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

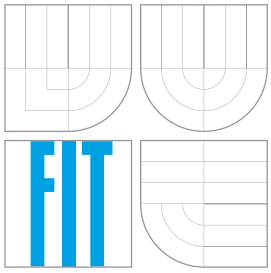
DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. JAN KOPŘIVA

BRNO 2014

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 2014

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. Jednou z možností urychlení výpočtu obyčejných diferenciálních rovnic je 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 celých čísel a jeho následnou redukci zvoleným modulem. 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

The thesis deals with speedup and accuracy of numerical computation, especially when differential equations are solved. Algorithms, which are fulling these conditions are named semi-analytical. One posibility how to accelerate computation of differential equation is paralelization. Presented paralelization is based on transformation numerical solution into residue number system, which is extended to floating point computation. A new algorithm for modulo multiplication is also proposed. As application applications in solution of differential calculus are the main goal it is dicussed numeric integration with modified Euler, Runge - Kutta and Taylor series method in residue number system. Next possibilities and extension for implemented residue number system are mentioned at the end.

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

Taylor's series method,ordinary 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, 2014

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
9. 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

1 Úvod 7
1.1 Simulace a aritmetika zbytkových tříd 7
1.2 Modelování reálných systémů 7
1.3 Motivace a cíle disertační práce 8
1.4 Struktura práce 8
2 Numerické metody a chyby matematických výpočtů 10
2.1 Druhy chyb 10
2.2 Absolutní a relativní chyba 10
2.3 Úvod do numerických metod 11
2.4 Zaokrouhlovací chyby numerických metod 12
2.5 Diskretizační chyby numerických metod 12
2.6 Stabilita numerických metod 13
2.7 Taylorova řada 16
2.8 Metoda Taylorovy řady 18
2.9 Podmíněnost úlohy 20
3 Semi - analytické výpočty ODR používané v současnosti 22
3.1 Semi - analytické výpočty ODR metodou Taylorovy řady 22
3.2 Symbolické výpočty 25
4 Teorie zbytkových tříd 31
4.1 Konečná tělesa 31
4.2 Jedno - modulová aritmetika zbytkových tříd 33
4.3 Více - modulová aritmetika zbytkových tříd 39
5 Paralelizace aritmetiky zbytkových tříd 42
5.1 Porovnávání čísel v RNS 43
5.2 Paralelní systém FP RNS 45
5.3 Převod celých čísel do RNS (FP RNS) 46
5.4 Převod čísel z RNS (FP RNS) 47
5.5 Rozšíření báze 52
5.6 Sčítání v RNS 53
5.7 Odečítání v RNS 56
5.8 Násobení v RNS 56
5.9 Dělení v RNS 61
5.10 Srovnání RNS (FP RNS) a standardní reprezentace na bitové úrovni 64
5.11 Urychlení výpočtu FP RNS na více - procesorovém systému MIMD 66
6 Přesnost aritmetiky zbytkových tříd 68
6.1 Standardní reprezentace celých čísel v počítači 68
6.2 Standardní reprezentace čísel s pohyblivou čárkou v počítači 69
6.3 Reprezentace celých čísel v RNS 70

6.4 Reprezentace čísel s pohyblivou čárkou v FP RNS . . . 72
7 Využití aritmetiky zbytkových tříd pro výpočet ODR . . . 76
7.1 Volba integračního kroku h . . . 76
7.2 ODR s řešením ve tvaru polynomiální funkce v aritmetice FP RNS . . . 83
7.3 Určení velikosti dynamického rozsahu M pro výpočet ODR . . . 84
7.4 Stanovení libovolné velikosti kroku h . . . 85
8 Možnosti rozšíření implementované aritmetiky zbytkových tříd . . . 86
8.1 Použití vysokých prvočísel . . . 86
8.2 Použití OEF modul . . . 89
8.3 Možnost rozšíření FPGA implementace metody Taylorovy řady . . . 92
9 Závěr . . . 93
9.1 Zvolený přístup k řešení . . . 93
9.2 Výpočty v aritmetice RNS a dosažené výsledky . . . 93
9.3 Výpočty ODR v FP RNS . . . 94
Literatura . . . 95
Přehled publikací autora . . . 101
A Řešené příklady . . . 103
A.1 Semi - analytické výpočty . . . 103
A.2 Využití aritmetiky zbytkových tříd pro výpočet ODR . . . 104
B Klasifikace algoritmů . . . 107
B.1 Asymptotická složitost algoritmů . . . 107
B.2 Klasifikace algoritmů podle výpočetní složitosti . . . 108
B.3 Výpočetní složitost paralelizovaných algoritmů . . . 108
B.4 Třídy problémů P, NP, RP . . . 109
C Druhy paralelních systémů a jejich výkonostní analýza . . . 111
C.1 Úvod do druhů paralelních systémů . . . 111
C.2 Flynntova taxonomie (klasifikace) paralelních systémů . . . 111
C.3 Architektura SIMT . . . 112
C.4 Architektura MIMD . . . 114
C.5 Úvod do výkonnostní analýzy paralelních systémů . . . 114
C.6 Amdahlův zákon . . . 116
C.7 Gustafsonův zákon . . . 116
C.8 Karp - Flatt metrika . . . 117
C.9 Iso - metriky . . . 118
C.10 Reálná rychlost operací prováděných počítačem . . . 121

Seznam obrázků

2.6.1 Stabilita explicitní Eulerovy metody 15
2.6.2 Stabilita implicitní Eulerovy metody 16
2.8.1 Stabilita MTR pro $ORD = 1,2,3,4$ 19
2.8.2 Stabilita MTR pro $ORD = 63$ 20

3.1.1 Hodnota funkce Z pro $ORD = 4$ (MTR) v intervalu $T \in < 0,10 >$ 24
3.1.2 Výpočet určitého integrálu pomocí MTR 25

5.2.1 Paralelní systém RNS - architektura 45
5.2.2 Paralelní systém RNS - komunikace 45
5.3.1 Převod čísla ve standartním vyjádření do RNS 46
5.6.1 Binární sčítání: Ladner - Fisher 54
5.6.2 Binární sčítání: Kogge - Stone 55
5.8.1 Příklad násobení modulo čísel Kop algoritmem pro $(8 \cdot 7) \bmod 9$ 59
5.8.2 Binární násobení: Wallace tree - rozdělení bitů do skupin 60
5.8.3 Binární násobení: Wallace tree - průběh výpočtu 61
5.10.1 Standartní bitové sčítání s přenosem 64
5.10.2 RNS bitové sčítání 64
5.10.3 32 - bitová sčítačka Koggle - Stone 65
5.10.4 RNS architektura složená z 8 bitových sčítaček Koggle - Stone 66

6.4.1 Tvar FP RNS čísla 72

C.3.1 Teoretický výkon a propustnost GPU 113
C.3.2 CPU vs GPU 113
C.4.1 Architektura MIMD 114

Seznam tabulek

4.1.1 Tabulka mocnin pro $GF(4)$ . . . 32
4.3.1 Reprezentace čísel ve zbytkových třídách a jejich systémech . . . 40

5.3.1 Převod čísla ve standartním vyjádření do RNS pomocí tabulky . . . 46
5.6.1 SIMT (GPU) sčítání . . . 55
5.8.1 Časy výpočtu Kop algoritmu pro $(A \cdot B) \bmod M$ . . . 60
5.8.2 SIMT (GPU) násobení . . . 61
5.11.1 Urychlení aritmetických operací na více-procesorovém systému RNS . . . 66

6.1.1 Ordinální datové typy . . . 68
6.2.1 Reprezentace čísel v IEEE - 754 . . . 69
6.2.2 Rozsah čísel v IEEE - 754 . . . 70
6.2.3 Přesnost čísel v IEEE 754 . . . 70
6.3.1 Bitová efektivnost RNS ($m_1 = 7$, $m_2 = 6$, $m_3 = 5$, $m_4 = 3$) pro 100 hodnot . . . 71
6.3.2 Bitová efektivnost RNS ($m_1 = 16$, $m_2 = 7$) pro 100 hodnot . . . 71

7.1.1 Srovanani chyb aritmetiky IEEE - 754 a FP RNS, řešení: $y = y'$, $y(0) = 1$, $h = 0.063$, MTR 10. řádu . . . 78
7.1.2 Srovnání chyb aritmetik v bodě t=1.0184674, $y' = y$, $y(0) = 1$, $h = 0.02909907$, MTR 20. řádu . . . 79
7.1.3 Aritmetika IEEE - 754, řešení: $y' = t + 2y$,$y(0) = 0$, $h = 0.25$, Eulerova met. . . . 80
7.1.4 Aritmetika FP RNS, řešení: $y' = t + 2y$,$y(0) = 0$, $h = 0.25$, Eulerova met. . . . 80
7.1.5 Chyby aritmetiky IEEE - 754, řešení: $y' = t + 2y$, $y(0) = 0$, h =0.3, Eulerova met. 80
7.1.6 Chyby aritmetiky FP RNS, řešení: $y' = t + 2y$, $y(0) = 0$, h =0.3, Eulerova met. . . 81
7.1.7 Aritmetika IEEE - 754, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.25$, Heunova met. . . 81
7.1.8 Aritmetika FP RNS, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.25$, Heunova met. . . . 82
7.1.9 Aritmetika IEEE - 754 v C++, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.15$, Runge - Kutta met. . . . 83
7.1.10 Aritmetika FP RNS, řešení: $y' = t + 2y$, $y(0) = 0$, $h = 0.15$, Runge - Kutta met. . 83

A.1.1 Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=0.1, $ORD = 4$, MTR . . . 103
A.1.2 Řešení: Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=0.1, $ORD = 20$, MTR . . . 104
A.1.3 Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=0.1, $ORD = 32$, MTR . . . 104
A.1.4 Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=1, $ORD = 32$, MTR . . . 104
A.2.1 Řešení: $y = y'$, $y(0) = 1$, $h = 0.063$, MTR 10. řádu (1. část) . . . 105
A.2.2 Řešení: $y' = y$, $y(0) = 1$, $h = 0.063$, MTR 10. řádu (2. část) . . . 106

C.10.1 Operace s plovoucí čárkou . . . 121

Seznam algoritmů

1 Symb. výpočty - prediktor 1 . . . 28
2 Symb. výpočty - prediktor 2 . . . 28
3 Symb. výpočty - prediktor 3 . . . 28
4 Symb. výpočty - konstruktor 1 . . . 28
5 Symb. výpočty - konstruktor 2 . . . 29
6 Symb. výpočty - prediktor 4 . . . 29
7 Symb. výpočty - selektor 1 . . . 29
8 Symb. výpočty - selektor 2 . . . 29
9 Symb. výpočty - prediktor 5 . . . 29
10 Symb. výpočty - selektor 3 . . . 29
11 Symb. výpočty - selektor 4 . . . 30
12 Symb. výpočty - derivace . . . 30
13 Euklidův algoritmus (GCD) . . . 36
14 Binární Euklidův algoritmus (BGCD) . . . 37
15 Rozšířený Euklidův algoritmus (EGCD) . . . 37
16 Algoritmus výpočtu multiplikativní inverze . . . 37
17 Binární algoritmus výpočtu multiplikativní inverze . . . 38
18 RNS sčítání . . . 53
19 RNS odečítání . . . 56
20 Barrettův algoritmus . . . 57
21 Montgomeryho produkt MP(A,B) . . . 57
22 Montgomeryho redukce MR . . . 57
23 Kop algoritmus pro RNS . . . 58
24 Algoritmus škálování . . . 62
25 Gamberger - iterace celočíselné dělení v RNS pro $GCD(Y,M) > 1$ . . . 63
26 Gamberger - iterace celočíselné dělení v RNS pro $GCD(Y,M) = 1$ . . . 63
27 Výpočet velikosti kroku h . . . 77
28 Algoritmus pro určení přesnosti FP RNS . . . 84
29 AKS algoritmus . . . 87
30 Rabin - Miller test . . . 89
31 Sčítání OEF . . . 90
32 Násobení OEF . . . 90
33 Redukce $M = 2^n - c$ v OEF . . . 91

Symboly a zkratky

$O(n)$ časová složitost

A(x) absolutní chyba

FP RNS aritmetika zbytkových tříd v plovoucí čárce

GCD největší společný dělitel

h krok numerické metody

LCM nejmenší společný násobek

MTR metoda Taylorovy řady

ODR obyčejná diferenciální rovnice

ORD řád výpočtu metody Taylorovy řady

R(x) relativní chyba

RNS aritmetika zbytkových tříd

TKSL program pro numerickou integraci

Kapitola 1

Úvod

1.1 Simulace a aritmetika zbytkových tříd

K vytvoření prvního ucelenějšího přehledu dosud známých matematických výpočtů dochází během období antického Řecka, kdy je matematikem Euklidem (ve 4. století před Kristem) sepsána kniha *Základy* [17]. Tato kniha patří ke stěžejním vědeckým dílům lidského poznání a její studium vedlo k další významné etapě matematiky začínající v 16. století, kdy se určujícími postavami stávají Descartes (geometrie, logika), Newton a Leibniz (matematická analýza), Gauss (matematická analýza, teorie čísel) a Euler (matematická analýza, teorie grafů).

Předkládaná práce se zabývá spojitou simulací prováděnou v aritmetice zbytkových tříd (RNS) v prostředí paralelních systémů. RNS je součástí matematické disciplíny nazývané teorie čísel. Její vznik je připisován Carlu Friedrichu Gaussovi v práci *Disquisitiones Arithmeticae* [22] v roce 1801.

Rozvoj simulací je spojen s nákladností výzkumu v oborech jako je fyzika, chemie, lékařství, sociologie, ekonomie. Z tohoto důvodu se hledají efektivnější způsoby využití používaných počítačových systémů. Jednou z cest je kombinace známých analytických a numerických technik, které se označují jako semi - analytické výpočty.

1.2 Modelování reálných systémů

Reálným systémem se bude označovat systém, vyznačující se špatně předvídatelným (nelineárním) chováním. Při modelování reálných systémů se naráží na dva níže popsané problémy.

1. **Převod systému do abstraktního modelu** - reálné systémy obsahují mnoho vzorců chování a vazeb na okolí, které nelze vždy přenést do abstraktní podoby (ať už z důvodů technického, nebo mentálního) a je tedy nutné model zjednodušit. Procesu zjednodušení se říká aproximace. Pokud se během aproximace zachytí podstatné vlastnosti reálného systému, vzniklý abstraktní model se prohlásí za validní. Obecným problémem aproximace je, že musí být provedena vždy, pokud se bude zkoumat jiná jeho vlastnost. Proces aproximace je časově náročný a vzhledem ke své komplexnosti náchylný k chybám (zanedbání vlastností, jež mohou být pro analýzu systému důležité).

2. **Simulace abstraktního modelu** - po převodu do abstraktního modelu se provádí jeho simulace. Simulace přináší poznatky zpětně aplikovatelné v reálném světě. Výsledky simulací jsou také zatíženy chybou, zvláště pokud jsou zkoumané systémy špatně analyticky řešitelné

a je nutné přistoupit k numerickému řešení. V případě numerického řešení dochází k další aproximaci, která se nazývá diskretizace.

Práce se bude zabývat především druhým typem problému, kde se soustředím na možnosti snížení zaokrouhlovacích chyb numerického výpočtů.

1.3 Motivace a cíle disertační práce

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 (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 jednotlivých kapitol práce.

1.4 Struktura práce

Práce je rozdělena do 9 kapitol. Kapitoly 2 až 4 obsahují současný stav poznání, ze kterého jsem čerpal ve svém výzkumu. Kapitoly 5 až 8 obsahují cíle disertační práce v návaznosti na současný stav.

První kapitola, pojednává o počítačové simulaci a důvodech vedoucích k jejich zpřesňování a paralelizaci. Kapitola také představuje cíl disertační práce a stručné obsahy jednotlivých kapitol.

Současný stav

Druhá kapitola se zabývá chybami vyskytujícími se během výpočtu. Práce se soustřeďuje především na zaokrouhlovací chyby, které jsou důsledkem použité architektury výpočetního stroje a chyby numerické metody. Nejrozsáhlejší část kapitoly je věnována metodě Taylorovy řady (MTR), na níž jsou založeny výpočty v programu TKSL [34]. Zmíněná metoda slouží k porovnání semi - analytických výpočtů s klasickými numerickými metodami v následující kapitole.

Třetí kapitola se zabývá semi - analytickými výpočty. V kapitole je uvedeno semi - analytické řešení polynomiálních, exponenciálních a goniometrických ODR. Dále jsou zde popsány symbolické výpočty, které jsou součástí dnešních matematických programů. Tyto typy výpočtů se vyznačují algebraickými úpravami řešených problémů, což vede k jejich bezchybnému analytickému řešení.

Čtvrtá kapitola se zaměřuje na teorii aritmetiky zbytkových tříd a uvádí příklady výpočtu v této aritmetice. Je zde uvedena jedno-modulová a více-modulová aritmetika, konečná tělesa a polynomy nad tělesy.

Výzkum s ohledem na současný stav

Pátá kapitola se věnuje možnostem paralelizace více-modulové aritmetiky. Popisuje implementované algoritmy pro sčítání, odčítání a násobení na architektuře SIMT a MIMD. V případě modulo násobení je zde uveden nový algoritmus s logaritmickou časovou složitostí.

Šestá kapitola pojednává o současných normách pro výpočty s pohyblivou desetinnou čárkou. Dále popisuje implementovaný FP RNS systém a problémy, se kterými jsem se při implementaci FP RNS setkal. Také ukazuje způsob jak přesunout mocninu z matisy do exponentu.

Sedmá kapitola ukazuje výpočty ODR v aritmetice zbytkových tříd s pohyblivou desetinnou čárkou. Pro výpočet jsou použity jednokrokové metody s různou mírou přesnosti. Dále jsou zde stanoveny pravidla pro volbu kroku numerické metody, tak aby číselný rozvoj výsledku byl konečný a bez zaokrouhlení.

Osmá kapitola naznačuje další možnosti rozšíření mého implementovaného systému FP RNS. Urychlení lze dosáhnout pomocí modulů speciálního tvaru a výpočty zpřesnit pomocí vysokých prvočísel.

Závěrečná devátá kapitola shrnuje dosažené výsledky, hodnotí naplnění stanovených cílů a dává podněty k dalšímu výzkumu v této oblasti.

Kapitola 2

Numerické metody a chyby matematických výpočtů

2.1 Druhy chyb

Po sestavení abstraktního modelu řešící daný problém následují experimenty s modelem (simulace). Simulace je zatížena následujícími chybami:

- **chybami architektury výpočetního systému** - patří sem především chyby zaokrouhlovací, způsobené omezenou přesností architektury,
- **chybami nezávislé na architektuře** - chyby vstupních dat, chyby numerické metody, podmíněnost řešené úlohy.

Na závěr je nutné poznamenat, že ve většině výpočtů se výše zmíněné chyby vyskytují zároveň.

2.2 Absolutní a relativní chyba

Vznik zaokrouhlovacích chyb je ve většině případů způsoben použitím výpočetních strojů, které nedokáží zachytit přesnou podobu čísla [36]. Zaokrouhlováním se označuje proces nahrazení reálného čísla x jeho aproximací $\tilde{x}$. Tato skupina chyb vzhledem k častému výskytu během numerického výpočtu bude popsána podrobněji.

Definice 2.2.1. Nechť x je přesná hodnota a $\tilde{x}$ její aproximace, pak jejich rozdíl se nazývá *absolutní chybou* aproximace $\tilde{x}$, značí se $A(\tilde{x})$ a definuje vztahem:

$$x - \tilde{x} = A(\tilde{x}).$$

Protože zpravidla není známa přesná hodnota chyb používají se odhady.

Definice 2.2.2. *Odhadem absolutní chyby* se nazývá nezáporné číslo $\varepsilon(\tilde{x})$ pro které platí:

$$|\, x - \tilde{x} \,| \leq \varepsilon\,(\tilde{x})\,.$$

V praxi je však používanější relativní relativní chyba, která je stanovena poměrem hodnot.

Definice 2.2.3. Nechť $x \neq 0$, pak *relativní chyba* aproximace $\tilde{x}$ se značí $R(\tilde{x})$ a je dána podílem absolutní chyby $A(\tilde{x})$ a čísla x:

$$R(\tilde{x}) = \frac{A(\tilde{x})}{x} = \frac{x-\tilde{x}}{x}. \tag{2.2.1}$$

Definice 2.2.4. Nechť $x \neq 0$, pak nezáporné číslo $\delta(\tilde{x})$ se nazývá *odhadem relativní chyby* a platí pro něj:

$$\frac{|x-\tilde{x}|}{|x|} \leq \delta(\tilde{x}),$$

což je totožné se vzorcem:

$$\frac{\varepsilon(\tilde{x})}{|\,x\,|} \leq \delta(\tilde{x}).$$

Aby bylo možné čísla zaokrouhlit je vhodné definovat pojem platná číslice [27] :

Definice 2.2.5. Aproximace $\tilde{x}$ čísla x má *n platných číslic*, jestliže n je největší přirozené číslo takové, že:

$$\frac{|\,x-\tilde{x}\,|}{|x|} \leq 0,5 \cdot 10^{-n}.$$

O čísle x se prohlásí, že je aritmeticky zaokrouhleno, pokud každá číslice jeho aproximace je platná.

2.3 Úvod do numerických metod

Praktické problémy bývají ve většině případů spojitého charakteru. Vzhledem ke komplexnosti a složitosti se praktický problém aproximuje a převede na numerickou úlohu. Mezi často užívané aproximace v integračním počtu se řadí obdélníková metoda, lichoběžníkovou metoda a tečnová metoda. Práce se především zabývá aproximacemi používanými v diferenčním počtu, zde se především jedná o náhradu nekonečně krátkého kroku dx (derivace) konečným krokem h. Body, kterými krok o velikosti h prochází, se nazývají uzly.

Numerické metody, diskutované v této práci se používají pro řešení diferenciálních rovnic. V případě diferenciálních rovnic vyššího řádu se dané rovnice převedou na rovnici prvního řádu a tato rovnice se pomocí numerické metody vyřeší.

Při numerickém řešení ODR je vhodné definovat počáteční úlohu.

Definice 2.3.1. Počáteční úlohou se označuje zápis:

$$\begin{array}{rcl} y'(x) & = & f(x,y(x)), \\ y(x_0) & = & a, \end{array} \tag{2.3.1}$$

kde $x \in < x_0, x_n >$, $a, x \in \mathbb{R}$.

Počáteční úloha bývá také často označována jako Cauchyho úloha.

2.4 Zaokrouhlovací chyby numerických metod

Vzhledem k nepřesnosti vyjádření reálných čísel na počítačích, dochází k výpočtům s neúplnými čísly (tedy projevují se zaokrouhlovací chyby). Obecně se rozlišují následující dva typy zaokrouhlovacích chyb:

Definice 2.4.1. *Lokální zaokrouhlovací chyba* ε_{n+1} numerického iteračního výpočtu funkce $f(x,y(x))$ na intervalu $< x_n, x_{n+1} >$je dána vztahem:

$$\tilde{y}_{n+1} = \tilde{y}_n + h\Phi f(x_n, \tilde{y}_n, h) + \varepsilon_{n+1},$$

kde $\tilde{y}_n$ označuje přibližně vypočtené hodnoty v uzlu n.

Zaokrouhlovací chyby se v jednotlivých krocích kumulují a vzniká celková zaokrouhlovací chyba.

Definice 2.4.2. *Celková zaokrouhlovací chyba* v uzlu r_n je dána rozdílem vypočtené hodnoty bez chyb y_n a se zaokrouhlovací chybou $\tilde{y}_n$:

$$r_n = \tilde{y}_n - y_n.$$

Protože velikost zaokrouhlovacích chyb ovlivňuje systém, na němž jsou výpočty řešeny, bude podrobně popsána reprezentace čísel na současných počítačích v kapitole 6.

2.5 Diskretizační chyby numerických metod

Aproximace matematického modelu numerickou metodou se celkově nazývá diskretizace a vede k diskretizačním chybám, které se dělí na globální a lokální chyby.

Definice 2.5.1. *Lokální diskretizační chybou* d_n obecné jednokrokové metody na intervalu $< x_n, x_{n+1} >$ označíme výraz:

$$d_n = y(x_{n+1}) - y(x_n) - h\Gamma(x_n, y_n, h),$$

kde vztah $h\Gamma(x_n, y_n, h)$ vychází z rekurentního vztahu výpočtu obecné jednokrokové metody:

$$y_{n+1} = y_n + h\Gamma(x_n, y_n, h),\ n = 0, 1, 2.... \tag{2.5.1}$$

Pro určení *lokální diskretizační chyby* se vychází z přesných hodnot funkce $y(x_n)$, $y(x_{n+1})$.

Tato chyba vyjadřuje nepřesnost, s níž hodnoty teoretického řešení úlohy splňují rekurentní vztah výpočtu, ze kterého je vypočtena hodnota y_{n+1} přibližného řešení 2.5.1.

Celková diskretizační chyba je tvořena lokálními diskretizačními chybami, jež vznikají v každém kroku metody. Protože v každém kroku (kromě prvního) vycházíme z nepřesných dat, není celková diskretizační chyba pouhým součtem všech lokálních chyb.

Definice 2.5.2. *Celková diskretizační chyba* e_n je dána rozdílem skutečné hodnoty v uzlovém bodě $y(x_n)$ a hodnoty získané pomocí numerické metody y_n:

$$e_n = y_n - y(x_n).$$

Celková diskretizační chyba tedy charakterizuje rozdíl hodnot mezi matematickým modelem a jeho aproximací po n krocích metody.

Kvalita aproximace přesného matematického řešení (konvergence numerické metody) se dá posuzovat také na základě řádu numerické metody. Chování lokální chyby je charakterizováno řádem numerické metody.

Definice 2.5.3. *Řádem numerické metody* budeme nazývat největší číslo $p \in \mathbb{N}$, kdy pro každý krok $h \to 0$ a libovolnou počáteční úlohu 2.3.1 splňuje metoda podmínku:

$$| d_n | \leq K \cdot h^p, \tag{2.5.2}$$

kde K je kladná konstanta.

