zvolena aritmetika RNS.

6.2 Výpočty v aritmetice RNS a dosažené výsledky

Aritmetika RNS je značně odlišná od standardně používaných aritmetik (a také závislá na konečných tělesech využívajících prvočísla), proto jsem v práci **shrnul teorii a současný výzkum** v této oblasti.

Aritmetika RNS bývá implementována především v systémech pro zpracování signálu a systémech pro detekci chyb. Implementace RNS jsou založeny na celočíselných operacích, zahrnujících sčítání, odčítání, redukci a v ojedinělých případech i dělení. Operace násobení společně s redukcí je časově náročná, proto byly navrhnuty techniky pro její urychlení (Motgomeryho algoritmus (MA),

Barrettův algoritmus (BA)). Během výzkumu jsem **navrhl a implementoval algoritmus pro násobení s redukcí, založený na binárních stromech** (logaritmická časová složitost). Tento algoritmus nepoužívá dělení a oproti BA nepotřebuje předvýpočet, nebo výpočet multiplikativní inverze jako v případě MA.

Urychlení numerického výpočtu jsem prováděl pomocí paralelizace v RNS. Z důvodu výzkumu volby vhodné výpočetní architektury pro RNS jsem **provedl implementací RNS na SIMT architektuře**. Operaci bitového sčítání (Kogge - Stone) a násobení (Wallace tree) jsem převedl do datově-paralelní podoby, založené na specifikaci OpenCL. Tento způsob výpočtu na grafické kartě se však ukázal jako neefektivní, což bylo dáno především latencí použitých pamětí, složitostí výpočetních algoritmů spouštěných na jednoduchých proudových procesorech a nemožností provádět aritmetické operace nad jednotlivými bity. Následně jsem **provedl implementaci RNS na architektuře MIMD**, což se ukázalo být dobrou volbou. Tato architektura poskytovala dostatečně rychlou paměť, schopnost efektivně provádět komplikovanější výpočty, možnost práce s jednotlivými bity.

6.3 Výpočty ODR v FP RNS

Protože RNS je celočíselnou aritmetikou, která neumožňuje práci s desetinným číslem, **sestrojil jsem pro výpočet ODR aritmetiku s mantisou, označenou jako FP RNS**. Tato hybridní aritmetika vychází z dostupných teoretických podkladů, ale během mé implementace se ukázaly podklady jako nedostatečné, proto jsem **původní teoretický návrh FP RNS přepracoval a doplnil**. Literatura o podobných implementacích není veřejně dostupná, a proto jsem v práci **popsal implementační problémy a jejich řešení** (především se jednalo o převody mezi mantisou (celočíselná RNS) a exponentem (celé číslo), volba vhodných základů soustavy, dynamická změna velikosti aritmetiky).

Poslední část prezentované práce se týká numerických výpočtů ODR v FP RNS, které neobsahují zaokrouhlovací chyby aritmetiky (výpočet je závislý pouze na stabilitě a chybě metody) a jsou paralelizovány. Aby bylo dosaženo absolutní přesnosti výpočtu, bylo nutné odstranit příčiny vzniku čísel s nekonečným rozvojem. Z tohoto důvodu jsem **navrhl a upravil integrační metody** (Euler, Heun, Runge-Kutta, Taylor) a **stanovil pravidla pro velikost integračního kroku** h. V poslední části práce se zabývám možnostmi dalšího přizpůsobení metod, kdy je možné volit jakoukoliv velikost integračního kroku a získat tak přesný výpočet ODR ve zvoleném bodě t.

Na závěr práce **uvádím možnosti rozšíření implementovaného paralelního FP RNS** systému pomocí vysokých prvočísel (vyšší přesnost) nebo prvočísel speciálního tvaru (OEF) umožňující bitové posuny (vyšší rychlost).

6.4 Možnosti dalšího výzkumu

Následující výzkum je možné zaměřit na další urychlení a zpřesnění výpočtu ve FP RNS. Urychlení je možné provést pomocí použití speciálních prvočísel OEF, která budou základem systému FP RNS, tyto prvočísla umožňují provádět aritmetické operace pouze na základě bitových posunů. Zpřesnění výpočtu je možné provést použitím vysokých prvočísel, které se dají nalézt v databázích nebo využitím dostupných algoritmů pro jejich hledání.

Literatura

[1] Aiken, H. H.; W.Semon: Advanced Digital Computer Logic. Technická Zpráva WADC TR 59-472, Cambridge, MA, 1959.

[2] Barrett, P.: Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. Springer Berlin Heidelberg, 1987, s. 311–323.