Obecně platí, že pokud je diskretizační chyba řádu $K \cdot h^p$, pak globální chyba je řádu $K \cdot h^{p+1}$, v práci budou diskutovány jednokrokové metody splňující uvedený vztah, u metod řádu p se obvykle uvádí:

$$| e_n | \lesssim K \cdot h^{p+1}.$$

to je i případ metody Taylorovy řady s níž se bude dále pracovat.

Důležitou vlastností numerické metody, je také konvergence, tedy schopnost získat přesnější řešení při použití menšího kroku h.

Definice 2.5.4. Numerickou metodu nazveme *konvergentní*, pokud pro každé $x \in < x_0, x_n >$ a krok h platí:

$$\lim_{h \to 0} y_n = y(x_n).$$

Jinými slovy se dá říci, že numerická metoda je konvergentní, pokud se dá chyba e_n libovolně zmenšit.

Celková chyba numerického výpočtu je dána součtem celkové zaokrouhlovací chyby a celkové diskretizační chyby metody. Se správností výpočtu numerickou metodou souvisí také její stabilita.

2.6 Stabilita numerických metod

V současné době je výzkum v numerické matematice zaměřen na hledání optimálních metod řešení tuhých systémů[1]. S výběrem optimální metody pro řešený problém je spojena její stabilita. Stabilita numerické metody je ovlivněna akumulací chyb metody, a závisí na velikosti kroku h použité metody. Pro zjištění stability numerických metod řešících tuhé soustavy diferenciálních rovnic se ukázalo jako vhodné použít jednoduchou úlohu tuhého systému. Nejznámější z těchto úloh se nazývá *Dahlquistův problém* [24].

Příklad 2.6.1. *Dahlquistovým problémem* je pojmenována následující úloha:

$$y' = \lambda \cdot y, \quad y(0) = 1, \lambda < 0 \ .$$

Řešení je ve tvaru:

$$y = y(0) \cdot e^{\lambda h}. \tag{2.6.1}$$

[1] Tuhé systémy lze popsat podmíněností úlohy, jsou to úlohy, u kterých nabývá C velkých hodnot.

Testované numerické metody jsou systematicky použity pro řešení uvedeného problému a na základě výsledku po změně konstanty λ se určí jejich stabilita.

Stabilita metody je obvykle omezena velikostí kroku, přesněji vztahem:

$$|h\lambda| \leq K,$$

kde K je konstanta závislá na konkrétní metodě a řešeném problému. Pokud je označen vztah λh z rovnice 2.6.1 jako $z = \lambda h$ přepíše se metoda do tvaru:

$$y_{i+1} = z \cdot y_i.$$

Takto lze nyní nadefinovat funkci stability.

Definice 2.6.2. *Funkce stability* numerické metody pro každé y_i je dána vztahem:

$$R(z) = \frac{y_{i+1}}{y_i}, \tag{2.6.2}$$

Na základě funkce stability se definuje oblast stability:

Definice 2.6.3. *Oblast absolutní stability numerické metody* je určena vztahem:

$$S = \{z \in \mathbb{C} :| R(z) | \leq K\}.$$

Dle velikosti oblasti S se dají numerické metody rozdělit do několika tříd stability, z nichž zde budou uvedeny pouze definice A-stabilních a L-stabilních metod.

Definice 2.6.4. Numerická metoda je *A-stabilní*, pokud její oblast absolutní stability zahrnuje celou komplexní polorovinu a pro reálnou část osy platí $Re(z) < 0$, lze tedy psát:

$$S \supset \mathbb{C}^- = \{z \in \mathbb{C} : Re(z) < 0\}.$$

Protože pro řešení tuhých systémů není A-stabilita postačující podmínkou, zavedl se pojem L-stabilita numerické metody.

Definice 2.6.5. *L-stabilní* metodou nazýváme metodu, která je A-stabilní a splňuje podmínku:

$$\lim_{Re(z)\to\infty} |R(z)| = 0.$$

Mezi L-stabilní metody se řadí lineární vícekrokové metody, např. Adams – Bashforth a Adams – Moulton metody [70], a také Eulerova implicitní metoda popsaná v další kapitole [83].

10 Jednokrokové metody

Jako jednokrokové metody se označují takové metody, kdy přibližná hodnota řešení y_{i+1} v bodě t_{i+1} je vypočítaná pouze ze znalosti hodnoty přibližného řešení y_i v bodě t_i. Základní jednokrokovou metodu je Eulerova metoda.

Eulerova metoda

Tuto metodu lze odvodit z prvních dvou členů Taylorova rozvoje a třetí člen Taylorova rozvoje představu lokální chybu metody.

Definice 2.6.6. *Eulerova metoda* je výpočetní postup pro řešení diferenciální rovnice definovaný následujícím způsobem:

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

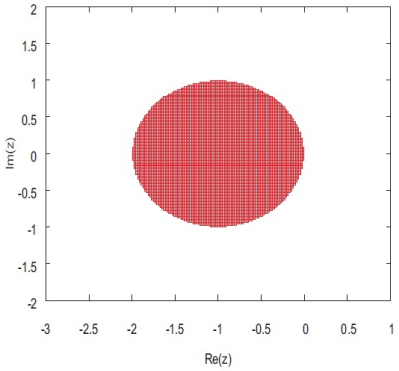
Řeší-li se touto metodou Dahlquistův problém dostane se:

$$y_{i+1} = y_i + h \cdot \lambda \cdot y_i = (1 + h \cdot \lambda) \cdot y_i = (1 + h \cdot \lambda)^{i+1} \cdot y_0.$$

Pro oblast stability metody bude platit:

$$|R(z)| = |1 + z| \leq 1.$$

Což lze v komplexní rovině znázornit následovně:



Obrázek 2.6.1: Stabilita explicitní Eulerovy metody

Z uvedeného obrázku je patrné, že stabilita této metody je velmi nízká (vyšrafovaná oblast).

Eulerova implicitní metoda

Metoda bývá často používaná pro řešení soustav tuhých systémů vzhledem ke své veliké oblasti stability.

Definice 2.6.7. *Implicitní Eulerova metoda* je výpočetní postup pro řešení diferenciální rovnice definovaný následujícím způsobem:

$$y_{i+1} = y_i + h \cdot f(t_{i+1}, y_{i+1}).$$

V argumentu funkce se nachází neznámá y_{i+1}, jež bývá počítána iterační Newtonovou metodou.

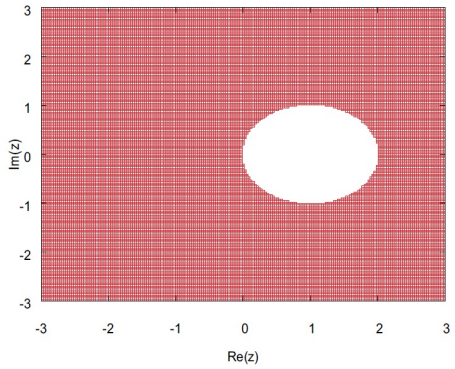
Stabilita metody je dle Dalquistova problému:

$$y_{i+1} = y_i + h \cdot \lambda \cdot y_{i+1} = (1 - h \cdot \lambda)^{-1} y_i = (1 - h \cdot \lambda)^{-(i+1)} y_0.$$

Pro oblast stability metody platí:

$$|R(z)| = |(1-z)^{-1}| \leq 1 \Longleftrightarrow |R(z)| \geq 1.$$

V komplexní rovině se tato skutečnost znázorní následovně:



Obrázek 2.6.2: Stabilita implicitní Eulerovy metody

Z uvedeného obrázku je patrná oblast stability vyplňující celou levou polorovinu (A - stabilita).

Runge-Kutta metody

Metoda Runge-Kutta 4.řádu patří mezi nejpoužívanější metody:

Definice 2.6.8. Runge-Kutta metoda 4. řádu je definována jako:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

$$\begin{aligned} k_1 &= h \cdot f(t_i, y_i), \\ k_2 &= h \cdot f(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_1), \\ k_3 &= h \cdot f(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_2), \\ k_4 &= h \cdot f(t_i + h, y_i + k_3). \end{aligned}$$

Stabilita metody 4. řádu

$$|R(z)| = |1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!}| \leq 1.$$

Pokud se zvýší řád metody, do rovnice stability přibudou další členy, avšak oblast absolutní stability se příliš nezmění.

Z popsaných metod se nejčastěji používá Eulerova implicitní metoda v případě tuhých soustav a Runge-Kuttova metoda v případech jiných. Předkládaná práce vychází především z výzkumu metody Taylorovy řady, která bude popsána v následující části.

2.7 Taylorova řada

Během 18. století bylo objeveno, že dosud známe rozvoje elementárních funkcí (sinus, cosinus, logaritmus) pomocí mocninných řad jsou vlastně speciálním případem obecného rozvoje nazývaného Taylorovou řadou [55].

Definice 2.7.1. Je dán řád polynomu $n \in \mathbb{R}$, bod $x_0 \in \mathbb{R}$, a funkce $f(x)$ definovaná v okolí bodu x_0. Dále se předpokládá, že funkce $f(x)$ má v bodě x_0 vlastní derivaci až do řádu n včetně. Pak polynom $T_n(x)$ se nazývá *Taylorovým polynomem funkce $f(x)$ se středem x_0 stupně n:*

$$T_n(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \frac{f'''(x_0)}{3!}(x-x_0)^3 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^{(n)}. \tag{2.7.1}$$

Zde je vhodné zmínit, že v případě numerických výpočtů založených na metodě Taylorovy řady, určuje stupeň Taylorova polynomu n počet členů Taylorovy řady a tedy i řád metody 2.5.2.

Nyní je dále definován rozvoj funkce $f(x)$ Taylorovým polynomem stupně n.

Definice 2.7.2. Taylorovým vzorcem se nazývá vztah:

$$f(x) = T_n(x) + R_{n+1}(x), \tag{2.7.2}$$

kde $R_{n+1}(x)$ označuje zbytek po n-tém členu Taylorova rozvoje.

Pro přesnější určení $R_{n+1}(x)$ poslouží následující *Taylorova věta.*

Věta. *Nechť $x_0, x \in \mathbb{R}, x \neq x_0$ a $n \in \mathbb{N}$. Předpokládá se, že funkce $f(x)$ 2.7.2 má v uzavřeném intervalu I o krajních bodech x_0, x derivaci až do řádu $(n+1)$. Nechť rozvoj $T_n(x)$ je dán vzorcem 2.7.1. Dále je dána funkce $\varphi(t), t \in [x_0, x]$ spojitá na intervalu I, která má v každém vnitřním bodě intervalu I derivaci $\varphi'(t) \neq 0$. Potom existuje vnitřní bod c intervalu I, pro který platí:*

$$R_{n+1}(x) = \frac{1}{n!}\frac{\varphi(x) - \varphi(x_0)}{\varphi'(c)} f^{(n+1)}(c)(x-c)^n.$$

Tato věta říká, jak lze vyjádřit zbytek $R_{n+1}(x)$ pro rozvoj funkce $f(x)$, která má v okolí bodu x_0 derivace až do řádu $(n+1)$.

Nyní je definován pojem Taylorova řada funkce:

Definice 2.7.3. Je dána funkce $f(x)$, která má v bodě x_0 derivaci všech řádů, potom se řada:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k.$$

nazývá *Taylorovou řadou funkce $f(x)$* se středem v bodě x_0.

Následuje věta vyjadřující konvergenci Taylorovy řady k funkci $f(x)$:

Věta. *Taylorova řada funkce $f(x)$ konverguje v bodě x k funkční hodnotě $f(x)$, právě když:*

$$\lim_{n\to\infty} R_{n+1}(x) = 0.$$

Tato věta je velmi důležitá, neboť říká, že v případě počtu členů Taylorovy řady $n \to \infty$ je Taylorův rozvoj a funkce $f(x)$ identická. Taylorova řada je základem numerické metody Taylorovy řady (MTR).

2.8 Metoda Taylorovy řady

Metoda vychází z práce [34]. Mezi důležité vlastnosti metody patří automatické nastavení řádu metody (tedy použití členů Taylorovy řady o počtu n) v závislosti na velikosti integračního kroku (tedy stabilitě metody). Výhodou tohoto přístupu je především použití tolika členů Taylorovy řady, kolik je potřeba pro zvolenou přesnost. Tato vlastnost ovlivňuje i rychlost výpočtu metodou.

Výhodou metody je také její poměrně snadná paralelizace, kdy jednotlivé mezi-výpočty jsou prováděny nezávisle v oddělených procesorech paralelního systému.

Pro numerický výpočet pomocí metody je nutné provést automatickou transformaci zadání řešené diferenciální rovnice. Původní zadání se transformuje na polynomiální tvar (tvar, u kterého lze rekurentně vypočítat jednotlivé členy Taylorovy řady) a následně se provede zamýšlený výpočet. Výhodou této transformace je především následný tvar zadání, obsahující pouze operandy s operátory pro sčítání, odčítání, násobení a dělení, jenž lze velmi snadno hardwarově implementovat.

Přesnost této metody je přímo závislá na počtu členů Taylorovy řady a tedy omezená délkou bitového slova kde jsou jednotlivé členy uloženy.

Věta. *Velikost použitého bitového slova ovlivňuje přesnost numerického výpočtu Taylorovy řady funkce* $f(x)$.

Důkaz. Je dána Taylorova řadu o počtu členů n pro danou funkci $f(x)$ a předpokládá se bitové pole o maximální velikosti L. Velikost bitového pole b je dána počtem použitých členů n, funkce je označena jako $b(n) = L$. Pokud se vezme další člen Taylorovy řady, bude velikost bitového pole pro $n+1$ členů $b(n+1)$ a $b(n+1) > L$. Tedy s počtem členů Taylorovy řady a velikostí bitového pole roste i přesnost numerického výpočtu. Tedy $b(n+1) > b(n)$. □

Protože statický bitový prostor je standardní vlastností používaných aritmetik, je i přesnost výpočtu Taylorovy řady omezena velikostí použitého bitového slova.V současné době, lze metodu Taylorovy řady použít pro řešení diferenciálních rovnic v programu TKSL a programu TKSL/C

Rekurentní výpočet členů Taylorovy řady

Rekurentní výpočet se definuje následujícím způsobem:

Definice 2.8.1. *Rekurentní výpočet* členu Y_{n+1} je definován vztahem:

$$Y_{n+1} = F(Y_n, Y_{n-1}, ..., Y_0),$$

kde $Y_{n-k}, Y_{n-k+1}, ...$ jsou předcházející výpočty a Y_0 je počáteční hodnota. Funkce F je funkce, na jejímž základě se získá z předcházejících hodnot hodnota Y_{n+1}.

Věta 2.8.2. *Rekurentní výpočet* $j+1$ *členu Taylorovy řady je dán vzorcem:*

$$y_{j+1} = \frac{h^j}{j!}(y^{(j-1)}(t_j)).$$

Důkaz. Explicitním Taylorovým rozvojem funkce $y(t) = y(t_i + h)$ je zápis ve tvaru:

$$y(t_i + h) = y(t_i) + h \cdot y'(t_i) + \frac{h^2}{2} y''(t_i) + ... + \frac{h^n}{n!} y^{(n)}(t_i), \tag{2.8.1}$$

kde $h = t - t_i$. Hodnota $y(t_i)$ je známá a značení $y^{(n)}$ se používá pro n-tou derivaci funkce y. Dle vzorců pro výpočet derivací vyšších řádů lze provést následující substitucí:

$$y(t_i + h) = y(t_i) + h \cdot y'(t_i) + \frac{h^2}{2}(y'(t_i))' + \ldots + \frac{h^n}{n!}(y^{(n-1)}(t_i))'.$$

Potom rekurentní výpočet členu Taylorovy řady lze snadno odvodit a je dán vzorcem:

$$y_{j+1} = \frac{h^j}{j!}(y^{(j-1)})'.$$

Což je odvozený rekurentní vztah. □

Tento zápis zjednodušuje výpočet derivací pro potřeby metody Taylorovy řady.

Stabilita Metody Taylorovy řady

Taylorův rozvoj 2.8.1 se přepíše do tvaru:

$$y_{i+1} = y_i + h \cdot y_i' + \frac{h^2}{2!}y_i'' + \frac{h^3}{3!}y_i''' + \ldots + \frac{h^n}{n!}y_i^{(n)}.$$

Po dosazení do Dalquistova problému vznikne:

$$y_{i+1} = y_i + h \cdot \lambda \cdot y_i + \frac{h^2}{2!} \cdot \lambda^2 \cdot y_i + \frac{h^3}{3!} \cdot \lambda^3 \cdot y_i + \ldots + \frac{h^n}{n!} \cdot \lambda^n \cdot y_i.$$

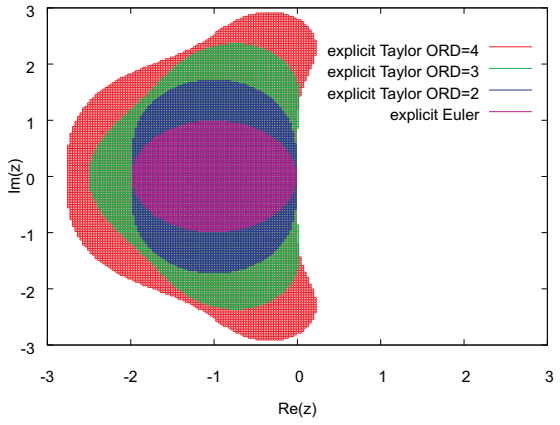
Funkce stability Taylorovy řady je dle vzorce 2.6.2 ve tvaru:

$$R(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \ldots + \frac{z^n}{n!},$$

kde $z = h \cdot \lambda$. Na základě vzorce 2.6.2 lze napsat:

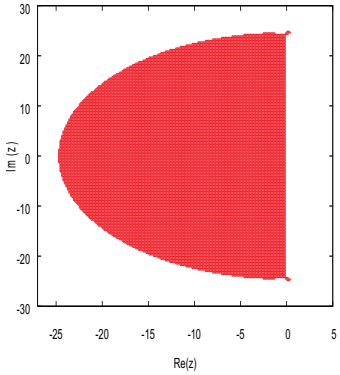
$$| R(z) | \leq 1.$$

Na následujícím obrázku je uvedena stabilita numerické metody řádu $ORD = 1,2,3,4$:



Obrázek 2.8.1: Stabilita MTR pro $ORD = 1,2,3,4$

Věta. *Při zvyšujícím se řádu metody ORD* $\to \infty$ *se metoda Taylorova řady stává A-stabilní.*



Obrázek 2.8.2: Stabilita MTR pro $ORD = 63$

Důkaz věty lze provést na základě zvyšujícího se počtu členů. Tuto skutečnost lze graficky znázornit na obrázku 2.8.2, kdy porovnáním s předchozím obrázkem 2.8.1 je patrné zvětšení oblasti stability.

2.9 Podmíněnost úlohy

V práci jsou zmíněny tuhé systémy, které lze popsat jako špatně podmíněné úlohy. Dříve však je nutné definovat obecný pojem matematická úloha.

Definice 2.9.1. *Matematická úloha* je zobrazení $y = f(x)$, které vstupním datům $x \in I$ přiřadí výstupní data $y \in O$ a množiny I, O jsou Banachovy prostory[2].

Protože v práci budou uvedeny pouze úlohy, které jsou jednoznačně řešitelné s ohledem na spojitost, zavede se pojem korektnost [8].

Definice 2.9.2. Jsou dány množiny I, O, které jsou Banachovými prostory, pak matematické úloha $y = f(x)$ je *korektní*, pokud:

- ke každému $x \in I$ existuje jediné řešení $y \in O$,
- řešení spojitě závisí na vstupních datech, tedy pokud platí: $x \to a$ pak také $f(x) \to f(a)$.

Nekorektními úlohami jsou tedy ty úlohy, které mají nejednoznačné řešení, nebo vykazují známky nespojitosti při řešení. Bohužel v praxi i korektní úloha může při malé změně vstupních parametrů vykazovat velké změny ve výsledcích, taková úloha se nazývá špatně podmíněnou úlohou (v diferenciálním počtu se bude za takové úlohy považovat řešení tuhých systémů). Proto pro rozlišení těchto dvou druhů úloh se zavádí pojem relativní číslo podmíněnosti úlohy.

[2] http://mathworld.wolfram.com/BanachSpace.html

Definice 2.9.3. Nechť je norma vstupu $\|x\| \neq 0$ a normu výstupu $\|y\| \neq 0$ pak *relativním číslem podmíněnosti úlohy* $y = f(x)$ se nazývá číslo C_r, jejíž hodnota bude stanovena jako poměr velikostí relativních změn výstupů vůči vstupům:

$$C_r = \frac{\frac{\|y+\triangle y\|}{\|y\|}}{\frac{\|x+\Delta x\|}{\|x\|}} .$$

Korektní úloha se prohlásí za dobře podmíněnou, pokud $C_r \approx 1$, a za špatně podmíněnou, pokud $C_r > 100$.

V praxi bývá často také použito absolutní číslo podmíněnosti úlohy:

Definice 2.9.4. Nechť $\|\triangle x\| \neq 0$, pak *absolutní číslo podmíněnosti úlohy* C_a je určeno jako poměr mezi velikostí změny výstupu, vůči velikosti změny vstupu:

$$C_a = \frac{\|\triangle y\|}{\|\triangle x\|} .$$

Určení čísla C_a ve většině případů řešení podmíněnosti úlohy postačuje.

Shrnutí

Kapitola se zabývá příčinami chyb vznikajících během matematických výpočtů na počítačích. Nejprve jsou diskutovány obecně absolutní a relativní chyby a následně se přechází k chybám, které jsou součástí numerických metod používaných pro výpočet ODR. Kapitola se kromě diskretizačních a zaokrouhlovacích chyb zaměřuje na stabilitu numerických metod. Závěrečná část kapitoly se věnuje Taylorově řadě a z ní odvozené metodě Taylorovy řady, jejíž přesné výpočty byly motivací disertační práce. Nakonec je uvedena podmíněnost úlohy charakterizující vliv vstupních dat na průběh řešení.

Kapitola 3

Semi - analytické výpočty ODR používané v současnosti

Následující kapitola bude určena dvěma nejrozšířenějším způsobům zpřesnění a urychlení matematických výpočtů s nimiž se lze v dnešní době setkat. Protože půjde především o řešení obyčejných diferenciálních rovnic (ODR) je zde uvedena alespoň základní definice v implicitním tvaru..

Definice 3.0.5. *Diferenciální rovnicí* je označena rovnice ve tvaru:

$$F(y^{(n)}, y^{(n-1)}, ..., y', y, t) = 0, \tag{3.0.1}$$

kde F je reálná funkce $n+2$ proměnných.

Dále je definováno řešení diferenciální rovnice:

Definice 3.0.6. *Řešením diferenciální rovnice* 3.0.1 se rozumí funkce y definovaná na neprázdném otevřeném intervalu I, která má v každém bodě intervalu I vlastní derivaci až do řádu n a jejíž hodnoty spolu s hodnotami derivací splňují rovnici 3.0.2:

$$F(y^{(n)}(t), y^{(n-1)}(t), ..., y'(t), y(t), t) = 0, \tag{3.0.2}$$

v každém bodě intervalu I (tedy platí pro každé $t \in I$).

U reálných systémů je analytické řešení ODR příliš náročné nebo nemožné (diskutováno v článku *Application of the Modern Taylor Series Method to a Multi-Torsion Chain*), proto je v těchto případech vhodné nalézt řešení pomocí numerických metod.

3.1 Semi - analytické výpočty ODR metodou Taylorovy řady

Jednoduché analytické modely řešené ve spojité simulaci umožňují získat přesné řešení za krátký čas. Řešení složitějších analytických modelů vede často k problémům a převodům na relativně složité integrální, případně rekurentní vztahy, kdy vyhodnocení je nutné provést numericky. Výpočty, které řeší složité analytické modely v relativně krátké době při zachování vysoké přesnosti, se označují jako semi - analytické.

Definice 3.1.1. Je dáno řešení ODR v $\mathbb{R}^n$, pokud množině vzorů $t \in T$ (hodnot dosazených do rovnice ODR) odpovídá okolí obrazu analytického řešení ODR $U(t, \varepsilon)$, takové, že ε je velmi malé, lze považovat řešení a způsob řešení za semi - analytický.

V této práci se za semi - analytické výpočty budou považovat především výpočty s přímým využitím metody Taylorovy řady (MTR) a s vysokým řádem metody. Řád metody označuje počet členů Taylorovy řady a změnou řádu metody lze dosáhnout požadované přesnosti výpočtu [25]. Dále jsou uvedeny výsledky, kterých bylo dosaženo za použití simulačního programu TKSL, využívajícího metodu MTR.

ODR s řešením ve tvaru polynomiální funkce

Definice 3.1.2. ODR ve tvaru:

$$F(y', C_0 \cdot t^{C_0-1}) = 0,$$

kde C_0 je konstanta a řešením je polynomiální funkce:

$$F(y(t), t^{C_0}) = 0,$$

se bude nazývat ODR s řešením ve tvaru polynomiální funkce.

U těchto rovnic byly provedeny experimenty viz. příloha (strana 103), kde bylo potvrzeno, že analyticky přesného výpočtu (semi - analytického) u polynomiálních funkcí pomocí MTR v programu TKSL se dosáhne při volbě řádu metody $ORD > C_0 + 1$ a při použití tohoto řádu nezáleží na velikosti kroku h.

Testovací funkce Z a řešení ODR ve tvaru exponenciální funkce

Definice 3.1.3. Testovací funkcí Z se bude označovat:

$$Z = x \cdot y,$$

kde proměnné x a y jsou řešením diferenciálních rovnic:

$$F(y', e^t) = 0, F(x', e^t) = 0,$$

ve tvaru:

$$F(y(t), e^t) = 0, F(x(t), e^t) = 0.$$

Exponenciální funkce na rozdíl od polynomiálních funkcí mají nekonečný rozvoj a z těchto důvodů nelze provést výpočty s libovolným integračním krokem (jako tomu bylo v případě polynomiálních funkcí při volbě vysokého řádu metody), přesto lze numerickým výpočtem pomocí MTR získat analytické řešení testovací funkce Z.

Příklad 3.1.4. Semi - analytickým výpočtem se ověří hodnota testovací funkce:

$$Z = x \cdot y,$$

kde funkce x, y jsou řešením diferenciálních rovnic:

$$x' = x, x(0) = 1,$$
$$y' = -y, y(0) = 1.$$

Analytické řešení diferenciálních rovnic je ve tvaru:

$$x = e^t, y = e^{-t}.$$

V případě použití MTR a TKSL lze získat $Z = 1$.

Podobný výsledek lze získat i v následujícím příkladě:

Příklad 3.1.5. Je dána rovnice:

$$y' = e^{(t+1)^2 sin(t)}[2(t+1)sin(t) + (t+1)^2 cos(t)],$$
$$x' = e^{-(t+1)^2 sin(t)}[2(t+1)sin(t) + (t+1)^2 cos(t)].$$

Opět bude použita testovací funkce $Z = x \cdot y$. Analytické řešení soustav rovnic je:

$$y = e^{(t+1)^2 sin(t)}, x = e^{-(t+1)^2 sin(t)}.$$

Při přesném výpočtu se vynásobením analytických funkcí získá hodnota testovací funkce $Z = 1$, platná v celém intervalu výpočtu. Program TKSL automaticky nastaví řád metody ($ORD = 22$) a rovnost $Z = 1$ je zachována.

Obrázek 3.1.1: Hodnota funkce Z pro $ORD = 4$ (MTR) v intervalu $T \in < 0, 10 >$

Pokud by se řád $ORD = 4$ v programu TKSL nastavil explicitně (odpovídá metodě Runge - Kutta 4. řádu), vyšel by výsledek zobrazený na předchozím obrázku 3.1.1. Z obrázku 3.1.1 je patrné, že v případě řádu metody $ORD = 4$, hodnota funkce $Z \neq 1$ pro $T = 10$.

Z výše uvedených příkladů je zřejmé, že při nízkém řádu metody použité v TKSL nejsou numerické hodnoty testovací funkce Z rovny analytickým a s přibývajícím počtem kroků h relativní chyba testovací funkce Z roste.

Přesný výpočet určitého integrálu pomocí TKSL

Definice 3.1.6. ODR ve tvaru:

$$F(y', f(t)) = 0,$$

kde funkce f je goniometrická a platí pro ni $\omega = \frac{2\pi}{T}, T > 0$ (symbolem T se bude označovat perioda). Řešením je goniometrická funkce:

$$F(y(t), f(t)) = 0,$$

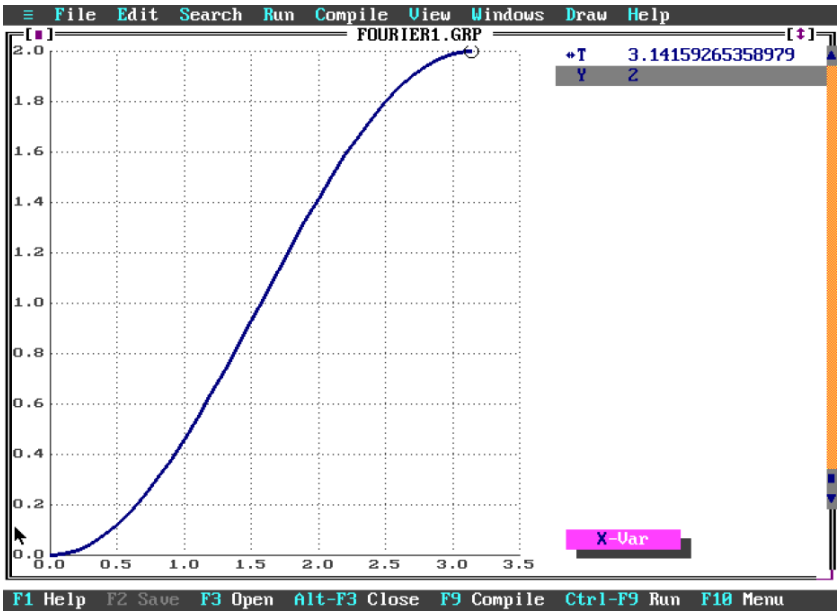
která se bude nazývat ODR s řešením ve tvaru goniometrické funkce.

Přesnost semi - analytického při výpočtu určitého integrálu si lze ukázat na následujícím příkladě:

Příklad 3.1.7. Provede se výpočet určitého integrálu, kde horní mez integrálu bude $u = \pi$:

$$y = \int_0^u sin(t)dt.$$

Průběh řešení v programu TKSL lze vidět na obrázku:



Obrázek 3.1.2: Výpočet určitého integrálu pomocí MTR

Z obrázku je patrná analytická přesnost výpočtu pomocí MTR v TKSL, kdy $y(\pi) = 2$. Při použití MTR se integrál:

$$Y = \int sin(t)dt,$$

převede na diferenciální rovnici:

$$Y' = sin(t).$$

a dospěje se k analyticky přesnému výsledku (relativní chyba je nulová).

Z výsledků na obrázku 3.1.2 je patrná analytická přesnost výpočtu určitého integrálu pomocí MTR řešící ODR. Další část kapitoly se bude zabývat symbolickými výpočty, kde řešení lze ODR získat algebraickými transformacemi bez zaokrouhlení.

3.2 Symbolické výpočty

Symbolické (algebraické) výpočty lze charakterizovat jako výpočty se symboly reprezentující matematické objekty. Mezi tyto objekty patří čísla, boolovské hodnoty, rovnice, výrazy, polynomy,

funkce a jiné algebraické struktury. Vzhledem k tomu, že symbolické operace nemají za následek zaokrouhlení výsledku (postup při řešení bývá striktně algebraický), lze je považovat za protiklad numerických výpočtů, kde aproximace řešení závisí na přesnosti zvolené výpočetní architektury (práce v pohyblivé čárce), přesnosti zvolené metody, případně její stabilitě[1].Software, který symbolické operace podporuje, se v odborné literatuře označuje zkratkou CAS[2]. Počátky zmíněných systémů se datují do 60. let 20. století, kdy docházelo k přesunům výpočtů prováděných na papíře na sálové počítače. Při implementacích algoritmů se zjistilo, že dosud používané algoritmy jsou v prostředí počítačů značně pomalé a nepřesné. Proto dochází k revizi známých algoritmů a jejich přizpůsobení výpočetní technice. Nejznámější výzkumná skupina zabývající se touto oblastí je SIGSAM[3], přímo podporující konferenci ISSAC[4] a časopis Journal of Symbolic Computation. Vzhledem k značné rozdílnosti těchto výpočtů od aproximací používaných v numerické matematice je vhodné alespoň okrajově zmínit její podstatné odlišnosti [63] v následujících odstavcích.

Reprezentace čísel

Při symbolických výpočtech je zapotřebí pracovat s přesnými čísly a k tomu je i uzpůsobena vnitřní reprezentace dat v programech CAS. V případě přesných výpočtů dochází k chování nazývanému *expression swell*, kdy velikost dat roste nepředvídatelným způsobem.

Příklad 3.2.1. Příkladem tohoto chování může být např. nekonečný rozvoj racionálního čísla vzniklého po dělení dvou nesoudělných čísel např.:

$$\frac{1}{3} = 0,\dot{3}.$$

Zmíněné číslo se obvykle použitím numerických výpočtů zaokrouhlí na velikost používaného bitového slova (nejčastěji 32 nebo 64-bitového), což vede ke ztrátě přesnosti. V případě CAS programů dochází k substituci čísla proměnnou, která je až po žádosti uživatele programu vyhodnocena.

Výrazy

Výrazy se skládají z operátorů a operandů. Symboly operátorů označují matematické operace a struktury, operandy jsou tvořeny proměnnými v nichž jsou uloženy hodnoty, ale ty se během symbolických výpočtů nevyhodnocují. Např. matici lze označit jako výraz s operátorem "matice" a operandy tvoří proměnné obsahující řádky této matice.

V symbolických výpočtech se na každý výraz, skládající se z operandů a operátorů, nahlíží jako na soubor podprogramů, kdy nejmenší podprogramy jsou složeny ze základních aritmetických operací. Např. výraz $a * b$, představuje podprogram pro násobení obsahující dva vstupní parametry. Toto členění umožňuje skládání složitějších výrazů z jednodušších.

Při řešení v symbolických výpočtech dochází k úpravám výrazů vzhledem k proměnným označující skutečné hodnoty, nikoliv k hodnotám samotným. Protože nedochází k numerickému vyhodnocování má popsaný algebraický postup fundamentální význam vzhledem ke správnosti výsledku.

[1] v předkládané práci se zabýváme především diferenciálním a integrálním počtem

[2] Computer algebra systems

[3] Special Interest Group on Symbolic and Algebra Manipulation

[4] International Symposium on Symbolic and Algebraic Computation

Zjednodušení

Při úpravách výrazů dochází často k jejich rozšiřování, proto je vhodné výrazy, kdykoliv je to možné (a nedojde ke ztrátě přesnosti) zjednodušit. K tomu zjednodušení dochází aplikací přepisovacích pravidel např. $cos(0) \rightarrow 1$ [5].

Vzhledem k příkladům uvedeným v další části je vhodné se také zmínit o několika druzích komplikací vznikajících při zjednodušování výrazů a jejich řešení:

- asociativní operace - (např. sčítání a násobení) - výrazy obsahující tyto operace se převádějí na symboly operátorů a operandů v prefixové podobě, např. $x*y*z$ se transformuje na tvar "*"(x,y,z) , v případě dělení a odečítání se výrazy $-x, x-y, x/y$ změní na $(-1)*x, x+(-1)*y, x*y^{-1}$, tedy ve vnitřní reprezentaci jsou používány pro symbolické výpočty pouze operace sčítání a násobení, u nichž je zajištěna jejich uzavřenost vzhledem k transformacím s výrazy.

- komutativnost operací sčítání a násobení - problém spočívá v rozpoznání, zda - li je vhodné aktuální výrazy spolu zkombinovat, či pracovat s nimi nezávisle. Časově méně náročná je práce se sloučenými výrazy, ale tato volba nemusí být vždy vzhledem k dalším transformacím optimální. Programy CAS využívají několika způsobů řešení tohoto problému, bude diskutováno řešení v programu Maple[6]. Program používá hashovací funkci,[7] která je schopna najít vhodné výrazy pro kombinaci a toto sloučení okamžitě provést. Hash funkce také rozpozná stejné podvýrazy a zabrání opakování už jednou provedených operací, čímž se rychlost výpočtu urychluje.

- distributivita výrazů - s problémem popsaným výše souvisí i použití přepisujících pravidel vyjadřující stejný výraz, např. $(x-1)(x^4+x^3+x^2+x+1) <=> x^4-1$, pokud neexistuje obecně správná volba k aplikaci těchto pravidel, zpravidla dochází k jejich užití pouze na explicitní žádost uživatele.

Nyní bude uvedeno použití symbolických výpočtů (je třeba ještě zmínit, že symbolické operace byly podnětem pro vytvoření programovacího jazyka Lisp [73], proto jeho přednosti v této oblasti využijeme).

Příklad 3.2.2. Symbolickými operacemi se provede derivace následujícího výrazu a ukáže postup výpočtu symbolickým způsobem. Symbol t bude považován za proměnnou a a,b,c budou konstanty. Výraz:

$$at^2+bt+c,$$

lze v prefixové formě přepsat na:

$$(+c(+(\cdot bt)(\cdot a(\cdot tt)))),$$

odpovídající zápisu v programu Lisp (v prefixové formě). Po derivaci výrazu dle proměnné t se získá výraz:

$$2at+b = (+b(\cdot 2(\cdot at)).$$

Nyní bude diskutována derivace výrazů v závorkách. Při derivování dochází k následujícím případům:

[5] přepisovací pravidla v symbolických výpočtech jsou obdobou přepisovacích pravidel používaných v počítačových gramatikách

[6] matematický program založený na symbolických výpočtech [38]

[7] funkce, která převede vstup na jednoznačnou sekvenci znaků, zabírající menší paměťový prostor počítače

- $\frac{dy}{dt} = 0$, pokud y je konstanta nebo proměnná jiná než t,
- $\frac{dy}{dt} = 1$, pokud $y = t$ (odstraní se proměnná t).

Nebo lze výraz zjednodušit:

- $\frac{d(u+v)}{dt} = \frac{du}{dt} + \frac{dv}{dt}$ představuje rekurzivní zjednodušení,
- $\frac{d(u \cdot v)}{dt} = u \cdot \frac{dv}{dt} + v \cdot \frac{du}{dt}$ také představuje rekurzivní zjednodušení.

Pro příklad z diferenciálního počtu se použije jazyk LISP, ve kterém bude proveden jednoduchý symbolický výpočet derivace. Na základě předcházejících úprav lze vytvořit následující podprogramy (funkce):

- Funkce *prediktor 1* určuje, zda-li je daný výraz ekvivalentní danému číslu:

Algoritmus 1 Symb. výpočty - prediktor 1

```
(define (=number? exp num)
    (and (number? exp) (= exp num)))
```

- Funkce *prediktor 2* zjistí, jestli je člen symbolem nebo proměnnou:

Algoritmus 2 Symb. výpočty - prediktor 2

```
(define (variable? x) (symbol? x))
```

- Funkce *prediktor 3* zjistí, zda-li jsou předané proměnné shodné:

Algoritmus 3 Symb. výpočty - prediktor 3

```
(define (same-variable? v1 v2)
    (and (variable? v1) (variable? v2) (eq? v1 v2)))
```

- Funkce *konstruktoru 1* vytváří součet:

Algoritmus 4 Symb. výpočty - konstruktor 1

```
(define (make-sum a1 a2)
        (cond ((=number? a1 0) a2)
              ((=number? a2 0) a1)
              ((and (number? a1) (number? a2)) (+ a1 a2))
              (else (list '+ a1 a2))))
```

- *Kostruktor 2* provede součin:

Algoritmus 5 Symb. výpočty - konstruktor 2

```
(define (make-product m1 m2)
    (cond ((or (=number? m1 0) (=number? m2 0)) 0)
        ((=number? m1 1) m2)
        ((=number? m2 1) m1)
        ((and (number? m1) (number? m2)) (* m1 m2))
        (else (list '* m1 m2))))
```

- Funkce *prediktor 4* zjistí, zda-li je první prvek součtem:

Algoritmus 6 Symb. výpočty - prediktor 4

```
(define (sum? x)
    (and (pair? x) (eq? (car x) '+)))
```

- Funkce *selektor 1* pro druhý člen součtu:

Algoritmus 7 Symb. výpočty - selektor 1

```
(define (addend s) (cadr s))
```

- Funkce *selektor 2* pro třetí člen součtu:

Algoritmus 8 Symb. výpočty - selektor 2

```
(define (augend s) (caddr s))
```

- Funkce *prediktor 5* zjišťuje, jestli je první prvek součin:

Algoritmus 9 Symb. výpočty - prediktor 5

```
(define (product? x)
    (and (pair? x) (eq? (car x) '*)))
```

- Funkce *selektor 3* pro druhý člen součinu:

Algoritmus 10 Symb. výpočty - selektor 3

```
(define (multiplier p) (cadr p))
```

- Funkce *selektor 4* pro třetí člen součtu:

Algoritmus 11 Symb. výpočty - selektor 4

```
(define (multiplicand p) (caddr p))
```

Nyní lze přistoupit k symbolickému výpočtu derivace:

Algoritmus 12 Symb. výpočty - derivace

```
(define (deriv exp var)
    (cond ((number? exp) 0)
        ((variable? exp)
         (if (same-variable? exp var) 1 0))
        ((sum? exp)
         (make-sum (deriv (addend exp) var)
                   (deriv (augend exp) var)))
        ((product? exp)
         (make-sum
          (make-product (multiplier exp)
                        (deriv (multiplicand exp) var))
          (make-product (deriv (multiplier exp) var)
                        (multiplicand exp))))
         (else
          (error "neznamy typ vyrazu" exp))))
```

Z výpočtu je možno vyčíst volání dříve definovaných programů. U větve pro sčítání a násobení je možné si všimnout rekurzivního volání programu, což umožňuje dostat se k základním objektům definujícím výraz. Pokud je nyní provedeno volání programu pro výraz:

```
> (deriv '(+ c (+ (* b t) (* a (* t t)))))
```

Dospěje se ke správnému výpočtu:

```
(+ b (* 2 (* a t)))
```

Vzhledem k tomu, že symbolické výpočty se staly nepostradatelnými pro dosažení přesných výpočtů, jsou standardně součástí matematických programů jako je Maple, Mathematica, Matlab. Problematika symbolických výpočtů je poměrně rozsáhlá a více informací lze nalézt např. v [84], [85].

Shrnutí

Kapitola se zabývala současnými přesnými způsoby výpočtu ODR. Nejprve jsou uvedeny příklady výpočtu ODR pomocí MTR v programu TKSL. Následně je uveden způsob výpočtu pomocí symbolických operací v programovacím jazyce LISP.

Kapitola 4

Teorie zbytkových tříd

Paralelní výpočty, které jsou popsány v následujících kapitolách, staví na poznatcích z oboru teorie čísel. Nejdůležitější definice, z nichž vychází předkládaný výzkum, jsou shrnuty v této kapitole.

4.1 Konečná tělesa

Vzhledem k používání množin prvočísel je vhodné uvést několik definic [6].

Definice 4.1.1. Množina F, která obsahuje alespoň dva prvky spolu se dvěma operacemi $+, \cdot$ se nazývá *tělesem*, pokud operace splňují následující vlastnosti:

1. $a+(b+c) = (a+b)+c$ pro libovolné $a,b,c \in F$,
2. $a+b = b+a$ pro libovolné $a,b \in F$,
3. existuje $0 \in F$, tak že pro všechna $a \in F$ platí: $a+0 = 0+a = a$,
4. pro všechna $a \in F$ existuje $-a \in F$ tak, že platí $a+(-a) = (-a)+a = 0$,
5. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ pro všechna $a,b,c \in F$,
6. $a \cdot b = b \cdot a$ pro všechna $a,b \in F$,
7. existuje $1 \in F$ tak, že pro všechna $a \in F$ platí $1 \cdot a = a \cdot 1 = a$,
8. pro všechna $a \neq 0$ existuje $a^{-1} \in F$ tak, že platí $a \cdot a^{-1} = a^{-1} \cdot a = 1$,
9. $a \cdot (b+c) = a \cdot b + a \cdot c$ pro všechna $a,b,c \in F$,
10. $0 \neq 1$.

V předkládané práci se bude vycházet především z množiny $\mathbb{Z}_p = \{0,1,...,p-1\}$, kde p je prvočíslo. Protože tato množina splňuje všechny axiomy z předcházející definice, je nazývána tělesem.

Definice 4.1.2. Polynom $f(x) \in F[x]$ (těleso polynomů F) se nazývá *ireducibilní nad* F, pokud stupeň polynomu $deg(f) > 0$ a je dělitelný pouze konstantou $c \in F, c \neq 0$, pro kterou platí $c \mid f(x)$, nebo $c \cdot f(x) \mid f(x)$.

Tedy jinými slovy, ireducibilní polynom nad daným tělesem je takový polynom, který není v daném tělese rozložitelný [69].

Věta. *Pro každé konečné těleso F existuje prvočíslo p a ireducibilní polynom* $f(x) \in \mathbb{Z}_p[x]$ *takové, že F je izomorfní s* $\mathbb{Z}_p[x]/f(x)$.

Důkaz věty lze nalézt např. v [21]. Zabývejme se nyní počtem prvků konečného tělesa.

Věta. *Počet prvků konečného tělesa* $GF(q)$[1] *je roven číslu* $q=p^k$*, kde číslo p je prvočíslo a k je kladné přirozené číslo.*

Důkaz této věty lze nalézt v [42]. Důsledkem předcházející věty je, že pro každé $k \geq 1$ a každé prvočíslo p existuje v $\mathbb{Z}_p[x]$ ireducibilní polynom stupně k.

Příklad 4.1.3. Např. neexistuje $GF(p^k)$, které by mělo 6 prvků. Neboť pro žádné prvočíslo a kladné přirozené číslo neplatí $6 = p^k$. Existuje ale $GF(5^1)$, nebo $GF(2^3)$.

Definice 4.1.4. Charakteristika tělesa λ udává nejmenší počet jedniček, jejichž součet dává číslo 0. Tedy platí $\sum_1^\lambda 1 = 0$, kde λ je nejmenší kladné číslo s touto vlastností.

Pokud vlastnost není splněna pro žádný počet jedniček je charakteristika rovna ∞.

Příklad 4.1.5. Charakteristika tělesa Z_p je rovna číslu p. Např. velmi používaným tělesem v informatice je $GF(2^k)$, jehož charakteristika je rovna číslu 2.

Vraťme se nyní k polynomům a zabývejme se GF.

Příklad 4.1.6. Mějme ireducibilní polynom $f(x) = x^2 + x + 1$ a těleso nad tímto polynomem $\mathbb{Z}_2[x]$. Pak pro toto těleso platí:

$$\mathbb{Z}_2[x]/(x^2 + x + 1) = \{0, 1, \beta, \beta + 1\}.$$

kde β je kořenem rovnice $f(x) = x^2 + x + 1$, počet prvků je izomorfní s tělesem $GF(2^2)$. Pro výpočet se používá tzv. tabulka mocnin.

β^k	$\beta^k = \beta \cdot \beta^{k-1}$	algebra	výsledek
β^0			1
β^1			β
β^2	$\beta + 1$		$\beta + 1$
β^3	$\beta(\beta + 1)$	$\beta^2 + \beta$	1

Tabulka 4.1.1: Tabulka mocnin pro $GF(4)$

Protože je poslední sloupec tabulky nenulový, nazýváme kořen β primitivním. Sčítání se provádí prostým součtem s ohledem na prvočíslo $p = 2$ (operace modulo). Tedy $\beta + 1 + \beta = 2\beta + 1 = 1$. Pro násobení platí stejná pravidla jako pro sčítání: $\beta \cdot \beta^2 = \beta(\beta + 1) = \beta^2 + \beta = \beta + 1 + \beta = 2\beta + 1 = 1$. Inverzní prvek najdeme jako nejvyšší mocninu mínus exponent prvku, ke kterému hledáme inverzi. Tedy $(\beta + 1)^{-1} = (\beta^2)^{-1} = \beta^{3-2} = \beta^1 = \beta$.

Nyní již lze přejít k aritmetice zbytkových tříd.

[1] Galois Field - takto označujeme konečná tělesa podle matematika Évariste Galoise

4.2 Jedno - modulová aritmetika zbytkových tříd

Nejprve bude uvedena definice zbytku a kongruence.

Definice 4.2.1. Nechť $a,q,m,r \in \mathbb{Z}$. Zbytek r po celočíselném dělení čísla a číslem m se zapíše rovnicí:

$$a = q \cdot m + r,$$

a tato operace se nazývá operací *modulo* a označuje jako $a \bmod m$.

Důsledek. *Zbytek r nabývá hodnot v intervalu* $\{0, m-1\}$.

Nyní je definována kongruence.

Definice 4.2.2. Nechť $a,b,m \in \mathbb{Z}$, kde $m > 1$. Pokud $m \mid (b-a)$ říká se, že b je *kongruentní* k a modulo m a píše se:

$$b \equiv a \,(mod\, m).$$

Pokud $m \nmid (b-a)$ pak b *není kongruentní* k a modulo m a píše se:

$$b \not\equiv a \,(mod\, m).$$

Tedy kongruence znamená, že rozdíl čísel $b-a$ je násobkem čísla m.

Příklad 4.2.3. Jsou dány čísla $a = -43, b = 37$, potom $b - a = -80$ a platí $-43 \equiv 37 \,(mod\, 4)$.

Věta 4.2.4. *Kongruence* $\equiv$ *pro stanovené mod m je relací ekvivalence na množině* $\mathbb{Z}$.

Na základě věty uvedené výše, se provede seskupení získaných stejných zbytků r.

Definice 4.2.5. Množinu čísel kongruentních k $a \bmod m$ (seskupených na základě velikosti zbytku r) se nazývá *zbytkovou třídou* a značí jako $[a]$.

Příklad 4.2.6. Zbytkové třídy $[a]$ k $a \bmod 3$ jsou:

$$\begin{aligned} [0] &= \{..., -9, -6, -3, 0, 3, 6, 9, ...\}, \\ [1] &= \{..., -7, -4, -1, 1, 4, 7, ...\}, \\ [2] &= \{..., -8, -5, -2, 2, 5, 8, ...\}, \end{aligned}$$

Definice 4.2.7. Množina zbytkových tříd vzhledem k m se značí jako $\mathbb{Z}_m$ a platí pro ni:

$$\mathbb{Z}_m = \{0, 1, 2, 3, ..., m-2, m-1\}.$$

Následující věty jsou uvedeny bez důkazů.

Věta 4.2.8. *Nechť* $a,b,m \in \mathbb{Z}$, *kde* $m > 1$. *Potom platí:*

$$a \equiv b \,(mod\, m),$$

a

$$b \equiv a \,(mod\, m),$$

potom

$$a - b \equiv 0 \,(mod\, m).$$

Uvedená věta říká, že pokud jsou dvě čísla vůči sobě a určitému zbytku kongruentní, pak uvedená operace zachovává kongruenci.

Věta 4.2.9. *Nechť* $a,b,c,m \in \mathbb{Z}$*, kde* $m > 1$*. Pokud*

$$a \equiv b \,(mod\, m),$$

a

$$b \equiv c \,(mod\, m),$$

potom

$$a \equiv c \,(mod\, m).$$

Věta vyjadřuje tranzitivnost kongruence (čísla patří do stejné zbytkové třídy).

Věta 4.2.10. *Nechť* $a,b,c,d,m \in \mathbb{Z}$*, kde* $m > 1$*. Pokud*

$$a \equiv b \,(mod\, m),$$

a

$$c \equiv d \,(mod\, m),$$

potom

$$ac \equiv bd \,(mod\, m),$$

i

$$a + c \equiv (b + d) \,(mod\, m).$$

Věta ukazuje asociativnost kongruence pro násobení a sčítání.