[3] Bartsch, H.-J.: *Matematické vzorce*. 4. vydání, Academia, 2006, ISBN 80-200-1448-9.

[4] Elleithy, K. M.; Bayoum, M. A.: A $\theta(1)$ algorithm for modulo addition. IEEE Computer Society, 1990, s. 628–631.

[5] Garrett, P.: Finite fields. [online], [cit. 2008-12-13].
URL <http://www.math.umn.edu/~garrett/m/algebra/notes/09.pdf>

[6] Hairer, E.; Wanner, G.: *Solving Ordinary Differential Equations II*. second revised ed. with 137 Figures, vol. Stiff and Differential-Algebraic Problems. Springer-Verlag Berlin Heidelberg, 2002, ISBN 3-540-60452-9.

[7] Halin, H. J.: The applicability of Taylor series methods in simulation. In *Proceedings of the 1983 Summer Computer Simulation Conference*, 1983, ISBN 0-444-86716-3, s. 1032–1070.

[8] Ikenaga, B.: The Chinese Remainder Theorem. [online], [cit. 2008-12-13].
URL <http://www.millersville.edu/~bikenaga/number-theory/chinese-remainder/chinese-remainder.pdf>

[9] Jungnickel, D.: *Finite fields: Structure and arithmetics*. B.I. Wissenschaftsverlag, 1993, ISBN 34-11161-11-6.

[10] Kunovský, J.: *Modern Taylor Series Method*. Dizertační práce, 1994.

[11] Kunovský, J.: High Performance Computing. [online], [cit. 2008-11-28].
URL <http://www.fit.vutbr.cz/~kunovsky/TKSL/index.html.en>

[12] l. Montgomery, P.: Modular Multiplication Without Trial Division. American Mathematical Society, 1985, s. 519–521.

[13] Neunhöffer, M.: Finite fields. [online], [cit. 2013-06-15].
URL <http://www-groups.mcs.st-and.ac.uk/~neunhoef/Teaching/ff/ffchap3.pdf>

[14] Omondi, A.: *Residue number systems*. Imperial College Press, 2007, ISBN 1-86094-866-9.

[15] Omondi, A.; Premkumar, B.: *Residue Number Systems - Theory and Implementation*. London: Imperial College Press, 2007, s. 193–195.

[16] R.I.Tanaka: Modular Arithmetic Techniques. Technická Zpráva 2-38-62-1A, Lockheeed Missiles and Space Co., 1962.

[17] Slotnick, D.: Modular Arithmetic Computing Techniques. Technická Zpráva ASD -TDR-63-280, Baltimore, MD, 1963.

[18] Svoboda, A.: The Numerical System of Residual Classes in Mathematical Machines. In *Proc. Congr. Int. Automa*, Madrid, Spain, 1958.

[19] Szabo, N.; Tanaka, R.: *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, New York, 1967.

[20] Wallace, C. S.: A suggestion for a fast multiplier, IEEE Trans. on Electronic Comp. *IEEE Transactions on Computers*, 1964.

[21] WWW stránky: Symbolic computation. [online], [cit. 2008-11-28].
URL <http://en.wikipedia.org/wiki/Symbolic_computation>

[22] WWW stránky: Semianalytic. [online], [cit. 2008-12-13].
URL <http://mathworld.wolfram.com/Semianalytic.html>

[23] WWW stránky: Polynomials over a field. [online], [cit. 2013-06-11].
URL <http://www.numbertheory.org/courses/MP274/poly.pdf>

[24] WWW stránky: Bézout's Lemma. [online], [cit. 2013-8-25].
URL <http://www.proofwiki.org/wiki/BÃl'zout's_Lemma>

[25] WWW stránky: Euler Phi Function of Integer. [online], [cit. 2013-8-25].
URL <http://www.proofwiki.org/wiki/Euler_Phi_Function_of_Integer#Alternative_Formulation>

[26] WWW stránky: Symbolic Differentiation. [online], [cit. 2013-8-25].
URL <http://www.billthelizard.com/2012/04/sicp-256-258-symbolic-differentiation.html>

Přehled publikací autora

[27] Šátek, V., Kunovský, J., Kopřiva, J.: Advanced Stiff Systems Detection, Proceedings of the Eleventh International Scientific Conference on Informatics, Rožňava, FEI TU v Košiciach, SK. 2011, ISBN 978-80-89284-94-8, s. 208–212.