Věta 4.2.11. *Nechť* $a,b,c,d,x,y,m \in \mathbb{Z}$*, kde* $m > 1$*. Pokud*

$$a \equiv b \,(mod\, m),$$

a

$$b \equiv a \,(mod\, m),$$

potom

$$ax + cy \equiv (bx + dy) \,(mod\, m),$$

i

$$(a + x)(c + y) \equiv (b + x)(d + y) \,(mod\, m).$$

V dalším textu se bude vycházet z následující definice.

Definice 4.2.12. Je dáno zobrazení $\mathbb{Z} \to \mathbb{Z}_m$ a zbytek $0 \leq r < m$, pak zápisem

$$| b |_m = r,$$

se označuje kongruence:

$$b \equiv r \,(mod\, m).$$

Nyní jsou uvedeny věty založené na předcházející definici.

Věta 4.2.13. *Nechť $a, b, m \in \mathbb{Z}$, kde $m > 1$. Potom:*

$$\begin{aligned} | a+b |_m &= ||a|_m + |b|_m|_m, \\ &= |a + |b|_m|_m, \\ &= ||a|_m + b|_m, \end{aligned}$$

$$\begin{aligned} | a \cdot b |_m &= ||a|_m \cdot |b|_m|_m, \\ &= |a \cdot |b|_m|_m, \\ &= ||a|_m \cdot b|_m. \end{aligned}$$

Věty říkají, že nezáleží na pořadí provedení kongruencí při naznačených operacích, výsledek je vždy stejný. Nyní je vhodné definovat aditivní inverzi:

Definice 4.2.14. *Aditivní inverzí modulo m* se bude nazývat vztah:

$$\underline{a} \equiv | -a |_m = m - a,$$

a lze ji zapsat ve tvaru:

$$| a - b |_m \equiv | a + \underline{b} |_m .$$

Důležité vlastnosti, které se budou používat při výpočtech v RNS, říkají následující věty.

Věta 4.2.15. *Množina $(\mathbb{Z}_m, +, \cdot)$ tvoří algebraickou strukturu okruh.*

Důkaz. Na základě ověření vlastností platných pro okruh. □

Věta 4.2.16. *Okruh $(\mathbb{Z}_m, +, \cdot)$ je komutativním tělesem právě tehdy, pokud modulo m je prvočíslo.*

Důkaz. Na základě aplikace pravidel platných pro komutativní těleso. □

V dalším textu se bude množinou $\mathbb{Z}_m$ označovat množina $\mathbb{Z}_p$.

Definice 4.2.17. Symbolem $\mathbb{Z}_p$se označí množina zbytkových tříd, kde p je prvočíslo.

Věta 4.2.18. *Z množiny zbytkových tříd $\mathbb{Z}_p$ se vezme libovolné číslo $b \neq 0$, kdy $b \in \mathbb{Z}_p$, pak existuje jediné číslo $c \in \mathbb{Z}_p, c \neq 0$ splňující rovnici:*

$$\mid c \cdot b \mid_p = \mid b \cdot c \mid_p = 1.$$

Důkaz. Důkaz sporem. □

Definice 4.2.19. Pro čísla $b, c \in \mathbb{Z}_p$ a prvočíslo p, lze nalézt jednoznačné číslo c označující *multiplikativní inverzi b modulo p*, toto vlastnost se zapisuje jako:

$$c = b^{-1} (mod\, p).$$

Příklad 4.2.20. K číslu $3 \in \mathbb{Z}_7$, se hledá multiplikativní inverze, tedy $\mid 3^{-1} \mid_7$. Dosazením čísla5 za 3^{-1} se dostane $\mid 3 \cdot 5 \mid_7 = \mid 15 \mid_7 = 1$, což je hledaná multiplikativní inverze.

Příklad 4.2.21. Pokud bude $p = 7$, pak $\mathbb{Z}_7 = \{0,1,2,3,4,5,6\}$a všechna tato čísla budou mít multiplikativní inverzi modulo 7.

Při výpočtu multiplikativních inverzí se používá několik algoritmů, které zde budou dále uvedeny.

Euklidův algoritmus (GCD)

Tento algoritmus se používá ke zjištění největšího společného dělitele (GCD) čísel $A, B \in \mathbb{N}$. Často je nazýván také Euklidovým algoritmem a formálně ho lze zapsat následovně:

Algoritmus 13 Euklidův algoritmus (GCD)

Require: $A, B \in \mathbb{N}$
Ensure: $C = GCD(A, B)$
  while $B \neq 0$ **do**
    $t = B$
    $B = A \bmod t$
    $A = t$
  end while
  return C

Uvedený algoritmus lze využít také pro zjištění, zda-li jsou daná dvě čísla a, b vůči sobě nedělitelná ($\mathrm{GCD}(a,b) = 1$) a označují se jako *coprime.* (Ve výpočtech v aritmetice zbytkových tříd se využije vlastnosti, že pokud je dané číslo X nesoudělné se všemi prvočísly menšími než X, pak je také prvočíslem). Nejhorší časová složitost výpočtu (klasifikace asymptotické časové složitosti algoritmu jsou uvedeny v příloze B) se získá, pokud číslo a, b jsou po sobě následující čísla *Fibonacciho posloupnosti*. Pokud $a \geq b$, pak složitost bude $O(log\, b)$ [86].V implementaci mého systému používám rychlejší binární Euklidův algoritmus (BGCD).

Algoritmus BGCD pracuje podobným způsobem jako GCD, ale místo operace modulo používá binární posuny, z těchto důvodů lze rychleji nalézt největšího společného dělitele.

Algoritmus 14 Binární Euklidův algoritmus (BGCD)

Require: $A = (a_n a_{n-1} ... a_0), B = (b_n b_{n-1} ... b_0)$
Ensure: $C = GCD(A, B)$
  $u = a, v = b, e = 1$
  while $u_0 = 0$ and $v_0 = 0$ **do**
    $u = u/2, v = v/2, e = 2 \times e$
  end while
  while $u \neq 0$ **do**
    while $u_0 = 0$ **do**
      $u = u/2$
    end while
    while $v_0 = 0$ **do**
      $v = v/2$
    end while
    if $u \geq v$ **then**
      $u = u - v$
    else
      $v = v - u$
    end if
  end while
  return $C = e \times v$

V následujícím odstavci je uveden algoritmus důležitý pro výpočty v RNS.

Rozšířený Euklidův algoritmus (EGCD)

Euklidův algoritmus lze rozšířit pro nalezení *Bézoutovy identity* (čísel $x, y \in \mathbb{Z}$ pro něž platí: $a \times x + b \times y = d$, kde $d =$GCD(a, b)). Formální zápis algoritmu je následující:

Algoritmus 15 Rozšířený Euklidův algoritmus (EGCD)

Require: $a, b \in \mathbb{N}, a \leq b$
Ensure: $d = gcd(a, b), x, y$
  $u = a, v = b, w = 1$
  $y_1 = 0, z = 0, y_2 = 1$
  while $u \neq 0$ **do**
    $q = \lfloor v/u \rfloor, r = v - q \times u, x = z - q \times w, y = y_2 - q \times y_1$
    $v = u, u = r, z = w, w = x, y_2 = y_1, y_1 = y$
  end while
  $d = v, x = z, y = y_2$
  return d, x, y

Uvedený algoritmus kromě nalezení GCD také vrátí členy x, y , jež jsou základem Bézoutovy identity. Tento algoritmus lze dále modifikovat do podoby pro výpočet multiplikativní inverze:

Algoritmus 16 Algoritmus výpočtu multiplikativní inverze

Require: p is prime, $a \in < 1, p - 1 >$
Ensure: $c = a^{-1} \, mod \, p$
  $u = a, v = p$
  $w = 1, z = 0$
  while $u \neq 1$ **do**
    $q = \lfloor v/u \rfloor, r = v - q \times u, x = z - q \times w$
    $v = u, u = r, z = w, w = x$
  end while
  return $c = x \, mod \, p$

Protože dělení čísel je výpočetně náročné, je v implementovaném systému RNS použit binární algoritmus pro nalezení multiplikativní inverze:

Algoritmus 17 Binární algoritmus výpočtu multiplikativní inverze

Require: $p = (p_n p_{n-1} ... p_0)$ is prime, $a = (a_n a_{n-1} ... a_0) \in < 1, p-1 >$
Ensure: $c = a^{-1} \, mod \, p$
  $u = a, v = p$
  $w = 1, z = 0$
  while $u \neq 1$ and $v \neq 1$ **do**
    while $u_0 = 0$ **do**
      $u = u/2$
      if $w_0 = 0$ **then**
        $w = w/2$
      else
        $w = (w+p)/2$
      end if
    end while
    while $v_0 = 0$ **do**
      $v = v/2$
      if $z_0 = 0$ **then**
        $z = z/2$
      else
        $z = (z+p)/2$
      end if
    end while
    if $u \geq v$ **then**
      $u = u - v, w = w - z$
    else
      $v = v - u, z = z - w$
    end if
  end while
  if $u = 1$ **then**
    return $c = w \, mod \, p$
  else
    return $c = z \, mod \, p$
  end if

Tento algoritmus je nejčastěji používaným algoritmem pro hledání inverzního čísla vzhledem ke své rychlosti.

Symetrické modulové třídy

Při výpočtech v modulo třídách je potřeba pracovat i se zápornými čísly, v tomto případě se zavádí množina symetrických reziduí modulo m. Pro symetrii vzhledem k nule musí být m liché číslo (číslo 2 a jeho násobky nesmí být modulem m).

Definice 4.2.22. Množinu symetrických zbytkových tříd vzhledem k m (kdy $m \neq 2$ a jeho násobkům) označíme jako $\mathbb{S}_m$ a hodnoty této množiny budou:

$$\mathbb{S}_m = \left\{ -\frac{m-1}{2}, .., -1, 0, 1, ..., \frac{m-1}{2} \right\}.$$

Pro zobrazení čísel v S_m se provede následující definice:

Definice 4.2.23. Zobrazení $\mathbb{S}_m \to \mathbb{Z}_m$ je definováno pro číslo $X \in \left\{1, ..., \frac{m-1}{2}\right\}$ jako:

$$/X/_m = |x|,$$

a pro číslo $X \in \left\{-\frac{m-1}{2}, ..., -1\right\}$ jako:

$$/m - X/_m = |-x|,$$

pro $X \in \{0\}$jako:

$$/0/_m = |0|.$$

Poznámka. Záporné číslo $|-x|$ se obvykle v odborném textu značí jako $|\bar{x}|$.

Příklad 4.2.24. Je dáno číslo $a = -3$ a $b = 2$ a bude proveden jejich součet v $\mathbb{Z}_7$.

$$\begin{gathered} a = /-3/_7 \equiv 4, \\ b = /2/_7 \equiv 2, \\ a+b = |4+2|_7 = |6|_7 = |1|_7. \end{gathered}$$

Což je správný výsledek.

4.3 Více - modulová aritmetika zbytkových tříd

I při použití čísel ze zbytkové třídy $\mathbb{Z}_m$ v jedno-modulové aritmetice může dojít k tzv. *pseudo přetečení* (např. při výpočtu příkladu 4.2.20), kdy mezivýsledek aritmetické operace je větší než modulo m (avšak výsledek tohoto přetečení je kongruentně správný). 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 s čísly jednotlivých $\mathbb{Z}_m$ nebude nikdy větší než součin všech jednotlivých modul $\mathbb{Z}_m$.

Definice 4.3.1. 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 4.3.2. 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 4.3.3. 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 4.3.4. *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 4.3.5. *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.

Definice 4.3.6. Podobně jako systém zbytkových tříd $\mathbb{Z}_\beta$ můžeme definovat také systém symetrických zbytkových tříd $\mathbb{S}_\beta$ reprezentovaný n-ticí:

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

Příklad 4.3.7. 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
$\mathbb{S}_M$	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)
$\mathbb{S}_\beta$	(0,0)	(1,1)	(-1,2)	(0,-2)	(1,-1)	(-1,0)	(0,1)	(1,2)
$\mathbb{Z}_M$	8	9	10	11	12	13	14	
$\mathbb{S}_M$	-7	-6	-5	-4	-3	-2	-1	
$\mathbb{Z}_\beta$	(2,3)	(0,4)	(1,0)	(2,1)	(0,2)	(1,3)	(2,4)	
$\mathbb{S}_\beta$	(-1,-2)	(0,-1)	(1,0)	(-1,1)	(0,2)	(1,-2)	(-1,-1)	

Tabulka 4.3.1: Reprezentace čísel ve zbytkových třídách a jejich systémech

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

Věta 4.3.8. *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 4.3.9. *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 4.3.10. 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). Na základě modula M je také možno předvídat pseudo přetečení, které je ukázáno na následujícím příkladě.

Příklad 4.3.11. Je dán bázový vektor$\beta = (5,7)$, pak $M = 35$ a čísla $a = 9, b$=8, převodem do vícemodulové aritmetiky se získá číslo:

$$| 9 |_\beta = (4,2)_\beta \quad a \quad | 8 | = (3,1)_\beta \ .$$

Nyní s čísly a, b se provede operace sčítání, odčítání a násobení:

$$\begin{gathered} | 9+8 |_\beta = (| 4+3 |_5, | 2+1 |_7) = (2,3)_\beta, \\ | 9-8 |_\beta = (| 4-3 |_5, | 2-1 |_7) = (1,1)_\beta, \\ | 9 \cdot 8 |_\beta = (| 4 \cdot 3 |_5, | 2 \cdot 1 |_7) = (2,2)_\beta. \end{gathered}$$

Při zobrazení zpět do $\mathbb{Z}_M$ se získá:

$$\begin{gathered} (2,3)_\beta = 17, \\ (1,1)_\beta = 1, \\ (2,2)_\beta = 2. \end{gathered}$$

Výsledek pro násobení je kongruentně správný, avšak pro reálné výpočty nepoužitelný, neboť velikost modula M je menší než provedený součin $a \cdot b$ ($35 < 72$) a tedy došlo k pseudo přetečení. Popsaná situace se řeší zvětšením modula M.

Shrnutí

Tato kapitola je teoretickým úvodem do aritmetiky zbytkových tříd. Nejprve jsou zde uvedeny souvislosti mezi konečnými tělesy a polynomy, následně je popsána jedno - modulová a více - modulová aritmetika zbytkových tříd. Také jsou zde popsány algoritmy pro nalezení inverzního čísla v daném modulu.

Kapitola 5

Paralelizace aritmetiky zbytkových tříd

Nevýhody standardně používaných aritmetik na počítači budou shrnuty v následujícím odstavci. Je dána uspořádaná množina $F \subset \mathbb{Q}$. Prvek $x \in F$ ve tvaru $x=(n_{k-1}n_{k-2}...n_1n_0,n_{-1}...n_{-l})_r$, kde $l,k \in \mathbb{N}$ (n_{k-1} označuje nejvyšší významovou číslici, n_{-l} nejméně významnou číslici a r základ soustavy). Číslo x bude číslo reprezentované dle určité normy (např. IEEE-754). Množina F je charakterizována následujícími vlastnostmi:

1. počet prvků množiny F je konstantní ($|F|$=*konst*.), tedy množinu nelze dále rozšířit,
2. prvky $x_i \in F$ nejsou rovnoměrně rozloženy podél souřadnicové osy např. IEEE - 754 (označí-li se vzdálenost mezi prvkem $x_i \in F$ a následujícím prvkem $x_{i+1} \in F$ jako L, pak $|L| \neq konst.$ pro každé x_i, x_{i+1}), kdy se zvětšující se vzdáleností od počátku osy se zvětšuje i vzdálenost mezi prvky x_i, x_{i+1},
3. množina F netvoří těleso $T(+,*)$,
4. zápis prvku je poziční. Prvek se dá zapsat jako: $x = \sum_{i=-l}^{k-1} n_i r^i$, kde r je základ soustavy (obvykle 10 nebo 2) a n_i je číslice na i-té pozici. Při výpočtech dochází k přenosům mezi vahami (např. u binární soustavy - přenosový bit),
5. v návaznosti na předcházející body také počet vah r^i prvku $x \in F$ množiny je omezen.

Výše popsané vlastnosti značně limitují přesné výpočty a použitím RNS je možné redukovat jejich nepříznivý dopad:

1. v RNS je možno dynamicky měnit velikost množiny F ($|F| \neq konst.$), rozsah je určen součinem modul (M),
2. v RNS se používají pouze celá čísla, tedy rozložení podél osy je rovnoměrné a vzdálenost mezi čísly je konstantní $|L|$=*konst*. pro všechny $x_i, x_{i+1} \in F$,
3. při použití kongruence může samostatné modulo m_i tvořit těleso $\mathbb{Z}_{m_i}(+,\cdot)$, avšak RNS vzhledem k součinu modul m_i bude tvořit vždy pouze okruh ($\mathbb{Z}_M$ je okruh $(+,\cdot)$),
4. číslo se dá rozložit podle modul m_i do samostatných prvků, kdy nad každým prvkem lze provádět aritmetické operace nezávisle a během výpočtu nedochází k přenosům mezi jednotlivými prvky,
5. vzhledem k dynamičnosti systému RNS není počet vah jednotlivých prvků omezen.

Vzhledem k tomu, že RNS je nepoziční systém, je problematika porovnávání čísel v RNS mnohem složitější než u standardně používaného pozičního systému a proto je mu věnována následující kapitolu.

5.1 Porovnávání čísel v RNS

Porovnávání čísel je během výpočtů nutné především z těchto důvodů:

- zjištění znaménka čísla,
- překročení intervalu čísel (*overflow*),
- určení, které číslo je větší menší (např. operace dělení).

Nejjednodušším způsobem je převod do pozičního systému a po provedení porovnání uskutečnit převod zpátky do systému RNS. Mezi tyto nejjednodušší techniky patří převod do pozičního *mixed radix system* (MRS), tento způsob byl důkladně popsán v [52], problémem tohoto převodu je především jeho sekvenčnost (i když byly publikovány možnosti urychlení převodu v $O(\log n)$). Další tradiční metodou je použití Čínské věty o zbytcích (CRT) (nevýhodou je především počet modulo M operací). Z důvodů časové náročnosti se začaly používat některé další způsoby porovnání, které budou představeny v následujících odstavcích.

Často používanou metodou (např. ve FIR filtrech [44]) je transformace z RNS do přibližného vyjádření v desetinném tvaru za použití aproximačního CRT (aCRT), výhodou tohoto přístupu je snížení režie oproti standardnímu CRT při opakovaných porovnáních na stanoveném počtu bitů. Číslo, které se porovnává se transformuje pomocí vztahu:

$$\frac{Y}{M}=\frac{(|y_0|_{m_0},|y_1|_{m_1},,...,|y_n|_{m_n})}{M}=\left|\sum_{i=0}^{n}\frac{|M_i^{-1}|_{m_i}}{m_i}y_i\right|_1, \tag{5.1.1}$$

kde $M_i=\frac{M}{m_i}$. Výhodou tohoto přístupu je možnost předvýpočtu výrazu $\frac{|M_i^{-1}|_{m_i}}{m_i}$ při použití tabulky, následně lze využít paralelní redukce (příloha B.3) a získat porovnání v logaritmickém čase.

Příklad 5.1.1. V následujícím příkladě se bude vycházet z [43], ke zjištění zda platí $X>Y$ nebo $Y>X$ (pro čísla X=$(|0|_3,|3|_5,|6|_7,|0|_8)=48_{10}$ a Y=$(|0|_3,|0|_5,|3|_7,|5|_8)=45_{10}$) se využije tabulka předem vypočtená podle 5.1.1 a dostane se:

$$\frac{X}{M}\approx|0.0000+0.2000+0.8571+0.0000|_1\approx 0.0571,$$
$$\frac{Y}{M}\approx|0.0000+0.0000+0.4286+0.6250|_1\approx 0.0536.$$

Protože platí $\frac{X}{M}>\frac{Y}{M}$, tak platí i $X>Y$, což odpovídá skutečnosti, neboť X=$48_{10}, Y=45_{10}$.

Pokud je maximální chyba aCRT ε, potom získaná hodnota RNS čísla nebude větší než $k\varepsilon$[28]. V uvedeném příkladě byla maximální chyba aCRT tabulky $\varepsilon=0.00005$ a použili se celkem $k=4$ číslice, tedy chyba hodnot je $4\varepsilon=0.0002$ a rozdíl hodnot je$|0.0571-0.4286|>0.0002$, tedy skutečně $X>Y$.

Dalším způsobem porovnání čísel v RNS je použití metody založené na diagonálním řazení čísel v n- dimenzionálním prostoru. Vzhledem k tomu, že jsou diagonály řazeny paralelně, lze je mezi sebou vzájemně porovnat. Metoda založená na této myšlence se nazývá *Sum of Quotients Technique*

(SQT) [13] a při jejím použití se nejprve zjistí suma kvocientů čísel $X = (|x_0|_{m_0}, |x_1|_{m_1},,..., |x_n|_{m_n})$ a $Y = (|y_0|_{m_0}, |y_1|_{m_1},,..., |y_n|_{m_n})$:

$$S_Q = \sum_{i=0}^{n} M_i,$$

a vypočte se hodnota s_i:

$$s_i = |-|m_i^{-1}|_{S_Q}|_{S_Q},$$

dále se bude vycházet při porovnávání z věty:

Věta 5.1.2. *Funkce:*

$$D(X) = |\sum_{i=0}^{n} s_i x_i|_{S_Q}, \tag{5.1.2}$$

je monotónně se zvyšující funkce.

Důkaz věty lze nalézt např. v [13]. Pokud platí $D(X) > D(Y)$, pak také $X > Y$ a naopak. Výjde-li $D(X) = D(Y)$, je nutné použít následující věty (důkaz také v [13]):

Věta 5.1.3. *Pokud $X < Y$ a $D(X) = D(Y)$, pak také $x_i < y_i$.*

Tato technika je prezentována na následujícím příkladě.

Příklad 5.1.4. Úkolem bude porovnat čísla X=$(|0|_3, |3|_5, |6|_7) = 48_{10}$ a Y=$(|0|_3, |0|_5, |3|_7) = 45_{10}$. Nejprve se vypočte S_Q:

$$S_Q = 35 + 21 + 15 = 71,$$

a pro jednotlivá s_i bude platit:

$$\begin{aligned} s_1 &= |-|3^{-1}|_{71}|_{71} = |-24|_{71} = 47, \\ s_2 &= |-|5^{-1}|_{71}|_{71} = |-57|_{71} = 14, \\ s_3 &= |-|7^{-1}|_{71}|_{71} = |-61|_{71} = 10. \end{aligned}$$

Nyní se určí $D(X)$:

$$D(X) = |47 \times 0 + 14 \times 3 + 10 \times 6|_{71} = 31.$$

Poté se určí $D(Y)$:

$$D(Y) = |47 \times 0 + 14 \times 0 + 10 \times 3|_{71} = 30.$$

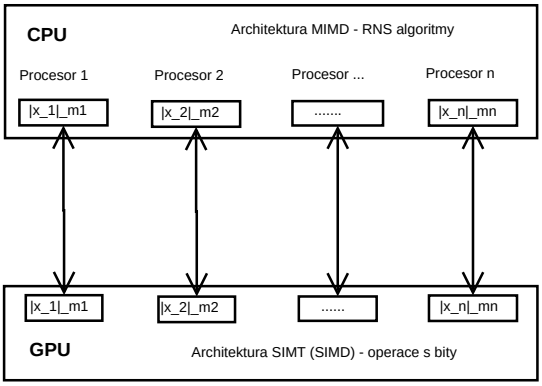
Vzájemným porovnáním se dospěje k nerovnosti $D(X) > D(Y)$.

Z funkce 5.1.2 je patrné, že lze provést paralelní redukci a v případě $D(X) \neq D(Y)$ získat porovnání v čase $O(log\, n)$ (v případě $D(X) = D(Y)$ závisí složitost na počtu porovnávaných prvků, tedy $O(log\, n + n)$, což je $O(n)$).

Poslední způsob porovnání dvou čísel v RNS o která zde bude uvedena, je metodika založená na středové funkci [43]. Středová funkce $C(M)$ mapuje čísla reprezentace RNS z množiny $\{0, M-1\}$ na číslo množiny $\{0, C(M)\}$, kde $C(M) << M$. Protože jsou čísla významně menší než číslo M, je bitové porovnání výrazně rychlejší, než v případě srovnávání nenamapovaných čísel.

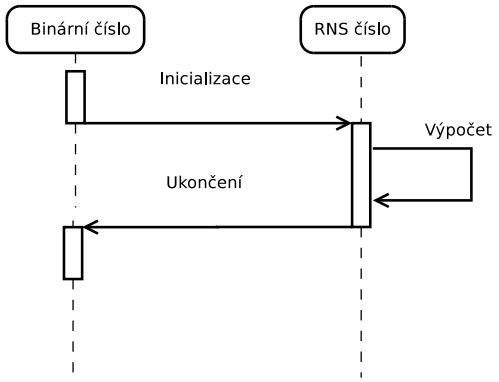
5.2 Paralelní systém FP RNS

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 5.2.1: 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 standartní aritmetice. Sekvenční diagram těchto činností je zobrazen na následujícím obrázku.

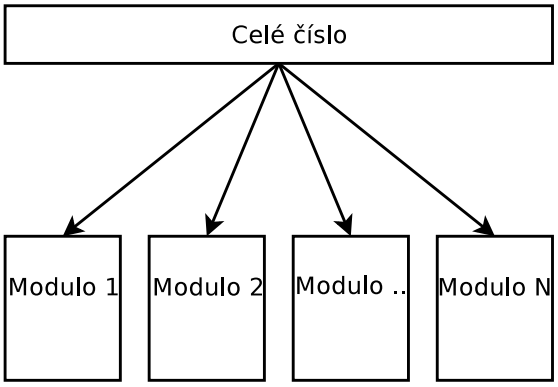


Obrázek 5.2.2: Paralelní systém RNS - komunikace

Jednotlivé fáze výpočtu budou popsány v následujících kapitolách.

5.3 Převod celých čísel do RNS (FP RNS)

Převod celých čísel se provádí v implementovaném systému pomocí operací modulo, jak naznačuje obrázek 5.3.1(na obrázku je znázorněn převod celého čísla do RNS (tento převod ne neliší od převodu do FP RNS):



Obrázek 5.3.1: Převod čísla ve standartním 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.

Protože aritmetická operace modulo u každého z m_i čísel je časově náročná, využívá se pro převod binárního čísla do RNS následující věty:

Věta 5.3.1. *Číslo $X \in \mathbb{Z}$ je možné zapsat vzhledem ke každému modulu m_i následovně:*

$$|(y_{k-1}y_{k-2}...y_1y_0)_2|_{m_i} = (|2^{k-1}y_{k-1}|_{m_i} + |2^{k-2}y_{k-2}|_{m_i} + ... + |2^1y_1|_{m_i} + |2^0y_0|_{m_i}).$$

Dle této věty se sestaví tabulka, podle které se dá převést binární číslo do reprezentace RNS. Tento převod bude ukázán v následujícím příkladě:

Příklad 5.3.2. Je dáno číslo X=65_{10} a pro převod do RNS s moduly $(3,5,7)$ se využije následující tabulka (první sloupec značí mocninu, druhý sloupec značí číslo po umocnění, třetí, čtvrtý a pátý sloupec značí zbytky po operaci modulo):

j	2^j	$< 2^j >_3$	$< 2^j >_5$	$< 2^j >_7$
0	1	1	1	1
1	2	2	2	2
2	4	1	4	4
3	8	2	3	1
4	16	1	1	2
5	32	2	2	4
6	64	1	4	1
7	128	2	3	2
8	256	1	1	4
9	512	2	2	1

Tabulka 5.3.1: Převod čísla ve standartním vyjádření do RNS pomocí tabulky

Číslo X je v binární formě zapsáno jako X=01000001$_2 = 2^6 + 2^0$, pak pro jednotlivá modula $m_i = \{3,5,7\}$ (sloupce) a $j = 6,0$ (řádky) bude platit:

$$\begin{aligned} x_0 &= X \, mod \, 3 = |1+1|_3 = 2, \\ x_1 &= X \, mod \, 5 = |4+1|_5 = 0, \\ x_2 &= X \, mod \, 7 = |1+1|_7 = 2. \end{aligned}$$

Což odpovídá vektoru RNS zapsanému v desítkové soustavě: $(|2|_3, |0|_5, |2|_7)$.

Další z možností urychlení převodu čísla do RNS [2] je rozdělení binárního čísla do bloků $B_{k-1}, .., B_0$, kde $|B_j| = p$ bitů a $k \triangleq n/p$. Tedy číslo X se dá vyjádřit jako:

$$X = \sum_{j=0}^{k-1} 2^{jp} B_j,$$

a redukované číslo X modulem m jako:

$$|X|_m = |\sum_{j=0}^{k-1} 2^{jp} B_j|_m = |\sum_{j=0}^{k-1} |2^{jp} B_j|_m|_m. \tag{5.3.1}$$

Příklad 5.3.3. Je dáno 12-bitové číslo např. $X = 1020 = 0011\,1111\,1100_2$, a cílem je získat číslo $|X|_5$. Číslo se rozdělí např. po 4 bitech (obvykle se volí kombinace nejbližší hardwarové reprezentaci). Pak se získají bloky: $B_2 = 0011_2 = 3$, $B_1 = 1111_2 = 15$, $B_0 = 1100_2 = 12$. Následně lze použít vzorec 5.3.1:

$$\begin{aligned} |X|_5 &= ||3 \times 2^8|_5 + |15 \times 2^4|_5 + |12 \times 2^0|_5|_5, \\ &= |3+0+2|_5, \\ &= 0. \end{aligned}$$

V implementovaném systému se používá pro modulování čísla Montgomeryho násobení, které je popsáno v kapitole 5.8.

Na závěr je dobré poznamenat, že např. v signálových procesorech jsou používána čísla tzv. vhodných forem (např. $2^n - 1$, $2n$, $2^n + 1$), kdy lze převody v binárním soustavě urychlit bitovými posuny.

5.4 Převod čísel z RNS (FP RNS)

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). CRT patří k nejstarším matematickým větám a její význam překračuje teorii čísel.

Čínská věta o zbytcích (CRT)

Věta 5.4.1. *Jsou dány přirozená čísla $m_1, m_2, ..., m_r$, která jsou po dvou nesoudělná ($GCD(m_i, m_j) = 1$ pro $i \neq j$). Pak pro libovolná celá čísla $a_1, a_2, ..., a_r$ má soustava rovnic:*

$$\begin{aligned} X &\equiv a_1 \, mod \, m_1, \\ X &\equiv a_2 \, mod \, m_2, \\ &... \\ X &\equiv a_r \, mod \, m_r. \end{aligned} \tag{5.4.1}$$

řešení X a toto řešení je určeno jednoznačně v modulu $M = m_1 \cdot m_2 \cdot ... \cdot m_r$.

Existuje mnoho různých důkazů této věty např. [64]. V této práci bude uveden důkaz, jehož součástí je algoritmus pro výpočet čísla X.

Důkaz. Důkaz lze rozdělit do dvou částí, v první se sestaví řešení pro dané kongruence, v druhé se ukáže, že řešení je jedinečné pro daná modula.

Nejprve se definuje součin modul $M = \prod_{i=1}^{r} m_i$, a pro každé $1 \leq k \leq r$ se určí hodnota $M_k = \frac{M}{m_k}$(M_k obsahuje všechny modula m_i s výjimkou modula m_k) Protože m_i je po dvojicích relativně prvočíselné, platí $\mathrm{GCD}(M_k, m_k)$ =$\mathrm{GCD}(m_1 m_2 ... m_{k-1} m_{k+1} ... m_r, m_k) = 1$. Následně se využije řešení lineární kongruence (pokud $\mathrm{GCD}(a,n) = d$, pak $ax \equiv b \, mod \, n$ má d řešení, které je dáno jedinečným řešením modulo $\frac{n}{d}$ kongruence $\frac{a}{d}x \equiv \frac{b}{d} \, mod \, \frac{n}{d}$), tedy lineární kongruence $M_k X \equiv 1 \, mod \, m_k$ má jedinečné řešení vzhledem k m_k.Potom se rovnice přeznačí do tvaru:

$$M_k x_k \equiv 1 \, mod \, m_k, \tag{5.4.2}$$

kde x_k je jedním z řešení rovnice 5.4.1. Nyní se sestrojí řešení uspokojující zároveň všechny kongruence tedy bude platit:

$$x' = a_1 M_1 x_1 + a_2 M_2 x_2 ... + a_r M_r x_r. \tag{5.4.3}$$

Vyhodnocuje se $x' \, mod \, m_k$ pro všechny $1 \leq k \leq r$. Pokud se moduluje řešení např.5.4.3 podle modula m_1:

$$x' = M_1 x_1 a_1 + M_2 x_2 a_2 ... + M_r x_r a_r \, mod \, m_1,$$

bude pro jednotlivé členy platit:

$$M_2 x_2 a_2 \equiv 0 \, mod \, m_1, M_3 x_3 a_3 \equiv 0 \, mod \, m_1, ..., M_r x_r a_r \equiv 0 \, mod \, m_1.$$

neboť členy $M_2, ..., M_r$ jsou z definice násobkem čísla m_1, s výjimkou členu M_1 který neobsahuje násobek m_1. Dosazením do rovnice 5.4.3 se tedy získá:

$$x' \equiv M_1 x_1 a_1 + 0 ... + 0 \, mod \, m_1 \equiv M_1 x_1 a_1 \, mod \, m_1.$$

S ohledem na rovnici 5.4.2 se získá $M_1 x_1 \equiv 1 \, mod \, m_1$. Rovnice 5.4.3 bude:

$$x' \equiv M_1 x_1 a_1 \, mod \, m_1 \equiv 1 a_1 \, mod \, m_1 \equiv a_1 \, mod \, m_1,$$

tedy x' uspokojuje první kongruenci $X \equiv a_1 \, mod \, m_1$, obdobně bude platit pro další kongruence:

$$x' \equiv a_k \, mod \, m_k,$$

a dokázali jsme 5.4.1.

Nyní se bude řešit jedinečnost řešení. Je dáno další řešení, které bude označeno y', pro všechny $k = 1, ..., r$ platí kongruence $x' \equiv y' \equiv a_k \, mod \, m_k$. Z těchto kongruencí bude platit vztah pro dělitelnost:

$$m_1 | (x' - y'), m_2 | (x' - y'), ..., m_r | (x' - y'), \tag{5.4.4}$$

protože jsou všechny m_i po dvojicích relativně prvočíselné dá se aplikovat věta říkající, že pokud platí $a|c$ spolu s $b|c$ a $\mathrm{GCD}(a,b) = 1$ pak $ab|c$. Přepisem 5.4.4 se získá:

$$(m_1 m_2 ... m_r) | (x' - y'),$$

tedy $x' \equiv y' \, (mod \, m_1 m_2 ... m_r)$, čímž je dokázána jedinečnost řešení.

□

Rovnice 5.4.3 se obvykle zapisuje ve tvaru (M_i^{-1} označuje x_i):

$$X = \sum_{i=1}^{r} a_i |M_i^{-1}|_{m_i} M_i, \tag{5.4.5}$$

Obvykle se omezuje interval řešení pouze na $X \in < 0, M-1 >$, neboť další řešení jsou kongruentní s tímto řešením, píše se rovnice 5.4.5 ve tvaru:

$$|X|_M = | \sum_{i=1}^{r} a_i |M_i^{-1}|_{m_i} M_i |_M, \tag{5.4.6}$$

což je nejpoužívanější tvar CRT.

Příklad 5.4.2. Je dáno číslo X=$(|0|_3, |3|_5, |6|_7)$ a cílem je jeho převod do desítkového vyjádření. K převodu se použije rovnice 5.4.6. Nejprve se určí M:

$$M = 3 \times 5 \times 7 = 105,$$

Pro jednotlivá M_i bude platit:

$$\begin{aligned} M_1 &= \tfrac{105}{3} = 35, \\ M_2 &= \tfrac{105}{5} = 21, \\ M_3 &= \tfrac{105}{7} = 15. \end{aligned}$$

Pro $|M_i^{-1}|_{m_i}$se získá:

$$\begin{aligned} |M_1^{-1}|_3 &= |35^{-1}|_3 = |2^{-1}|_3 = 2, \\ |M_2^{-1}|_5 &= |21^{-1}|_5 = |1^{-1}|_5 = 1, \\ |M_3^{-1}|_7 &= |15^{-1}|_7 = |1^{-1}|_7 = 1. \end{aligned}$$

Nyní lze provést součiny pro jednotlivé členy rovnice 5.4.6:

$$|0 \times 2 \times 35 + 3 \times 1 \times 21 + 6 \times 1 \times 15|_{105} = 48.$$

Což je správný výsledek.

Dle způsobu výpočtu je snadné určit, že nejnáročnější z hlediska rychlosti bude poslední součet členů výpočtu. Protože tento součet lze provést pomocí paralelní redukce bude časová složitost převodu při dostatečném počtu procesorů $O(log\, n)$.

Číselná soustava s různými bázemi (MRS)

Nejprve se charakterizují číselné poziční soustavy obecně. Číslo se v číselných soustavách vyjadřuje jako posloupnost číslic, z nichž každá má určitou váhu R_i a číslo je z rozsahu:

$$\{0, \prod_{i=1}^{n} R_i - 1\}.$$

Pak číslo A se dá vyjádřit jako:

$$A = a_n \prod_{i=1}^{n-1} R_i + ... + a_3 R_1 R_2 + a_2 R_1 + a_1, \tag{5.4.7}$$

kde pro a_i a index i platí:

$$a_i \in < 0, R_i - 1 >, i \in < 1, n >$$

Standardní soustavy jsou poziční soustavy se stejnou bází pro všechny pozice (např. desítková, dvojková) a každá pozici odpovídá násobek báze.

Příklad 5.4.3. Např. číslo A z rozsahu $\{0, 999\}$ se dá zapsat v desítkové soustavě $R_i = r = 10$ jako:

$$A = 10^2 a_3 + 10^1 a_2 + 10^0 a_1 \Longleftrightarrow A = (a_3, a_2, a_1),$$

kde a_3, a_2, a_1 jsou čísla na třetí, druhé a první pozici.

V následujícím odstavci budou uvedeny soustavy, kde pro každou pozici existuje jiná báze. Výhodou těchto soustav je především jejich poměrně snadná transformace do RNS a nazpět. Vzhledem k pozičnosti je možnost provést porovnání dvou různých čísel na základě srovnání číslic na stejných pozicích, čímž se zkracuje čas porovnávání.

Číselná soustava s modulo bázemi (MRNS)

V aritmetice RNS (FP RNS) se místo báze R_i bude používat modulo m_i. Pokud platí $m_{i+1} > m$ pak je číslo A zapsáno v soustavě označované jako MRNS. Transformace z RNS do MRNS je provedena v důkazu následující věty:

Věta 5.4.4. *Číslo $X = (x_n, x_{n-1}, ..., x_1)$ zapsané v RNS odpovídá číslu $A = (a_n, a_{n-1}, ..., a_1)$ zapsanému v MRNS.*

Důkaz. Nejprve se upraví číslo A dle rovnice 5.4.7 za použití modul m_i do následujícího tvaru:

$$A = a_n m_{n-1} m_{n-2} ... m_1 + ... + a_3 m_2 m_1 + a_2 m_1 + a_1, \tag{5.4.8}$$

pro získání složky x_1 se provede redukce podle m_1:

$$x_1 = |A|_{m_1} \quad = \quad a_1.$$

Aby se získal prvek a_2, přepíše se rovnice 5.4.8 do tvaru:

$$A - a_1 = a_n m_{n-1} m_{n-2} ... m_1 + ... + a_3 m_2 m_1 + a_2 m_1,$$

při redukci m_2 se získá:

$$|A - a_1|_{m_2} = |a_2 m_1|_{m_2},$$

při převodu na druhou stranu (násobením $|m_1^{-1}|_{m_2}$) vznikne:

$$||m_1^{-1}|_{m_2}(A - a_1)|_{m_2} = |a_2|_{m_2},$$

protože $a_2 < m_2$, lze napsat:

$$||m_1^{-1}|_{m_2}(A - a_1)|_{m_2} = a_2.$$

Neboť platí:

$$x_2 = |A|_{m_2} \quad = \quad a_2 m_1 + a_1 \ ,$$

je možné napsat:

$$a_2 = ||m_1^{-1}|_{m_2}(x_2 - a_1)|_{m_2}.$$

Tedy pro číslo $A = (a_n, a_{n-1}, ..., a_1)$ lze postup zobecnit:

$$\begin{array}{lcc} a_1 & = & x_1, \\ a_2 & = & ||m_1^{-1}|_{m_2}(x_2 - a_1)|_{m_2}, \\ a_3 & = & ||(m_1 m_2)^{-1}|_{m_3}(x_3 - (a_2 m_1 + a_1))|_{m_3}, \\ & ... & \\ a_n & = & ||(m_1 m_2 ... m_{n-1})^{-1}|_{m_n}(x_n - (a_{n-1} m_{n-2} ... m_1 + ... + a_2 m_1 + a_1))|_{m_n}. \end{array}$$

□

Příklad 5.4.5. Číslo zapsané v RNS X=$(|6|_7, |3|_5, |0|_3) = 48_{10}$. se převedeme do MRNS a následně do desítkové soustavy. Pro člen a_1 platí:

$$a_1 = 0.$$

Člen a_2 se vypočítá:

$$\begin{array}{rcc} a_2 & = & ||3^{-1}|_5 \times (3 - 0)|_5, \\ & = & |2 \times 3|_5, \\ & = & 1. \end{array}$$

Pro člen a_3 platí:

$$\begin{array}{rcc} a_3 & = & |(3 \times 5)^{-1}|_7 \times |(6 - (1 \times 3 + 0))|_7, \\ & = & |1 \times 3|_7, \\ & = & 3. \end{array}$$

Konečné číslo v MRNS vypadá následovně:

$$A = (3, 1, 0).$$

Pro převod do desítkové soustavy dosadíme do vzorce 5.4.8:

$$X = 3 \times 3 \times 5 + 1 \times 3 + 0 = 48_{10},$$

tedy převod skrze MRNS byl správný.

5.5 Rozšíření báze

Definice 5.5.1. Symbol m_E označuje redundantní modulo, jenž rozšiřuje bázi β. Báze obsahující modulo m_E se označuje jako báze β_E a je složena z vektoru $(m_0, ..., m_n, m_E)$.

Při převodech z a do RNS se modulo m_E nepoužívá a vytváří redundanci pouze za účelem rozšíření báze. Modulo m_E je také součástí implementovaného systému, neboť je využito při redukci číslem 10. V následující části se bude vycházet z přeznačené CRT 5.4.6:

$$\widetilde{X} = \sum_{i=0}^{i=n} |x_i M_i| M_i^{-1}|_{m_i}|_M,$$

Věta 5.5.2. *Výpočet nové báze lze provést pomocí vzorce:*

$$q = ||M^{-1}|_{m_E}(|\widetilde{X}|_{m_E} - x_E)|_{m_E},$$

kde x_E značí zbytek po redukci m_E

Důkaz. Do vzorce 5.4.6 se dosadí $X_i = M_i|M_i^{-1}|_{m_i}$ a použije se zbytku definovaného jako číslo po celočíselném dělení, vzorec pro CRT pak lze přepsat do pseudo CRT (pCRT):

$$X = \sum_{i=1}^{n} |x_i X_i|_M - qM, \tag{5.5.1}$$

pro některá $q \in \mathbb{Z}$, zkráceně se zapisuje jako $X = \widetilde{X} - qM$, pro něž platí $|\widetilde{X} - qM| < M$.Vzhledem k tomu, že každý člen $|x_i X_i|$ z rovnice je menší než M, pak také platí, že $\sum_{i=1}^{n} |x_i X_i|_M < nM$, a vzorec lze přepsat do tvaru:

$$X \quad = \quad \textstyle\sum_{i=1}^{n} |x_i X_i|_M - qM < nM - qM \ ,$$

Nyní se přistoupí k volbě redundantního modula m_E (nová báze), takového pro které platí: $m_E > n$ a proměnnou $x_E = |X|_{m_E}$. Pak se přepíše vzorec 5.5.1 do tvaru:

$$|X|_{m_E} \quad = \quad |\textstyle\sum_{i=1}^{n} |x_i X_i|_M - qM|_{m_E} = |\widetilde{X} - qM|_{m_E} \ .$$

Pokud se převede M na druhou stranu, tak se získá:

$$|q|_{m_E} = ||M^{-1}|_{m_E}(|\widetilde{X}|_{m_E} - x_E)|_{m_E},$$

protože $q < n$ a $m_E > n$, platí $|q|_{m_E} = q$ a rovnici lze přepsat do tvaru:

$$q = ||M^{-1}|_{m_E}(|\widetilde{X}|_{m_E} - x_E)|_{m_E}, \tag{5.5.2}$$

což je tvar pro výpočet zbytku v nové bázi. □

Výpočet nové báze se zbytkem je proveden v následujícím příkladě:

Příklad 5.5.3. Číslo $X = (|2|_3, |2|_5, |2|_7) = 2_{10}$ se rozšíří o bázi $m_E = 4$ a následně se dopočte modulo.

$$\begin{aligned} M_1 = 35, \quad & |M_1^{-1}|_3 = |2^{-1}|_3 = 2, \\ M_2 = 21, \quad & |M_2^{-1}|_5 = |1^{-1}|_5 = 1, \\ M_3 = 15, \quad & |M_3^{-1}|_7 = |1^{-1}|_7 = 1. \end{aligned}$$

Dále se vypočte M a $|M^{-1}|_{m_E}$:

$$M = 105, \quad |M^{-1}|_4 = |1^{-1}|_4 = 1.$$

Pro výpočet $\widetilde{X}$ platí:

$$\begin{aligned} \widetilde{X} &= |2 \times 35 \times 2|_{105} + |2 \times 21 \times 1|_{105} + |2 \times 15 \times 1|_{105}, \\ &= 35 + 42 + 30, \\ &= 107. \end{aligned}$$

protože $qM = 105$, lze určit číslo X jako:

$$X = 107 - 105 = 2,$$

Pro q dle vzorce 5.5.2 a $x_E = |105|_4 = 1$ bude platit:

$$\begin{aligned} q &= |1 \times (|107|_4 - 1)|_4, \\ &= 2, \end{aligned}$$

tedy modulo pro rozšířenou bázi m_E je rovno číslu 2 a číslo v RNS s novou bází $X = (|2|_3, |2|_5, |2|_7, |2|_4) = 2_{10}$.

5.6 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 C.3 a C.4.

RNS algoritmus pro sčítání

Zobecněný algoritmus pro sčítání čísel v RNS bude vypadá následovně:

Algoritmus 18 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. Tento návrh je založen na generování a předávání přenosu skrze jednotlivé stupně sčítaček. Výhodou tohoto způsobu výpočtu je propagace všech přenosů paralelně, čímž se zabrání čekání na správný přenos generovaný z předcházející sčítačky. Přenos c_{i+1}na i-tém stupni sčítaček je dán rovnicí:

$$c_{i+1} = x_i y_i + (x_i \oplus y_i) c_i,$$

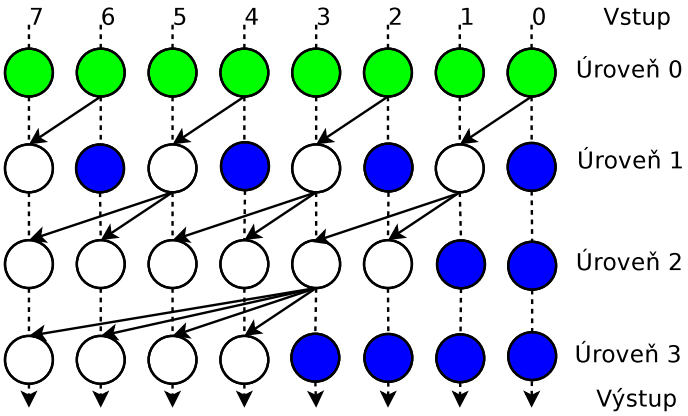
Tuto rovnice lze rozložit na generování přenosu G na stupni i a propagaci přenosu P na stupni i.

$$\begin{aligned} G_i &= x_i y_i, \\ P_i &= x_i \oplus y_i, \\ c_{i+1} &= G_i + P_i c_i. \end{aligned}$$

Výše popsaný výraz umožňuje výpočet všech přenosů paralelně, např. pro 4-bitovou sčítačku platí:

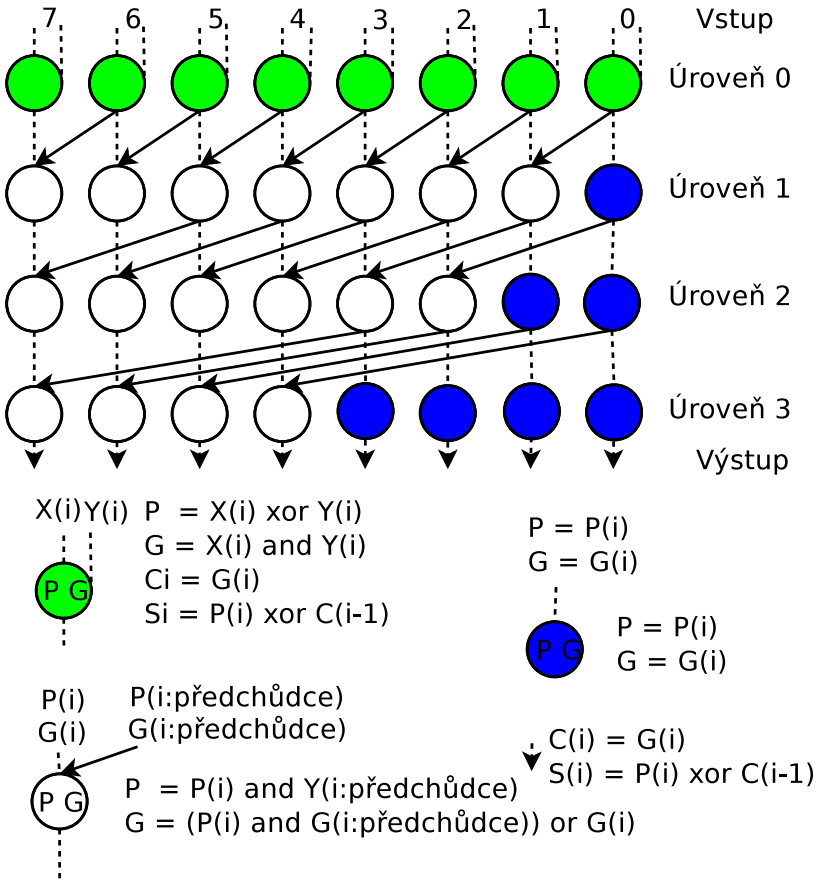
$$\begin{aligned} c_0 &= c_0, \\ c_1 &= G_0 + c_0 P_0, \\ c_2 &= G_1 + G_0 P_1 + c_0 P_0 P_1, \\ c_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2, \\ c_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3. \end{aligned}$$

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. Dle Ladner-Fisher návrhu jsou sčítačky seřazeny způsobem naznačeným na obrázku:



Obrázek 5.6.1: Binární sčítání: Ladner - Fisher

Schéma binární sčítačka Koggle - Stone vypadá následovně:



Obrázek 5.6.2: Binární sčítání: Kogge - Stone

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 $O(logn)$. 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 5.6.1: 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.

5.7 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 4.2.14.*

RNS algoritmus pro odečítání

Zobecněný algoritmus pro odečítání čísel v RNS vypadá následovně:

Algoritmus 19 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 5.6.1, z nichž je patrná neefektivita výpočtu v případě použití GPU.

5.8 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 rychlejší násobení a redukci, které jsou shrnuty v následujících odstavcích.