[28] Šátek, V., Kunovský, J., Kopřiva, J.: Advanced Stiff Systems Detection, Acta Electrotechnica et Informatica, SK. 2012, ISSN 1335-8243, s. 66–71.

[29] Drozdová, M., Kopřiva, J., Kunovský, J., Pindryč, M.: Methodology of the Taylor Series Based Computations, Proceedings of Third Asia International Conference on Modelling and Simulation, ID, IEEE CS, Bandung/Bali. 2009, ISBN 978-0-7695-3648-4, s. 206–211.

[30] Kaluža, V., Kopřiva, J., Kunovský, J: Partial Differential Equations in TKSL, Proceedings of CSE 2008, TU v Košiciach ,elfa, MARQ, Košice, SK. 2008, ISBN 978-80-8086-092-9, s. 312–319.

[31] Kaluža, V., Kopřiva, J., Kunovský, J., Sehnalová, P.: Using Differential Equations in Electrical Circuits Simulation, Proceedings of 42nd Spring International Conference MOSIS '08, MARQ, Ostrava, CZ. 2008, ISBN 978-80-86840-40-6, s. 150–154.

[32] Kaluža, V., Kunovský, J., Sehnalová, P., Kopřiva, J: Technical Initial Problems and Automatic Transformationion, 2009 International Conference on Computational Intelligence, Modelling and Simulation, IEEE CS, Brno, CZ. 2009, ISBN 978-0-7695-3795-5, s. 75–80.

[33] Kopřiva, J., Kraus, M.: Application of the Modern Taylor Series Method to a Multi-Torsion Chain ,Proceedings of the 7th EUROSIM Congress on Modelling and Simulation, VCVUT, Praha, CZ. 2010, ISBN 978-80-01-04589-3, s. 7–8.

[34] Kopřiva, J., Kunovský, J., Šátek, V., Drozdová, M. : Parallel Computations Based on Automatic Transformation of Ordinary Differential Equations, Proceedings of the 11th International Conference of Numerical Analysis and Applied Mathematics, American Institute of Physics, Rhodes, GR. 2013, ISBN 978-0-7354-1184-5, s. 2293–2296.

[35] Kopřiva, J., Kunovský, J., Drozdová, M. : Parallel system based on the RNS, The Proceedings of the 12th Conference Informatics'2013, University of Technology Košice, Košice, SK. 2013, ISBN 978-80-8143-127-2, s. 323–328.

[36] Kopřiva, J., Kunovský, J., Kocina, F. : Numerical integration in the RNS, The Proceedings of the 12th Conference Informatics'2013, University of Technology Košice, Košice, SK. 2013, ISBN 978-80-8143-127-2, s. 318–322.

[37] Kunovský, J., Šátek, V., Kraus, M., Kopřiva, J.: Semi-analytical Computations Based on TKSL, Second UKSIM European Symposium on Computer Modeling and Simulation, IEEE CS, Liverpool, GB. 2008, ISBN 978-0-7695-3325-4, s. 159–164.

[38] Kunovský, J., Šátek, V., Kraus, M., Kopřiva, J.: Automatic Method Order Settings, Proceedings of Eleventh International Conference on Computer Modelling and Simulation, IEEE CS,Cambridge, GB. 2009, ISBN 978-0-7695-3593-7, s. 117–122.

[39] Sehnalová, P., Kunovský, J., Kaluža, V., Kopřiva, J.: Using Differential Equations in Electrical Circuits Simulation, International Journal of Autonomic Computing, London, GB. 2009, ISSN 1741-8569, s. 192–201.

Životopis

Osobní informace

Jméno: Jan Kopřiva

Datum narození: 30.11.1982

Adresa: nám. Svobody 725, 691 23 Pohořelice

E-mail: koprivajan@gmail.com

Vzdělání

2007-dodnes Vysoké učení technické v Brně, Fakulta informačních technologií, **Doktorský studijní program**, Výpočetní technika a informatika, Téma disertační práce: *Semi-analytické výpočty a spojitá simulace*

2001-2006 Vysoké učení technické v Brně, Fakulta informačních technologií, **Magisterský studijní obor**, Informatika a výpočetní technika, Téma diplomové práce: *Semi-analytické řešení lineárních diferenciálních rovnic*

Zaměstnání

2014-dodnes **Programátor, živnostník,** tvorba informačních systémů na zakázku

2009-dodnes **Programátor, administrátor a manager, Sdružení občanů chorvatské národnosti v ČR,** správa informačních systémů a multimediálních databází, vedení organizace

2006-2007 **Administrátor, KSZ Brno,** správa a údržba informačních systémů