Barettovo modulární násobení

Toto modulární násobení bylo představeno v [5]. Před samotným výpočtem je potřeba vypočítat proměnnou $\mu = \lfloor b^{2k}/M \rfloor$, což je velmi omezující podmínka vzhledem k použití operace děleno. Hlavní myšlenkou tohoto algoritmu je využití dvou dělení násobku báze b, které lze provést pomocí bitových posunů. Algoritmus Barettova násobení je následující:

Algoritmus 20 Barrettův algoritmus

Require: $k = \lfloor log_b M \rfloor + 1, 0 \leq Z < b^{2k}, q = \lfloor b^{2k}/M \rfloor, Z = A \times B$
Ensure: $C = Z \bmod M$
  $\hat{q} = \lfloor \lfloor Z/b^{k-1} \rfloor \times \mu / b^{k+1} \rfloor$
  $C = Z \bmod b^{k+1} - \hat{q} \times M \bmod b^{k+1}$
  if $C < 0$ **then**
    $C = C + b^{k+1}$
  end if
  while $C \geq M$ **do**
    $C = C - M$
  end while
  return C

I přes své zjevné nevýhody (nutnost dělení) se Barettův algoritmus používá především pro svou jednoduchost.

Motgomeryho násobení a redukce

Motgomeryho násobení [40] patří v současnosti k nejpoužívanějším algoritmům pro modulární násobení, jeho použití přesahuje teorii čísel a používá se např. pro výpočet veřejného a soukromého klíče v kryptografii [87] (výpočet modula součinu dvou přirozených čísel s velkým exponentem). Hlavní myšlenkou je přesun výpočetních operací do Montgomeryho prostoru a po provedení výpočtu zpět. Nevýhodou tohoto způsobu výpočtu je především potřeba multiplikativní inverze. Součástí algoritmu je také výpočet proměnné M' z Bézoutovy identity ($r \cdot r^{-1} - M \cdot M' = 1$) [72]. Nejprve bude uveden algoritmus pro výpočet Montgomeryho produktu (MP):

Algoritmus 21 Montgomeryho produkt MP(A, B)

Require: $M', M, r = 2^k, \bar{A} = A \cdot r \bmod M, \bar{B} = B \cdot r \bmod M$
Ensure: $u = \bar{A} \cdot \bar{B} \cdot r^{-1} \bmod M$
  $t = \bar{A} \cdot \bar{B}$
  $m = t \cdot M' \bmod r$
  $u = (t + m \cdot M) \cdot r^{-1}$
  if $u \geq n$ **then**
    return $u - n$
  else
    return u
  end if

Myšlenkou tohoto algoritmu je bitový posun pomocí proměnné r provedený v Montgomeryho prostoru. Algoritmus je volán jako funkce z algoritmu Montgomeryho redukce. Následuje algoritmus nazývaný Mongomeryho redukce (MR).

Algoritmus 22 Montgomeryho redukce MR

Require: $M, r = 2^k, A, B, M'$
Ensure: $C = A \cdot B \bmod M$
  $\bar{A} = A \cdot r \bmod M$
  $\bar{B} = A \cdot r \bmod M$
  $\bar{C} =$MP$(\bar{A}, \bar{B})$
  $C =$MP$(\bar{C}, 1)$
  return C

Výše uvedený algoritmus se použije pro výpočet $A \times B \bmod M$. Z formálního zápisu algoritmu je

zřetelné, že k volání MP dochází dvakrát, jednou z důvodu převodu čísel A, B do Montgomeryho prostoru, podruhé z důvodu transformací z Montgomeryho prostoru do standardního vyjádření. Časová složitost algoritmu je závislá především na implementaci násobení použitého v algoritmu, tedy obecně ji nemění a lze vyjádřit jako $O(M(n))$, kde $M(n)$ vyjadřuje složitost použitého násobení.

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 23 Kop algoritmus pro RNS

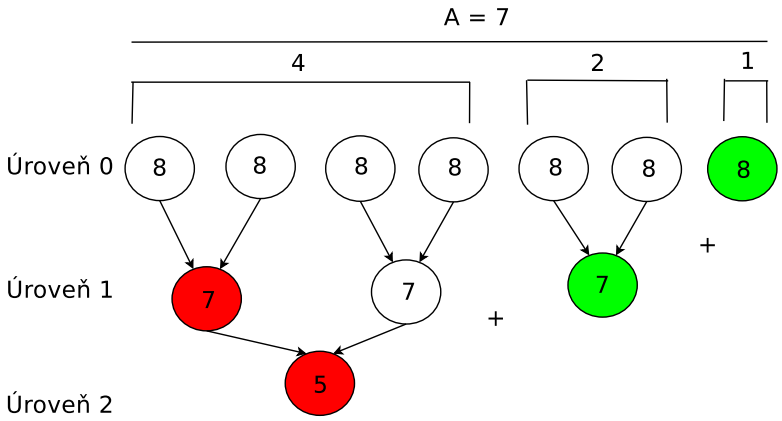
Require: $A = (a_n a_{n-1} ... a_0)_2 < M, B = (b_n b_{n-1} ... b_0)_2 < M$
Ensure: $C = A \times B \bmod M$
1: **if** $A = 0$ or $B = 0$ **then**
2: **return** $C = 0$
3: **end if**
4: **if** $A > B$ **then**
5: SWAP(A, B)
6: **end if**
7: $rtree = B$,$ltree = 0$
8: $level = 0$, $r = 0$
9: **if** $A > 1$ **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 23 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 stomu se zbytkem po stromech nižší výšky (řádek 32).

Průběh algoritmu je ukázán na následujícím příkladě:

Příklad 5.8.1. 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.



Obrázek 5.8.1: Příklad násobení modulo čísel Kop algoritmem pro $(8 \cdot 7)$ mod 9

Výhodou navrženého 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:

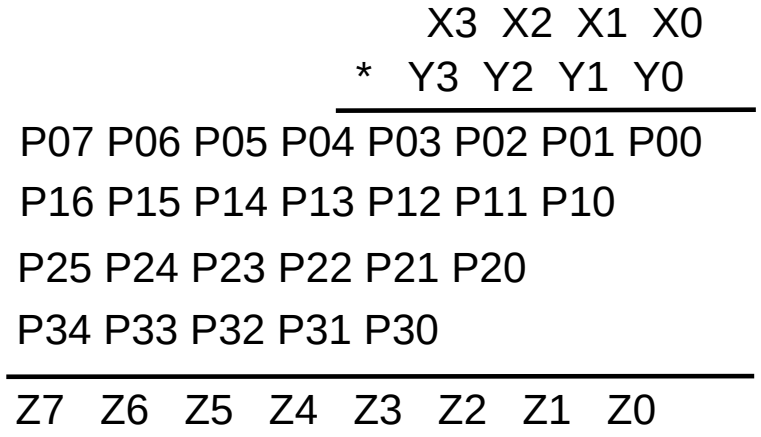
	M = 31 5 bitů	M = 61 6 bitů	M = 127 7 bitů	M = 509 8 bitů	M = 1021 9 bitů
Barrett: avg.(ms)	101	486	2295	44968	199250
Montgomery: avg.(ms)	101	461	2285	44621	193272
Kop: avg.(ms)	79	375	1846	37532	166885

Tabulka 5.8.1: Časy výpočtu Kop algoritmu pro $(A \cdot B) \bmod M$

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 [57]. 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 5.8.2(4-bitové násobení).

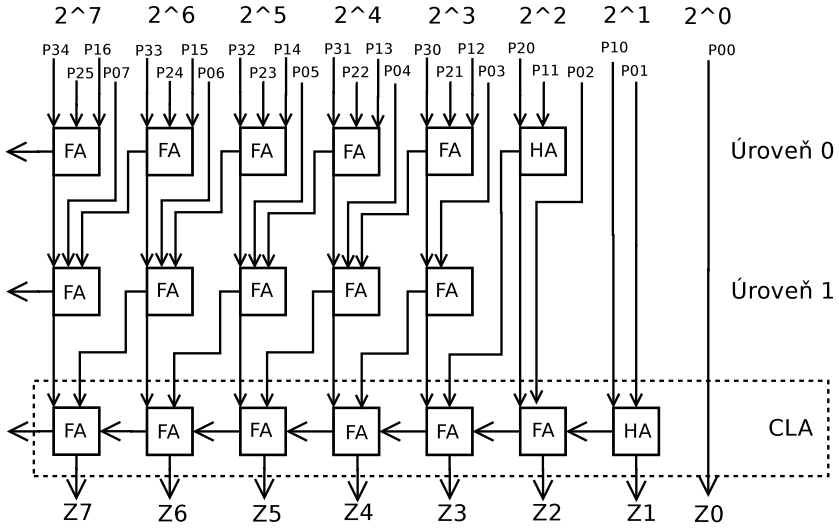


Obrázek 5.8.2: Binární násobení: Wallace tree - rozdělení bitů do skupin

Druhým krokem je opakovaná redukce vzniklých produktů pomocí plných sčítaček (FA) a poloviních sčítaček (HA) do stupně, ve kterém existují pouze dvojice přenášených bitů (obrázek 5.8.3).

Třetím krokem je součet dvojic bitů pomocí CLA sčítaček (obrázek 5.8.3).

V provedené implementaci lze považovat každou FA a HA sčítačku za mikroprocesor a každý stupeň výpočtu lze provést paralelně. Proto redukce stupňů výpočtu zobrazených na obrázku zabere pouze $O(log\,n)$ času a čas potřebný k součtu posledního stupně paralelní redukcí je také $log\,n$.



Obrázek 5.8.3: Binární násobení: Wallace tree - průběh výpočtu

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 5.8.2: 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).

5.9 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

Škálování

Škálování se používá např. pro udržení čísla ve zvoleném rozsahu (zabrání se tak přetečení). Bude se vycházet z následující věty:

Věta 5.9.1. *Číslo* $A = (|a_1|_{m_1}|a_2|_{m_2}...|a_n|_{m_n})$ *se škáluje číslem K následovně:*

$$A = B \cdot K + |A|_K,$$

úpravou předchozího vztahu vznikne škálované číslo $B = (|b_1|_{m_1}|b_2|_{m_2}...|b_n|_{m_n})$*, pro něž platí*

$$B = (A - |A|_K) \cdot K^{-1}.$$

Proces škálování je popsán následujícím algoritmem (funkce *base_extension()* označuje rozšíření báze, funkce *to_decimal()* označuje převod čísla do desítkové soustavy):

Algoritmus 24 Algoritmus škálování

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: $C = \frac{A}{B} \bmod M$
B_{10}=*to_decimal*(B_{RNS})
$residue_{RNS}$=*base_extension*(A_{RNS},B_{10})
A_{RNS}=A_{RNS}-$residue_{RNS}$
$C = A_{RNS} \cdot B_{RNS}^{-1}$

Zjednodušeně lze popsat výše uvedený algoritmus jako zjištění zbytku po dělení čísel A a B, odečtení zbytku po dělení od čísla A (pomocí rozšíření báze) a vynásobení zmenšeného čísla A číslem B^{-1}.

Příklad 5.9.2. Číslo $A = (|1|_3|0|_5|2|_7) = 100_{10}$ se bude dělit číslem $B = (|2|_3|1|_5|4|_7) = 11_{10}$. Protože tato čísla spolu nejsou dělitelné beze zbytku ($A \bmod B \neq 0$), je nutné nejprve zjistit zbytek (pomocí rozšíření báze $mod\,11$), což je číslo 1, které lze zapsat v RNS jako $R = (|1|_3|1|_5|1|_7) = 1$ (z rozsahu $\{0, 10\}$). Tento zbytek se odečte od čísla A:

$$A - R = (|1|_3|0|_5|2|_7) - (|1|_3|1|_5|1|_7) = (|0|_3|4|_5|1|_7),$$

následně se provede součin:

$$(A - R) \times B^{-1} = (|0|_3|4|_5|1|_7) \times (|2|_3|1|_5|2|_7),$$

Výsledkem je:

$$(|0|_3|4|_5|2|_7) = 9_{10}.$$

což je správný výsledek (multiplikativní inverzní číslo k číslu 0 neexistuje, značí násobek).

Celočíselné dělení

Celočíselné dělení se nejsnadněji provádí převodem čísla do pozičního systému (MRNS nebo standardní reprezentace), vydělením a zpětným převodem do RNS. Protože je však tento převod časově i prostorově náročný, došlo k představení několika algoritmů, které se snaží transformace z RNS a do RNS omezit.

Celočíselné dělení lze popsat jako:

$$Z = \lfloor \frac{X}{Y} \rfloor,$$

kdy Z je rovno zaokrouhlené celočíselné nižší hodnotě (značení $\lfloor \rfloor$). V následujícím odstavci bude uveden postup založený na iteračním hledání podílu X'/Y' [20]. Popisované celočíselné dělení je založeno na iteracích v závislosti na proměnné G=GCD(Y,M). První iterace dělení se použije, pokud GCD$(Y,M) > 1$:

Algoritmus 25 Gamberger - iterace celočíselné dělení v RNS pro $GCD(Y,M) > 1$

$$X'_{i+1} = \lfloor \frac{X}{G} \rfloor,$$

$$Y'_{i+1} = \frac{Y}{G}.$$

Druhá iterace dělení se použije, pokud GCD$(Y,M) = 1$:

Algoritmus 26 Gamberger - iterace celočíselné dělení v RNS pro $GCD(Y,M) = 1$

$$X'_{i+1} = \lfloor \frac{X \times |Y^{-1}|_M}{M} \rfloor,$$

$$Y'_{i+1} = \lfloor \frac{Y \times |Y^{-1}|_M}{M} \rfloor = \frac{Y \times |Y^{-1}|_M - 1}{M}.$$

Uvedený způsob výpočtu je demonstrován na následujícím příkladě:

Příklad 5.9.3. Jsou dány dvě čísla, číslo $X = 98$ a číslo $Y = 30$ a následující báze vektoru RNS $m_1 = 3, m_2 = 5, m_3 = 7, M = 105$. Cílem je provést celočíselné dělení (pro názornost použijeme poziční soustavu). Nejprve se zjistí GCD$(30,105) = 15$ a následně se provede iterace:

$$\begin{array}{rcccl} X' & = & \lfloor \frac{98}{15} \rfloor & = & 6, \\ Y' & = & \frac{30}{15} & = & 2. \end{array}$$

Provede se další test $GCD(2,105) = 1$ a následuje iterace:

$$\begin{array}{rcccl} X' & = & \lfloor \frac{6 \times 53}{105} \rfloor & = & 3, \\ Y' & = & \frac{2 \times 53 - 1}{105} & = & 1. \end{array}$$

Protože $Y' = 1$, ukončí se iterace a výsledkem celočíselného podílu je číslo $Z = 3$, což odpovídá skutečnosti.

Mezi další známé techniky pro RNS dělení patří Hitz a Kaltofen algoritmus [26] vyhodnocující $X/Y \bmod M$ jako:

$$(\frac{X}{M} \lfloor \frac{M}{Y} \rfloor) \bmod M,$$

kde operace M/Y je realizovaná pomocí Newtonovy iterační metody.

Poslední dvě metody pro celočíselné dělení, které zde budou pro úplnost uvedeny, byly publikovány v knize [52]. První je založena na převodu do MRNS, provedení dělení a zpětné transformace do RNS. Druhá je založena na iteračním výpočtu rovnice:

$$X_{i+1} = (X - Q_i \times Y),$$

kde se dle daných vlastností volí velikost zbytku Q_i.

5.10 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 5.10.1, 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 5.10.1: Standartní 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 5.10.2, 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 5.10.2: RNS bitové sčítání

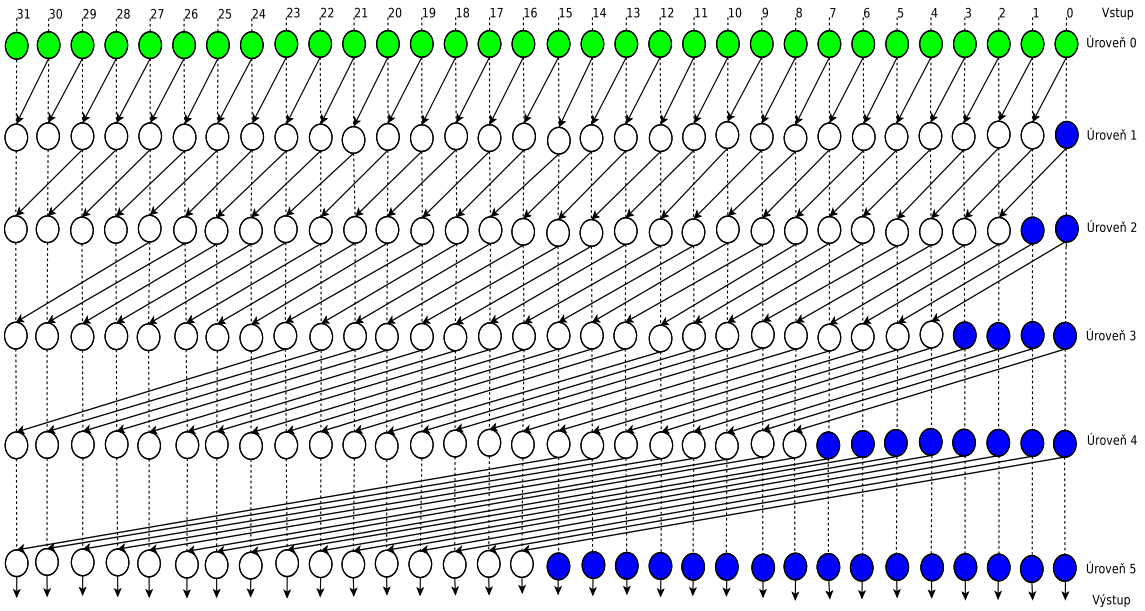
Přestože se časová složitost bitových operací ve standardní aritmetice zvyšuje s délkou slova n pouze nepatrně ($O((log\,n))$, při velkém počtu operací s dlouhými bitovými slovy výhoda paralelizace výpočtu pomocí RNS roste (rychlejší výpočet, jednodušší výpočetní logika pro aritmetické operace a tedy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

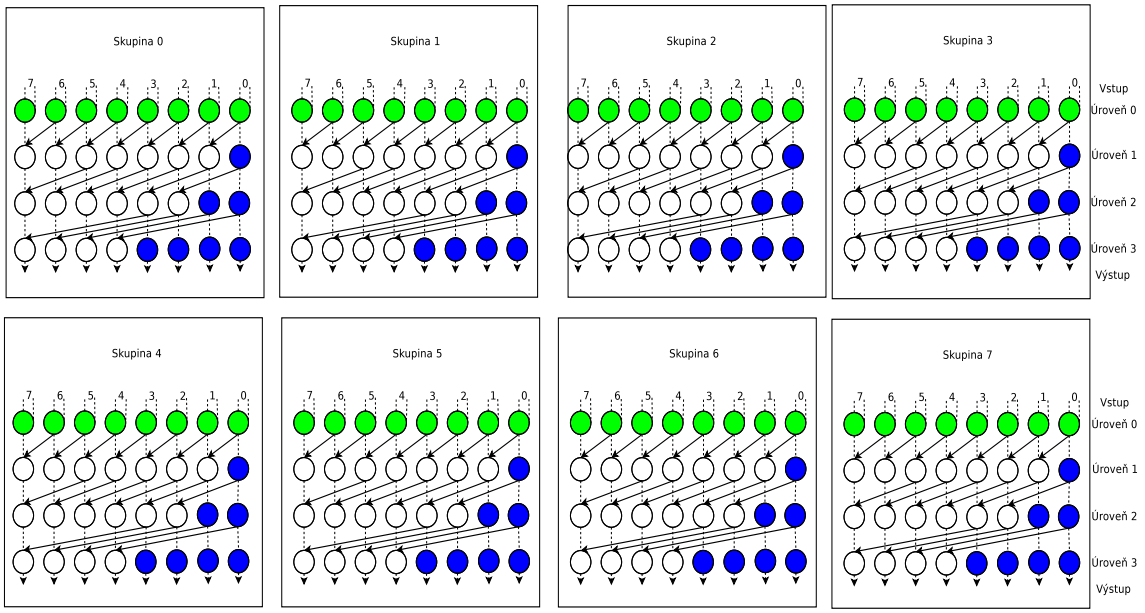
Výše uvedené zásady jsou použity v následujícím příkladě.

Příklad 5.10.1. 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 5.6.2, tato sčítačka pro 32 bitů (odpovídá maximálnímu prvočíslu v bázi β) bude vypadat následovně:



Obrázek 5.10.3: 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 5.10.4: RNS architektura složená z 8 bitových sčítaček Koggle - Stone

Porovnáním obrázku 5.10.3 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 diskutovaných v kapitole 8, lze RNS systém dále urychlit.

5.11 Urychlení výpočtu FP RNS na více - procesorovém systému MIMD

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 5.11.1: Urychlení aritmetických operací na více-procesorovém systému RNS

Z tabulky 5.11.1 je patrné, že při zvětšujícím se počtu aritmetických operací, roste i celkové zrych-

lení (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 zmíněny všechny tři fáze, jimiž výpočet ve FP RNS prochází. Nejprve je zde uveden převod čísel do FP RNS a způsoby jak lze převod urychlit. Následně jsou uvedeny výpočty ve FP RNS (srovnání čísel, rozšíření báze, operace sčítání, odčítání, násobení) na architekturách SIMT a MIMD. Nakonec je uvedena časově nejnáročnější fáze, což je převodem do standardní reprezentace. V závěru kapitoly se také nachází srovnání FP RNS a standardní reprezentace na bitové úrovni a urychlení výpočtu FP RNS na paralelní architektuře MIMD.

Kapitola 6

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.

6.1 Standardní reprezentace celých čísel v počítači

Základní celočíselné typy se často označují jako ordinální. Obecně platí, že hodnoty ordinálního typu tvoří lineárně uspořádanou množinu s jasně definovaným předchůdcem a následníkem. Výhodou toho uspořádání jsou především přesně stanovené hranice množiny, při jejichž překročení dochází k přetečení. Při respektování hranic tvoří tato množina čísel algebraický okruh.

Často jsou čísla [27] dělena na znaménkové (kdy v jedné půlce bitového rozsahu se nachází záporná čísla a v druhé půlce čísla kladná, číslo a je v rozsahu $-2^{n-1} \leq a \leq 2^{n-1}$ a první bit určuje znaménko), a bezznaménková (tyto čísla obsahují pouze čísla kladná, a je v rozsahu: $0 \leq a \leq 2^n - 1$).

Nejčastěji se setkáváme s následujícími ordinálními datovými typy, jejichž vlastnosti jsou zachyceny v tabulce:

Datový typ	Velikost v paměti	Rozsah (bez znaménka)	Rozsah (se znaménkem)
boolean	1 bit	$< 0, 2^1 - 1 >$	
byte	8 bitů (1 bajt)	$< 0, 2^8 - 1 >$	$< -2^7, 2^7 - 1 >$
short	16 bitů (2 bajty)	$< 0, 2^{16} - 1 >$	$< -2^{15}, 2^{15} - 1 >$
integer_32	32 bitů (4 bajty)	$< 0, 2^{32} - 1 >$	$< -2^{31}, 2^{31} - 1 >$
integer_64	64 bitů (8 bajtů)	$< 0, 2^{64} - 1 >$	$< -2^{63}, 2^{63} - 1 >$

Tabulka 6.1.1: Ordinální datové typy

Výhodou těchto typů je především jejich přesnost, kdy absolutní a relativní chyba výpočtu v daném intervalu je nulová.

6.2 Standardní reprezentace čísel s pohyblivou čárkou v počítači

Při numerických výpočtech [9] se v počítači nejčastěji používá reprezentace čísel s pohyblivou čárkou podle normy *IEEE - 754* [60]. Symbolem q se značí základ číselné soustavy (u počítačů většinou 2, nebo 16), dále je potřebný exponent $e \in \mathbb{Z}$ (rozsah je dán intervalem $-e_{min} \leq e \leq e_{max}$) a mantisa (posloupnost jednotlivých číslic d). Konečné číslo vyjádřené v *semilogaritmickém tvaru* vypadá následovně:

$$x = \pm 0, d_1 ... d_n d_{n+1} ... \times q^e.$$

Protože čísla v počítači mohou být uložena pouze v konečném tvaru, používá se jejich zaokrouhlování. Zaokrouhlování se provádí dvěma způsoby, prvním z nich je odseknutí nepotřebných číslic (*chopping*):

$$\tilde{x}_{chop} = \pm 0, d_1 ... d_n \times q^e,$$

kde pro absolutní chybu platí:

$$|x - \tilde{x}_{chop}| \quad = \quad 0.d_{n+1} ... \times q^{e-n},$$

tedy chyba je v rozsahu: $| x - \tilde{x} | \leq q^{e-n}$.

Druhým způsobem je zaokrouhlení poslední číslice d_n dle následujících číslic (*rounding*):

$$\tilde{x}_{round} = \pm 0, d_1 ... d_n \times q^e,$$

kde pro absolutní chybu platí:

$$|x - \tilde{x}_{round}| \quad = \quad 0.d_{n+1} ... \times q^{e-n},$$

ale chyba je v rozsahu $| x - \tilde{x} | \leq q^{e-n}/2$.

Srovnáním obou přístupů je zřejmé, že v nejhorším možném případě bude absolutní chyba zaokrouhleného čísla pomocí choppingu dvakrát větší než v případě použití roundingu.

Norma IEEE - 754

Obvykle se číslo X v této normě vyjadřuje ve tvaru [53]:

$$X = (-1)^{znaménko} \times 2^{exponent - bias} \times mantisa,$$

kde hodnota *bias* se používá z důvodu kladnosti exponentu a vypočte se dle vztahu *bias*=$2^{eb} - 1$, kde *eb* je počet bytů vyhrazených pro exponent.

Pro tuto normu je 32 bitové a 64 bitové číslo v plovoucí řádové čárce uloženo v paměti počítače následovně:

Datový typ	Počet bitů	Znaménko	Mantisa	Exponent	Exponent (rozsah)
float	32 bitů (4 bajty)	1 bit	23 bitů	8 bitů	$< -2^8 - 2, 2^8 - 1 >$
double	64 bitů (8 bajtů)	1 bit	53 bitů	11 bitů	$< -2^{11} - 2, 2^{11} - 1 >$

Tabulka 6.2.1: Reprezentace čísel v IEEE - 754

Číslo může být v normalizované formě (první bit mantisy roven 1), nebo v nenormalizované formě (první bit mantisy roven 0), rozsah čísel pro 32-bitové číslo je následující:

Datový typ	Normalizace	nejmenší číslo	největší číslo
float	ano	$\pm 2^{-126}$	$\pm(2-2^{-23})\times 2^{127}$
float	ne	$\pm 2^{-23}\times 2^{-126}$	$\pm(1-2^{-23})\times 2^{-126}$

Tabulka 6.2.2: Rozsah čísel v IEEE - 754

Přesnost čísla je dána počtem prvků mantisy pro 32 - bitové číslo (23+1) bitů a 64 - bitové číslo (53+1).

Datový typ	Přesnost (počet cifer)
float	$log_{10}(2^{23+1}) = 7.22$
double	$log_{10}(2^{53+1}) = 15.95$

Tabulka 6.2.3: Přesnost čísel v IEEE 754

Tato norma kromě uložení čísel obsahuje i pravidla implementací operací pro práci s těmito čísly a konverze mezi jednotlivými jednoduchými typy.

6.3 Reprezentace celých čísel v RNS

Aritmetika zbytkových tříd je založena na celočíselných zbytcích 4.2.1, 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 (4.2.23), 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 6.3.1. 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 6.3.1: 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:

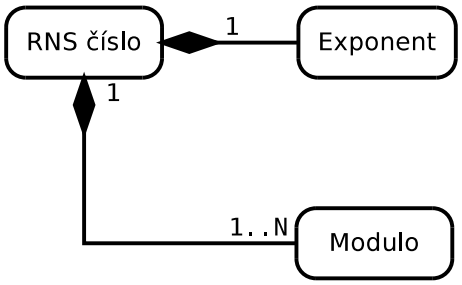
mod 16				mod 7			modula
0	1	2	3	0	1	2	celkem 7 bitů

Tabulka 6.3.2: 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 vyjádření 100 hodnot na dvouprocesorovém systému optimální.

6.4 Reprezentace čísel s pohyblivou čárkou v FP RNS

Systém, který jsem navrhl a implementoval pracuje s pohyblivou desetinnou čárkou, tento systém jsem nazval RNS v plovoucí řádové čárce (dále FP RNS). Původní návrh vycházel z článku [30], 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 6.4.1: 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 6.4.1. 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 6.4.2. 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 6.4.3. 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 6.4.4. 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. Protože pro výpočty ODR (vyskytujících se v této práci) nebyla tato operace použita, je součástí práce pouze obecné shrnutí dělení v RNS v předešlé kapitole.

Shrnutí

Na začátku kapitoly jsou uvedeny dnešní způsoby uložení celých a desetinných čísel v počítačích. Následně je popsán systém 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 7

Využití aritmetiky zbytkových tříd pro výpočet ODR

Výsledky řešení ODR získané pomocí metody Taylorovy řady lze nalézt v kapitole věnované semi - analytickým výpočtům. V této kapitole navazuji na výše uvedenou kapitolu a uvádím svůj výzkum výpočtu ODR v RNS.

7.1 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 7.1.1. 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 7.1.2. 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 27 Výpočet velikosti kroku h

Require: všechna čísla x ve výrazech x/h zvolené numerické metody, jejíž počet je n, krok h je velikosti 10^{-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 7.1.3. Zadání rovnice je ve tvaru:

$$y' = y, y(0) = 1, \tag{7.1.1}$$

a MTR je použita pro interval $t \in <0, 0.945>$(číslo 0.945 je násobek kroku h vypočteného pomocí algoritmu 27). 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 27 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 7.1.1. 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 7.1.1: 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 7.1.1 je patrné, že typ *double* postrádá potřebnou přesnost pro tento typ výpočtu (dle tabulky 6.2.3 je přesnost výpočtu v *double* 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 A.2.

Při použití MTR 20. řádu je situace více názornější. Následující tabulka 7.1.2 zobrazuje výsledek je pro uzel $t = 1.01846745$ a stejnou ODR (7.1.1):

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 7.1.2: 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 7.1.2 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 7.1.4. 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 7.1.3: 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 7.1.4: 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 7.1.5: 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 7.1.6: 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 7.1.5. 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 7.1.6. Rovnice 7.1.4 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 7.1.7: 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 (tabulka 6.2.3), 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 7.1.8: 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 27. 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 7.1.7. Bude se řešit ODR z předešlého příkladu 7.1.4. 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 7.1.9: 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	1.26631690800854350251368017688817361593246459960937500	51
1.20	1.90544611181398234007430976577029005329623818397521972656250	58
1.35	2.79440449095570638697005610845295389981626090966165065765380859375	65
1.50	4.02059165956042332012169311229386415974323208564540836960077285766601562500	72

Tabulka 7.1.10: 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.

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

Tyto rovnice byly řešeny v kapitole 3.1. Rovnice lze pomocí Taylorova rozvoje v FP RNS řešit velice snadno, což lze ukázat na následujícím příkladě:

Příklad 7.2.1. 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.

7.3 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 28 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 ← |△(y2(t)) − △(y1(t))|
v ← |△(y1(t)) − △(y0(t))|
if v < w then
    t ← w
else
    t ← v
end if
X ← t × 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 7.1.8, 7.1.10). 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.

7.4 Stanovení libovolné velikosti kroku h

Systém FP RNS umožňuje získat výsledek ODR bez zaokrouhlovací chyb. Tohoto stavu je dosaženo použitím kroku h, jehož velikost je stanovena algoritmem 27.

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

Takto kapitola pojednávala 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 8

Možnosti rozšíření implementované aritmetiky zbytkových tříd

Implementovaný systém RNS lze dále rozšířit. Jednou z možností zvýšení přesnosti je použití vysokých prvočísel, které mohou být dynamicky počítány během výpočtu. Pro urychlení lze rozložit počítaná čísla na více modul (čímž se sníží bitová zátěž na každém procesoru), nebo použít jako báze modulů prvočísla speciálního tvaru. Jak již bylo napsáno, modifikace systému na zpřesnění a urychlení výpočtu se neovlivňují a lze je implementovat zároveň.

8.1 Použití vysokých prvočísel

Vysoká prvočísla lze získat dvěma způsoby, prvním z nich je použití databáze dosud známých prvočísel, (mezi nejznámější databáze patří [79], [80], [78]), druhým způsobem je test prvočíselnosti čísel v daném rozsahu.

Vzhledem k jednoduchosti nalezení požadovaných čísel v databázi, bude tato kapitola věnována především hledání prvočísel pomocí algoritmů.

Algoritmus AKS

Důležitým algoritmem testujícím prvočíselnost je AKS algoritmus, který byl prezentován v roce 2002 [37]. Deterministicky je schopen říci, zda-li je dané přirozené číslo prvočíslem v polynomiálním čase. Tento algoritmus se odlišuje od jiných testů prvočíselnosti následujícími vlastnostmi:

- je schopen rozpoznat jakékoliv prvočíslo, nejen prvočísla určitého druhu (např. Lucas-Lehmer [59] test pro Mersennovy čísla nebo Pépinův test [77] pro Fermatovy čísla),
- určení prvočíselnosti pro všechna čísla je provedeno v polynomiálním čase (např. algoritmus ARPCL [10] a ECPP [58] jsou polynomiální jen pro určitá prvočísla),
- algoritmus je deterministický, pravděpodobnostní algoritmy (např. Miller-Rabin [81], Baillie [71]) jsou schopny s určitou mírou pravděpodobností určit, jestli je dané číslo prvočíslem a se zvyšujícím se počtem běhů algoritmu, se pravděpodobnost správného určení prvočísla zvětšuje, avšak vždy je menší než 1,
- správnost algoritmu není založena na nedokázaných hypotézách teorie čísel (např. v kontrastu s Millerovým testem [75] založeným na Riemannově hypotéze [82]).

Poznámka 8.1.1. Dokázáním Riemannovy hypotézy by se vyřešilo mnoho problémů současné matematiky. Lze ji popsat jako domněnku o rozložení kořenů Riemannovy zeta - funkce:

$$\zeta(s) = 1 + \frac{1}{2^s} + \frac{1}{3^s} + \frac{1}{4^s} + ... = \sum_{n=1}^{\infty} \frac{1}{n^s},$$

pro $s \in \mathbb{C}, s > 1$. Tyto kořeny se dělí na triviální nulové body (sudá záporná celá čísla) a netriviální nulové body. Následující tvrzení nazýváme Riemannovou hypotézou:

Tvrzení. *Všechny netriviální nulové body Riemannovy zeta - funkce mají reálnou část rovnu* $1/2$.

Algoritmus AKS se skládá z několika testů, které při nalezení složeného čísla ukončí běh. Formální zápis AKS algoritmu je následující:

Algoritmus 29 AKS algoritmus

Require: $a,b,n \in \mathbb{N}, a,b,n > 1$
Ensure: n je prvočíslo (prime), nebo složené číslo (composite)
  if $n = a^b$ **then**
    return composite
  end if
  find smallest r such that $o_r(n) > log^2 n$
  if $1 < GCD(a,n) < n$ for some $a \leq r$ **then**
    return composite
  end if
  if $n \leq r$ **then**
    return prime
  end if
  for $a = 1$ to $\lfloor \sqrt{\varphi(r)} log(n) \rfloor$ **do**
    if $(X+a)^n \neq X^n + a (mod\, X^r - 1, n)$ **then**
      return composite
    end if
  end for
  return prime

V algoritmu se vyskytuje výraz $o_r(n)$ značící multiplikativní řád $n\ mod\ r$.

Definice 8.1.2. Mějme celé číslo n a přirozené číslo r, pro něž platí $\mathrm{GCD}(n,r) = 1$, multiplikativní řád $n\ mod\ r$ (značené $o_r(n)$) je nejmenší přirozené číslo k takové, že:

$$n^k \equiv 1\, mod\, r,$$

Nyní je uveden příklad výpočtu dle definice.

Příklad 8.1.3. Určíme multiplikativní řád $o_5(3)$ pro číslo $n = 3$ a $r = 5$, tedy:

$$\begin{aligned} 3^1 mod\, 5 &\equiv 3\, mod\, 5, \\ 3^2 mod\, 5 &\equiv 4\, mod\, 5, \\ 3^3 mod\, 5 &\equiv 2\, mod\, 5, \\ 3^4 mod\, 5 &\equiv 1\, mod\, 5. \end{aligned}$$

Multiplikativní řád 3 mod 5 je $o_5(3) = 4$.

Symbol $\varphi(n)$ značí Eulerovu funkci čísla n.

Definice 8.1.4. Eulerova funkce $\varphi(n)$ značí počet všech přirozených čísel, která jsou menší než přirozené číslo n a jsou s ním vzájemně nesoudělná. Její výpočet je založena na větě:

Věta 8.1.5. *Každé přirozené číslo n lze zapsat ve tvaru:*

$$n = p_1^{e_1} \times p_2^{e_2} \times ... \times p_n^{e_n},$$

kde p_i značí prvočísla a e_i mocniny mocniny prvočísla, pak Eulerova funkce $\varphi(n)$ lze vypočíst následovně:

$$\varphi(n) = n \prod_{p_i|n} (1 - \frac{1}{p_i}).$$

Důkaz. Lze nalézt v [74] □

Příklad 8.1.6. Vypočítáme Eulerovu funkci pro číslo 12, tedy $\varphi(12)$:

$$\varphi(12) = \varphi(2^2 3) = 12(1 - \frac{1}{2})(1 - \frac{1}{3}) = 12\frac{1}{2}\frac{2}{3} = 4.$$

Správnost ověříme následovně:

- pro čísla $2, 3, 4, 6, 8, 9, 10$ je GCD větší než 1
- pro čísla $1, 5, 7, 11$ je GCD roven 1

Tedy výpočet Eulerovy funkce je správný $\varphi(12) = 4$.

Důsledek. *Pokud je číslo $n \in \mathbb{N}$ prvočíslem, funkce udává počet multiplikativních inverzí modulo n.*

Časová složitost algoritmu AKS je $O(log^{12} n)$ s úpravami lze čas snížit až na $O(log^6 n)$.

Rabin - Millerův test

Tento test patří k nejpoužívanějším, přestože je schopen testované číslo prohlásit za prvočíslo (nebo číslo složené) pouze s určitou pravděpodobností. S každým dalším spuštěním testu se zvyšuje pravděpodobnost správného určení, zda-li je dané číslo číslem složeným nebo prvočíslem .

Tento test má časovou složitost $O(\kappa log^3 n)$ (za použití modulárního umocňování), kde κ je počet různých čísel vybraných v cyklu. Při použití násobení založeného na FFT, klesne doba výpočtu na $O(\kappa log^3 log\, n\, log\, log\, log\, n)$ což je mnohem méně času než při použití algoritmu AKS.

Algoritmus lze formálně zapsat následovně:

Algoritmus 30 Rabin - Miller test

Require: $n \in \mathbb{N}, n > 3, n \bmod 2 \neq 0$
Ensure: n je složené číslo (composite), nebo pravděpodobně prvočíslo (probably prime)

```
d = n − 1
while d mod 2 = 0 do
    d = d/2
    s = s + 1
end while
loop
    random a in the range < 2, n − 2 >
    x = a^s mod n
    if x = 1 or x = n − 1 then
        do next loop
    end if
    for r = 0 to s − 1 do
        x = x^2 mod n
        if x = 1 then
            return composite
        end if
        if x = n − 1 then
            do next loop
        end if
    end for
    return composite
end loop
return probably prime
```

V algoritmu lze nalézt smyčku κ, která určuje, kolikrát má daný test proběhnout a tedy s jakou pravděpodobností bude nalezené číslo prvočíslem. Na závěr lze říci, že tímto algoritmem lze testovat stejné prvočíslo v několika paralelních větvích a tak hledání prvočísla značně urychlit a zvýšit pravděpodobnost jeho správného určení.

8.2 Použití OEF modul

Nejpoužívanější operací v systémech *RNS* je bezesporu operace modulo. Proto se jeví jako výhodné použití OEF modul, které umožňují redukci čísel pouze použitím bitových posunů [4] .

Definice 8.2.1. Nechť p je prvočíslo a m je přirozené číslo, pak optimálním rozšířeným tělesem (OEF) budeme nazývat konečné těleso $GF(p^m)$ s následujícími vlastnostmi:

1. $p = 2^n - c$, pro celá čísla n a c, splňuje podmínku $log_2|c| \leq \lfloor n/2 \rfloor$,
2. existuje ireducibilní polynom $P(x) = x^m - \omega$ v $GF(p)$, kde ω je přirozené číslo.

Definice 8.2.2. Pokud $c \in \{\pm 1\}$ potom OEF bude typu 1 (Mersennovy prvočísla), pokud $\omega = 2$ bude OEF typu 2.

Pro určení, zda-li je daný polynom ireducibilní se používá následující věta:

Věta 8.2.3. *Nechť $m \in \mathbb{Z}, m \geq 2$ a $\omega \in GF(p)$, potom dvojčlen $P(x) = x^m - \omega$ je ireducibilní v $GF(p)$, pouze a jen tehdy pokud následující dvě podmínky jsou splněny:*

1. každý prvočíselný faktor čísla m dělí řád e čísla ω v $GF(p)$ ale nikoliv $(p-1)/e$,

2. $p \equiv 1 \, mod \, 4$, pokud $m \equiv 0 \, mod \, 4$.

Důkaz. Lze nalézt např. v [31] a [4]. □

Příklad 8.2.4. Příkladem OEF může být například těleso $GF(p^6)$, kde $p = 2^{32} - 387$, a ireducibilní polynom je tvaru $p(x) = x^6 - 387$.

Protože polynomy v $GF(p^m)$ lze chápat jako zápis binárního čísla, byly v článku [4] zveřejněny algoritmy, které umožňují základní aritmetické operace s těmito polynomy velmi jednoduchým způsobem.

Sčítání a odčítání

Sčítání a odčítání patří mezi přímočaré operace, po sečtení (odečtení) koeficientů polynomu stejného řádu se v případě výsledku většího než p (vyjadřující modulo), provede jeho odečtení. Postup při sčítání polynomů $A(x)$ a $B(x)$ je zapsán následovně:

Algoritmus 31 Sčítání OEF

Require: $A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0, B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + ...b_1x + b_0, \; A,B \in GF(p^m)$
Ensure: $A(x) + B(x) \equiv C(x) \in GF(p^m)$
  for $i = 0$ to $m-1$ **do**
    $c_i = a_i + b_i$
    if $c_i \geq p$ **then**
      $c_i = c_i - p$
    end if
  end for

Obdobně se postupuje i při operaci odečítání, pouze je místo operace $c_i = a_i + b_i$ použita operace $c_i = a_i - b_i$.

Násobení

Operace násobení se skládá ze dvou částí, první z nich je samotné násobení, druhou redukce. Nejprve je uvedeno násobení:

Algoritmus 32 Násobení OEF

Require: $A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0, B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + ...b_1x + b_0, A,B \in GF(p^m)$
Ensure: $C'(x) = A(x) \times B(x)$
  for $i = 0$ to $2m-2$ **do**
    $c'_i = 0$
    if $i < m$ **then**
      $s = i, e = 0$
    else
      $s = m-1, e = i-m+1$
    end if
    for $j = s$ down to e **do**
      $c'_i = c'_i + a_{i-j}b_j$
    end for
  end for

Algoritmus se skládá z cyklu, který prochází jednotlivými členy a provádí jednoduché aritmetické operace. Násobení lze dále urychlit způsoby popsanými v kapitole 5.8 (protože školní násobení se skládá z m^2 operací násobení a $(m-1)^2$ sčítání). Produktem násobení polynomů $A(x)$ a $B(x)$ se označí proměnná $C'(x) = A(x) \times B(x)$, samotná redukce bude rovna proměnné $C(x) = C'(x) \, mod \, M$, $C(x) \in GF(p^m)$ a nejjednodušší algoritmus pro vznik mezivýsledku $C'(x)$ je popsán v následující kapitole.

Redukce v $GF(p^m)$

Počet operací potřebných pro redukci po násobení a získání výsledku $C(x)$, dokumentuje následující věta:

Věta 8.2.5. *Pro daný polynom $C'(x)$ nad $GF(p^m)$ stupně menšího nebo rovno $2m-2$ lze polynom$C'(x)$ redukovat modulem $M(x) = x^m - \omega$, složeným z $m-1$ násobení číslem ω a $m-1$ sčítání v $GF(p^m)$.*

Důkaz. Lze nalézt např. v [4]. □

Protože moderní mikroprocesory jsou optimalizovány pro celočíselné operace, je vhodné využít šířku jejich používaného slova (dnes 32 a 64 bitů) také pro násobení. OEF využívá této skutečnosti konstrukcí podmnožin polynomů, které lze celočíselně reprezentovat v jednotlivých bitových slovech. Při této reprezentaci lze využít následující algoritmus pro *subfield* redukci (polynom bude uložen v poli $x[\,]$, a jednotlivé členy polynomu budou přístupny skrze index i).

Algoritmus 33 Redukce $M = 2^n - c$ v OEF

Require: $x \in \mathbb{N}, x < p^2$,ireducibilní polynom $M = 2^n - c, log_2 c \leq \frac{1}{2}n$
Ensure: $r \equiv x(mod \, M)$
  $r = x[0], q = x[1]$
  while $q > 0$ **do**
    $q = q \times c$
    $r = r + q[0]$
    $q = q[1]$
  end while
  while $r \geq M$ **do**
    $r = r - M$
  end while

Algoritmus se skládá ze dvou cyklů, kdy druhý je spuštěn pouze pokud je výsledek větší než M. V případě použití OEF lze redukce provádět bez použití dělení pouze pomocí bitových posunů jak naznačuje uvedený algoritmus.

Pokud se provede násobení dvou polynomů délky bitového slova, získá se dvojnásobná bitová délka mezivýsledku $C'(x)$:

$$C'(x) = c'_{2m-2}x^{2m-2} + c'_{2m-3}x^{2m-3} + \ldots + c'_1x^1 + c'_0.$$

Při použití ireducibilního polynomu je nutné redukovat polynomem $M(x) = x^m - \omega$ pouze členy $c'_{m+i}x^{m+i}$, kde $i \geq 0$. Při využití kongruencí lze získat výpočet $C(x) = A(x) \times B(x) \, mod \, M(x)$ následujícím postupem:

$$
\begin{aligned}
C(x) &= \left(\sum_{i=0}^{m-1} a_i x^i \cdot \sum_{i=0}^{m-1} b_j x^j\right) mod\, M(x), \\
&= \sum_{i,j\geq 0, i+j=k<m} a_i b_j x^k + w \sum_{i,j\geq 0, i+j=k\geq m} a_i b_j x^{k-m}, \\
&= \sum_{k=0}^{m-1} \left(\sum_{i=0}^{k} a_i b_{k-i} + w \sum_{i=k+1}^{m-1} a_i b_{m+k-i}\right) x^k.
\end{aligned}
$$

Tedy se celý výpočet násobení v $GF(p^m)$ zjednoduší.

8.3 Možnost rozšíření FPGA implementace metody Taylorovy řady

Aritmetika FP RNS je implementována i s ohledem na výzkum vycházející z [33], jehož cílem byla implementace MTR v FPGA procesoru. V práci byly popsány a navrhnuty tři integrátory: paralelně - paralelní integrátor, sériově - paralelní integrátor a sériově - sériový integrátor. Tyto integrátory používají celočíselnou aritmetiku, kterou lze modifikovat na aritmetiku FP RNS.

Shrnutí

Tato kapitola se věnovala možnostem urychlení a zpřesnění výpočtů ve FP RNS. Urychlení je možné provést pomocí použití speciálních provočísel OEF, která budou základem systému 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 lze prvočísla ve stanoveném intervalu ověřit (např. algoritmus AKS).

Kapitola 9

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.

9.1 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.

9.2 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.

9.3 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).

Literatura

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

[2] Alia, G.; Martinelli, E.: Short note: VLSI Binary-Residue Converters for Pipelined Processing. 1990, s. 473–475.

[3] Amdahl, G.: Validity of the single processor approach to achieving large scale computing capabilities. 1967.

[4] Bailey, D. V.; Paar., C.: Optimal extension fields for fast arithmetic in public-key algorithms.

[5] 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.

[6] Barto, L.; Tůma, J.: Konečná tělesa. [online], [cit. 2013-06-11].
URL <http://www.karlin.mff.cuni.cz/~barto/student/SkriptaKonTel.pdf>

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

[8] Bortlíček, Z.: *Přesnost a stabilita numerických algoritmů*. Přírodovědecká fakulta MU, 2006.

[9] Bortlíček, Z.: *Přesnost a stabilita numerických algoritmů*. Bakalářská práce, Přírodovědecká fakulta MU v Brně, 2006.

[10] Cohen, H.; Lenstra, H.: *Primality testing and Jacobi sums*. Report. Department of Mathematics. University of Amsterdam, University of Amsterdam, Department, Univ., 1982.

[11] Dalík, J.: *Numerické metody*. Akademické nakladatelství CERM, s.r.o. Brno, 1997, ISBN 80-214-0646-1.

[12] Dana Černá: Základy numerické matematiky. [online], [cit. 2013-8-25].
URL <http://kmd.fp.tul.cz/old/lide/cerna/ZNM/znm_prednaska_1.pdf>

[13] Dimauro, G.; Impedovo, S.; Pirlo, G.: A New Technique for Fast Number Comparison in the Residue Number System.

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

[15] Čermák, L.: *Numerické metody II*. Akademické nakladatelství CERM, s.r.o. Brno, 2004, ISBN 80-214-2722-1.

[16] Čermák, L.; Hlavička, R.: *Numerické metody*. Akademické nakladatelství CERM, s.r.o. Brno, 2005, ISBN 80-214-3071-0.

[17] Euklides: Elements. [online], [cit. 2008-12-13].
URL <http://aleph0.clarku.edu/~djoyce/java/elements/toc.html>

[18] Řezáč, D.: *Stiff Systems of Differential Equations and Modern Taylor Series Method*. Dizertační práce, FIT VUT v Brně, 2004.

[19] G.A. Orton, L. P.; Tavares, S. E.: New Fault tolerant techniques for residue number systems. IEEE Computer Society, 1992, s. 1453–1464.

[20] Gamberger, D.: New approach to integer division in residue number systems. 1991, s. 84–91.

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

[22] Gauss, C. F.: Disquisitiones Arithmeticae. [online], [cit. 2008-12-13].
URL <http://gdz.sub.uni-goettingen.de/dms/load/img/?PPN=PPN235993352&IDDOC=137206>

[23] Gilpin, A.: Complexity Classes. [online], [cit. 2013-06-10].
URL <http://www.cs.cmu.edu/afs/cs/academic/class/15859-f04/www/scribes/lec2.pdf>

[24] 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.

[25] 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.

[26] Hitz, M.; Kaltofen, E.: Integer division in residue number systems. 1995, s. 983–989.

[27] Horová, I.; Zelenka, J.: Numerika. [online], [cit. 20012-12-13].
URL <http://www.math.muni.cz/~zelinka/dokumenty/numerika.pdf>

[28] Hung, C. Y.: Error analysis of approximate Chinese-remainder-theorem decoding. IEEE Transactions on Computers, 1995, iSSN 0018-9340.

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

[30] Jen-Shiun Chiang, M. L.: Floating-point Numbers in Residue Number Systems. *Computers Mathematics Application*, ročník 22, 1991.

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

[32] Kopřiva, J.: *Srovnání algoritmů při řešení problému obchodního cestujícího*. Diplomová práce, Fakulta podnikatelská VUT v Brně, 2009.

[33] Kraus, M.: *Parallel Computer Systems based on numerical integrations*. FIT VUT Brno, Disertation work, 2013.

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

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

[36] Limpouch, J.: Úvod do numerické matematiky. [online], [cit. 20012-12-13].
URL <http://fluorescence.fjfi.cvut.cz/~limpouch/numet/foluvoda.pdf>

[37] M. Agrawal, N. K.; Saxena, N.: PRIMES is in P. *Ann. of Math*, ročník 2, 2002.

[38] Maplesoft: Maple 16. [online], [cit. 12-12-2012].
URL <http://www.maplesoft.com/products/maple/>

[39] Mikulášek, K.: *Polynomial Transformations of Systems of Differential Equations and Their Applications*. Dizertační práce, FEI VUT v Brně, 2000.

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

[41] Nannarelli, A.; Cardarilli, G. C.: Reducing Power Dissipation in FIR Filters using the Residue Number System. Institute of Electrical and Electronics Engineers, 2001, s. 305–308.

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

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

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

[45] R. L. Rivest, A. S.; Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. ACM, 1978, s. 120–126.

[46] Rábová, Z.; Hanáček, P.; Peringer, P.; aj.: Užitečné rady pro psaní odborného textu. [online], [cit. 2008-11-28].
URL <http://www.fit.vutbr.cz/info/statnice/psani.textu.html.cs>

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

[48] Rybička, J.: *pro začátečníky*. Konvoj, 1999, ISBN 80-85615-77-0.

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

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

[51] Svoboda, A.; Valach, M.: Rational Residual System of Residual Classes. *Stroje na zpracovani informaci CSAV*, ročník 5, 1957.

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

[53] Tišnovský, P.: Norma IEEE 754. [online], [cit. 2012-06-25].
URL <http://www.root.cz/clanky/norma-ieee-754-a-pribuzni-formaty-plovouci-radove-tecky/#k02>

[54] Vaníček, J.: *Teoretické základy informatiky*. Addison-Wesley UK, 2007, ISBN 80-85867-35-4.

[55] Václav Ibl: Taylorovy řady elementárních funkcí. [online], [cit. 2013-8-25].
URL <http://www.karlin.mff.cuni.cz/katedry/kdm/diplomky/vaclav_ibl/taylorovy_rady.pdf>

[56] Vu, T. V.: Effecient implementation of Chinese remainder theorem for sign detection and residue decoding. IEEE Computer Society, 1985, s. 646–651.

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

[58] Weisstein, E. W.: Elliptic Curve Primality Proving. [online], [cit. 2013-8-25].
URL <http://mathworld.wolfram.com/EllipticCurvePrimalityProving.html>

[59] Weisstein, E. W.: Lucas-Lehmer Test. [online], [cit. 2013-8-25].
URL <http://mathworld.wolfram.com/Lucas-LehmerTest.html>

[60] WWW stránky: IEEE-754 Reference Material. [online], [cit. 20012-12-13].
URL <http://babbage.cs.qc.cuny.edu/IEEE-754.old/References.xhtml>

[61] WWW stránky: Reevaluating Amdahl's Law. [online], [cit. 2008-11-28].
URL <http://www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.html>

[62] WWW stránky: Reevaluating Amdahl's Law and Gustafson's Law. [online], [cit. 2008-11-28].
URL <http://spartan.cis.temple.edu/shi/public_html/docs/amdahl/amdahl.html>

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

[64] WWW stránky: Chinese Remainder Theorem. [online], [cit. 2008-12-13].
URL <http://www.proofwiki.org/wiki/Chinese_Remainder_Theorem>

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

[66] WWW stránky: Složitost. [online], [cit. 2012-06-10].
URL <http://ksp.mff.cuni.cz/study/cooks/cookbook-chapters/01-slozitost.pdf>

[67] WWW stránky: Třídy vyčíslitelnosti a Turingovy stroje. [online], [cit. 2012-06-10].
URL <http://www.algoritmy.net/article/5774/Tridy-slozitosti>

[68] WWW stránky: Turingovy stroje. [online], [cit. 2012-06-10].
URL <http://www.fit.vutbr.cz/study/courses/TIN/public/Prednasky/tin-pr07-ts1.pdf>

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

[70] WWW stránky: Adams Methods. [online], [cit. 2013-8-25].
URL <http://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node6.html>

[71] WWW stránky: Baillie-PSW Primality Test. [online], [cit. 2013-8-25].
URL <http://mathworld.wolfram.com/Baillie-PSWPrimalityTest.html>

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

[73] WWW stránky: Common Lisp. [online], [cit. 2013-8-25].
URL <http://common-lisp.net/>

[74] 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>

[75] WWW stránky: Miller's Primality Test. [online], [cit. 2013-8-25].
URL <http://mathworld.wolfram.com/MillersPrimalityTest.html>

[76] WWW stránky: Numerická stabilita. [online], [cit. 2013-8-25].
URL <http://kfe.fjfi.cvut.cz/~limpouch/numet/foluvux/node9.html>

[77] WWW stránky: Pepin's test. [online], [cit. 2013-8-25].
URL <http://primes.utm.edu/glossary/page.php?sort=PepinsTest>

[78] WWW stránky: Prime database project. [online], [cit. 2013-8-25].
URL <http://sourceforge.net/projects/primedb/?source=navbar>

[79] WWW stránky: The Prime Database: The List of Largest Known Primes. [online], [cit. 2013-8-25].
URL <http://primes.utm.edu/primes/>

[80] WWW stránky: Prime Numbers. [online], [cit. 2013-8-25].
URL <http://www.bigprimes.net/>

[81] WWW stránky: Rabin-Miller Strong Pseudoprime Test. [online], [cit. 2013-8-25].
URL <http://mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html>

[82] WWW stránky: Riemann Hypothesis. [online], [cit. 2013-8-25].
URL <http://mathworld.wolfram.com/RiemannHypothesis.html>

[83] WWW stránky: Stiffness Detection. [online], [cit. 2013-8-25].
URL <http://reference.wolfram.com/mathematica/tutorial/NDSolveStiffnessTest.html>

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

[85] WWW stránky: Symbolic Differentiation. [online], [cit. 2013-8-25].
URL <http://mitpress.mit.edu/sicp/full-text/sicp/book/node39.html>

[86] WWW stránky: What is the time complexity of Euclid's Algorithm. [online], [cit. 2013-8-25].
URL <http://math.stackexchange.com/questions/258596/what-is-the-time-complexity-of-euclids-algorithm-upper-bound-lower-bound-and-a>

[87] Yoshino, M.; Okeya, K.; Vuillaume, C.: Unbridle the bit-length of a crypto-coprocessor with montgomery multiplication. In *Proceedings of the 13th international conference on Selected areas in cryptography*, Springer-Verlag, 2007, s. 188–202.

Přehled publikací autora

[I] Kopřiva Jan, Kunovský Jiří, Šátek Václav, Kraus Michal. Semi-analytical Computations Based on TKSL. In: Second UKSIM European Symposium on Computer Modeling and Simulation. Liverpool: IEEE Computer Society, 2008, s. 159-164. ISBN 978-0-7695-3325-4.

[II] Kaluža Vlastimil, Kopřiva Jan, Kunovský Jiří a Sehnalová Pavla. Using Differential Equations in Electrical Circuits Simulation. In: Proceedings of 42nd Spring International Conference MOSIS '08. Ostrava: MARQ, 2008, s. 150-154. ISBN 978-80-86840-40-6.

[III] Kaluža Vlastimil, Kopřiva Jan a Kunovský Jiří. Partial Differential Equations in TKSL. In: Proceedings of CSE 2008. Košice: elfa, s.r.o., TU v Košiciach, 2008, s. 312-319. ISBN 978-80-8086-092-9.

[IV] Sehnalová Pavla, Kunovský Jiří, Kaluža Vlastimil a Kopřiva Jan. Using Differential Equations in Electrical Circuits Simulation. International Journal of Autonomic Computing. London: Inderscience Publishers, 2009, roč. 1, č. 2, s. 192-201. ISSN 1741-8569.

[V] Kunovský Jiří, Šátek Václav, Kraus Michal a Kopřiva Jan. Automatic Method Order Settings. In: Proceedings of Eleventh International Conference on Computer Modelling and Simulation. Cambridge: IEEE Computer Society, 2009, s. 117-122. ISBN 978-0-7695-3593-7.

[VI] Kaluža Vlastimil, Kunovský Jiří, Sehnalová Pavla a Kopřiva Jan. Technical Initial Problems and Automatic Transformation. In: 2009 International Conference on Computational Intelligence, Modelling and Simulation. Brno: IEEE Computer Society, 2009, s. 75-80. ISBN 978-0-7695-3795-5.

[VII] Šátek Martina, Kopřiva Jan, Kunovský Jiří a Pindryč Milan. Methodology of the Taylor Series Based Computations. In: Proceedings of Third Asia International Conference on Modelling and Simulation. Bandung/Bali: IEEE Computer Society, 2009, s. 206-211. ISBN 978-0-7695-3648-4.

[VIII] Kopřiva Jan a Kraus Michal. Application of the Modern Taylor Series Method to a Multi-Torsion Chain. In: Proceedings of the 7th EUROSIM Congress on Modelling and Simulation. Praha: Vydavatelství ČVUT, 2010, s. 100-106. ISBN 978-80-01-04589-3.

[IX] Šátek Václav, Kunovský Jiří a Kopřiva Jan. Advanced Stiff Systems Detection. In: Proceedings of the Eleventh International Scientific Conference on Informatics. Rožňava: Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2011, s. 208-212. ISBN 978-80-89284-94-8.

[X] Šátek Václav, Kunovský Jiří a Kopřiva Jan. Advanced Stiff Systems Detection. Acta Electrotechnica et Informatica. 2012, roč. 11, č. 4, s. 66-71. ISSN 1335-8243.

[XI] Kopřiva Jan, Kunovský Jiří, Šátek Václav a Kocina Filip. Numerical integration in the RNS. In: The Proceedings of the 12th Conference Informatics'2013. Košice: Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2013, s. 318-322. ISBN 978-80-8143-127-2.

[XII] Kopřiva Jan, Kunovský Jiří, Šátek Václav a Šátek Martina. Parallel system based on the RNS. In: The Proceedings of the 12th Conference Informatics'2013. Košice: Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2013, s. 323-328. ISBN 978-80-8143-127-2.

[XIII] Kopřiva Jan, Kunovský Jiří, Šátek Václav a Šátek Martina. Parallel Computations Based on Automatic Transformation of Ordinary Differential Equations. In: Proceedings of the 11th International Conference of Numerical Analysis and Applied Mathematics. Rhodes: American Institute of Physics, 2013, s. 2293-2296. ISBN 978-0-7354-1184-5. ISSN 0094243X.

Příloha A

Řešené příklady

A.1 Semi - analytické výpočty

ODR s řešením ve tvaru polynomiální funkce

Příklad A.1.1. je dána rovnice:

$$y' = 2t, y(0) = 0,$$

analytické řešení je ve tvaru: $y = t^2$. V tomto případě se použil řád $ORD = 3$ MTR a výpočet vypadá následovně:

$$\begin{aligned} y_1 &= y_0 + DY1 + DY2 + DY3, \\ DY1 &= h \cdot y'(0) = h \cdot 0 = 0, \\ DY2 &= \frac{h^2}{2!} \cdot y''(0) = \frac{h^2}{2!} \cdot 1 = 1, \\ DY3 &= \frac{h^3}{3!} \cdot y'''(0) = \frac{h^3}{3!} \cdot 0 = 0. \end{aligned}$$

Přesný výpočet lze získat i pro vyšší řád polynomu. Tato skutečnost je ukázána na dalším příkladě:

Příklad A.1.2. Je dána rovnice $y' = 31t^{30}$, $y(0) = 0$. Analytické řešení je $y = t^{31}$. Řešení pro $ORD = 4, DT = 0.1, T = 1$ je v tabulce A.1.1.

T	Y
0	0
0.1	0
0.2	2.06367E-0026
0.3	1.7115243554335E-0018
0.4	8.12974437520533E-0014
0.5	1.82651371657836E-0010
0.6	7.72381950327754E-0008
0.7	1.1355027397379E-0005
0.8	8.03984113462566E-0004
0.9	0.0332747342645103
1	0.911702106353534

Tabulka A.1.1: Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=0.1, $ORD = 4$, MTR

Z obrázku je možné vyčíst, že hodnota $y(1) \neq 1$, řád metody $ORD = 4$ odpovídá metodě Runge-Kutta 4. řádu.

T	Y
0	0
0.1	0
0.2	1.60645503E-0023
0.3	4.184693362691 83E-0017
0.4	4.31437552669 98E-0013
0.5	4.59709392977515E-0010
0.6	1.322646287788 88E-0007
0.7	1.576591429484 05E-0005
0.8	9.901386285629 43E-0004
0.9	0.0381493543715231
1	0.999974648093839

Tabulka A.1.2: Řešení: Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=0.1, $ORD = 20$, MTR

Při změně řádu ORD=20 bylo získáno přesnější řešení $\delta(ORD = 20) < \delta(ORD = 4)$, avšak stále platí, že $y(1) \neq 1$. Analyticky přesná hodnota $y(1) = 1$ se získá při volbě řádu $ORD = 32$ a kroku $h = 0.1$, jak ukazuje tabulka A.1.3

T	Y
0	0
0.1	1E-0031
0.2	2.147483648E-0022
0.3	6.176733962839 47E-0017
0.4	4.611686018427 39E-0013
0.5	4.656612873077 39E-0010
0.6	1.326443518324 4E-0007
0.7	1.577753820348 46E-0005
0.8	9.903520314283 04E-0004
0.9	0.0381520424476946
1	1

Tabulka A.1.3: Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=0.1, $ORD = 32$, MTR

Analyticky přesného výpočtu se dosáhne také při $ORD = 32$ a kroku $h = 1$, jak ukazuje tabulka A.1.4

T	_ORDER	Y
0	0	0
1	32	1

Tabulka A.1.4: Řešení: $y' = 31t^{30}$, $y(0) = 0$, h=1, $ORD = 32$, MTR

A.2 Využití aritmetiky zbytkových tříd pro výpočet ODR

Příklad A.2.1. Výsledky řešení rovnice : $y' = y$, $y(0) = 1$: metodou Taylorovy řady 10. řádu:

t	program	$y(t)$	$\delta(y(\tilde{t}))$
0.063	MAPLE (an.)	1.06502683923130546430...	0
	RNS (nu.)	1.06502683923130546430...	0
	MATLAB (nu.)	1.06502683923130546430...	0
	C++ (double) (nu.)	1.06502683923130540000...	6.037407E-17
0.126	MAPLE (an.)	1.13428216828302497603...	0
	RNS (nu.)	1.13428216828302497603...	0
	MATLAB (nu.)	1.13428216828302497603...	0
	C++ (double) (nu.)	1.13428216828302510000...	1.0929379E-16
0.189	MAPLE (an.)	1.20804095248290181116...	0
	RNS (nu.)	1.20804095248290181116...	0
	MATLAB (nu.)	1.20804095248290181116...	0
	C++ (double) (nu.)	1.20804095248290140000...	3.403527E-16
0.252	MAPLE (an.)	1.286596037284840590...	0
	RNS (nu.)	1.286596037284840590...	0
	MATLAB (nu.)	1.286596037284840590...	0
	C++ (double) (nu.)	1.286596037284840000...	4.58574E-16
0.315	MAPLE (an.)	1.3702593109569966109...	0
	RNS (nu.)	1.3702593109569966109...	0
	MATLAB (nu.)	1.3702593109569966109...	0
	C++ (double) (nu.)	1.3702593109569963000...	2.268914E-16
0.378	MAPLE (an.)	1.459362942875796631...	0
	RNS (nu.)	1.459362942875796631...	0
	MATLAB (nu.)	1.459362942875796631...	0
	C++ (double) (nu.)	1.459362942875796200...	2.95334E-16
0.441	MAPLE (an.)	1.5542607023423058792...	0
	RNS (nu.)	1.5542607023423058792...	0
	MATLAB (nu.)	1.5542607023423058792	0
	C++ (double) (nu.)	1.5542607023423054000...	3.083138E-16
0.504	MAPLE (an.)	1.6553293631570549199...	0
	RNS (nu.)	1.6553293631570549199...	0
	MATLAB (nu.)	1.6553293631570549199...	0
	C++ (double) (nu.)	1.6553293631570543000...	3.744874E-16
0.567	MAPLE (an.)	1.7629701995299279888...	0
	RNS (nu.)	1.7629701995299279888...	0
	MATLAB (nu.)	1.7629701995299279888...	0
	C++ (double) (nu.)	1.7629701995299274000...	3.339818E-16

Tabulka A.2.1: Řešení: $y = y'$, $y(0) = 1$, $h = 0.063$, MTR 10. řádu (1. část)

t	program	$y(t)$	$\delta(y(\tilde{t}))$
0.630	MAPLE (anal.)	1.8776105792643431324...	0
	RNS (num.)	1.8776105792643431324...	0
	MATLAB (num.)	1.8776105792643431324...	0
	C++ (double) (num.)	1.8776105792643435000...	1.957807E-16
0.693	MAPLE (anal.)	1.9997056605411638985...	0
	RNS (num.)	1.9997056605411638985...	0
	MATLAB (num.)	1.9997056605411638985...	0
	C++ (double) (num.)	1.9997056605411632000...	3.493014E-16
0.756	MAPLE (anal.)	2.1297401990391056625...	0
	RNS (num.)	2.1297401990391056625...	0
	MATLAB (num.)	2.1297401990391056625...	0
	C++ (double) (num.)	2.1297401990391047000...	4.519331E-16
0.819	MAPLE (anal.)	2.2682304725664700867...	0
	RNS (num.)	2.2682304725664700867...	0
	MATLAB (num.)	2.2682304725664700867...	0
	C++ (double) (num.)	2.2682304725664690000...	4.790959E-16
0.882	MAPLE (anal.)	2.4157263308455979564...	0
	RNS (num.)	2.4157263308455979564...	0
	MATLAB (num.)	2.4157263308455979564...	0
	C++ (double) (num.)	2.4157263308455965000...	6.028829E-16
0.945	MAPLE (anal.)	2.572813378588326089...	0
	RNS (num.)	2.572813378588326089...	0
	MATLAB (num.)	2.572813378588326089...gv	0
	C++ (double) (num.)	2.572813378588324700...	5.39876E-16

Tabulka A.2.2: Řešení: $y' = y$, $y(0) = 1$, $h = 0.063$, MTR 10. řádu (2. část)

Příloha B

Klasifikace algoritmů

Algoritmy v disertační práci jsou srovnávány především na základě jejich asymptotické složitosti, proto je zde uveden krátký úvod do této problematiky.

Při provádění matematických výpočtů je často žádoucí, porovnat jak je daný algoritmus (postup výpočtu) rychlý nebo paměťově náročný vzhledem k jiným algoritmům řešícím stejný problém. Protože rychlost výpočtu počítačů a velikost paměti se neustále zvyšuje, bylo potřeba najít objektivní klasifikaci algoritmů, která by umožnila porovnání jejich rychlosti a paměťové náročnosti bez ohledu na technický pokrok výpočetních architektur. Používanou metrikou je počet kroků algoritmu vzhledem k velikosti vstupu n (časová složitost) a počet použitých paměťových buněk vzhledem k velikosti vstupu n (paměťová náročnost). Souhrnně se tyto metriky nazývají asymptotické složitosti algoritmů.

B.1 Asymptotická složitost algoritmů

Při porovnávání algoritmů se často záměrně zanedbávají aditivní a multiplikativní konstanty, neboť při velkém datovém vstupu ovlivňují efektivnost algoritmu pouze minimálně. Definici asymptotické složitosti algoritmu [54],[66] vypadá následovně:

Definice B.1.1. jsou dány funkce $f : \mathbb{N} \to \mathbb{R}_0^+$ a funkce $g : \mathbb{N} \to \mathbb{R}_0^+$. O funkcích se řekne, že

- funkce $f(n)$ je asymptoticky ohraničena funkcí $g(n)$ shora (značí se $f(n) \in O(g(n))$, pokud $\exists n_0 \in \mathbb{N}$ a $\exists C \in R^+$ tak, že $\forall n \in \mathbb{N} : n \geq n_0$ platí $f(n) \leq C \cdot g(n)$,
- funkce $f(n)$ je asymptoticky ohraničena funkcí $g(n)$ zdola (značí se $f(n) \in \Omega(g(n))$, pokud $\exists n_0 \in \mathbb{N}$ a $\exists C \in R^+$ tak, že $\forall n \in \mathbb{N} : n \geq n_0$ platí $C \cdot g(n) \leq f(n)$,
- funkce $f(n)$ je asymptoticky ohraničena funkcí $g(n)$ z obou stran (značí se $f(n) \in \Theta(g(n))$, pokud $\exists n_0 \in \mathbb{N}$ a $\exists C \in R^+$ tak, že $\forall n \in \mathbb{N} : n \geq n_0$ platí $C \cdot g(n) \leq f(n) \leq C \cdot g(n)$.

Uvedenou definici lze použít na časovou i prostorovou závislost na velikosti vstupu n. V praxi se nejčastěji používá složitost $O(g(n))$, vzhledem k odhadům nejhorších případů, které mohou být pro běh algoritmu kritické. Na závěr je třeba říci, že v případě určování složitosti se zachovává jen nejvýznamnější člen a odstraňují se konstanty, jak bylo dříve napsáno:

Příklad B.1.2. Je dán algoritmus A, jehož složitost pro počet kroků n bude: $n^2/4 + 6n + 12$. Pak se konstanty u jednotlivých členů odstraní, vznikne: $n^2 + n + 1$. Protože při $n \to \infty$ lze $n + 1$ vůči n^2 zanedbat, pak pro algoritmus platí $A \in O(n^2)$

Dle výše zmíněné definice, lze rozdělit algoritmy dle výpočetních složitosti.

B.2 Klasifikace algoritmů podle výpočetní složitosti

Preferovaným algoritmem vzhledem k rychlosti i paměťové náročnosti je algoritmus, jenž nezávisí na velikosti vstupu n a označuje se konstantní složitostí $O(1)$. Dále se dělí složitosti algoritmů na polynomiální a nepolynomiální.

Polynomiální složitosti lze rozřadit následovně:

$O(log(n))$−logaritmické, neboť $O(log(n)) \subset O(n)$. Na základu logaritmu nezáleží, protože jakýkoliv logaritmus lze vyjádřit dle vzorce: $log_a n = log_b n / log_b a$,

$O(n)$−lineární, složitost je závislá pouze na velikosti vstupu,

$O(n \cdot log(n))$−lineárně-logaritmické, složitost je kombinací, předcházejících dvou složitostí,

$O(n^m), m \in \mathbb{N}$−polynomiální, v praxi se často se vyskytuje kvadratická $O(n^2)$ a kubická složitost $O(n^3)$.

Mezi nepolynomiální složitosti se řadí:

$O(c^n), c \in \mathbb{R}^+$−exponenciální, vzhledem k bitovým operacím se velmi často vyskytují algoritmy se složitostí 2^n,

$O(n!)$−faktoriální složitost.

V praxi preferovanými algoritmy jsou polynomiální algoritmy, přesto existují důvody i pro použití algoritmů nepolynomiální složitosti, které jsou popsány dále.

B.3 Výpočetní složitost paralelizovaných algoritmů

Moderní systémy obvykle obsahují více procesorů, na kterých lze současně a nezávisle počítat určitou část řešené úlohy. Tato architektura tedy umožňuje u některých úloh dosáhnout nižší výpočetní (především časové) složitosti, než v případě jednoprocesorového systému. Obvyklým příkladem této praxe je paralelní redukce, na kterou je v práci odkazováno.

Příklad. Je dán vektor $x = (x_1, x_2, ..., x_n)$ a vektor y=$(y_1, y_2, ..., y_n)$. Složitost skalárního součtu s=$\sum_{i=1}^{n} x_i y_i$ na jednoprocesorovém počítači je $O(n)$. Pokud bude paralelní systém obsahovat n procesorů, pak na každém procesoru P lze provést výpočet $a_i = x_i y_i$ v čase $O(1)$. Budeme-li v dalším kroku sčítat výsledek a_i a a_{i+1} vznikne $a_i = a_i + a_{i+1}$a sníží se počet potřebných procesorů o polovičku. Tak tomu bude v každém kroku až se získá skalární součin s=$\sum_{i=1}^{n} a_i$. Počet redukcí odpovídá binárnímu stromu o n listech, tedy celková doba výpočtu v závislosti na počtu kroků je rovna $O(log_2 n) = O(log\, n)$, dle vzorce B.2.

Algoritmy jsou mezi sebou porovnávány také na základě druhu problému, proto v následujícím odstavci, se provede úvod do vyčíslitelnosti.

B.4 Třídy problémů P, NP, RP

Přestože se při implementacích preferují algoritmy s polynomiální složitostí, existují problémy (budou označovány jako úlohy), kde nalezení řešení je zatím možné provést pouze použitím algoritmů s nepolynomiální složitostí. [1].

Vzhledem k tomu, že v aritmetice zbytkových tříd se lze setkat s třídami problémů P a PN, je vhodné alespoň částečně shrnout poznatky z teorie vyčíslitelnosti [68, 67].

Definice B.4.1. Turingův stroj (TS) je šestice tvaru M=$(Q,\Sigma,\Gamma,\delta,q_0,q_F)$, kde Δ je prázdným symbolem a L,R představuje posun doleva nebo doprava:

- Q je konečná množina stavů,
- Σ je konečná množina vstupních symbolů, $\Delta \notin \Sigma$,
- Γ je konečná množina páskových symbolů $\Sigma \subseteq \Gamma$,$\Delta \in \Gamma$,
- δ je přechodová funkce: $(Q\backslash\{q_F\}) \times \Gamma \rightarrow Q \times (\Gamma \cup \{L,R\})$, kde $L,R \notin \Gamma$,
- q_0 je počáteční stav, $q_0 \in Q$,
- q_F je konečný stav,$q_F \in Q$.

Před začátkem algoritmu je na pásce umístěn vstup v prvních n buňkách, zbytek pásky je vyplněn prázdným symbolem Δ (blank). Hlava Turingova stroje ukazuje na první buňku vstupu a řídící jednotka je v počátečním stavu. Po přečtení první buňky (symbolu X) a na základě přechodové funkce δ přejde Turingův stroj do stavu p, přepíše symbolem Y a posune se o jedno místo vpravo R (nebo vlevo L). Popsaná akce se zapisuje jako: $\delta(q_0,X) \rightarrow (p,Y,R)$. Turingův stroj dle přechodové funkce prochází pásku a zapisuje, až se dostane do konečného stavu q_F, kdy výpočet končí. Turingovým strojem lze řešit problém rozhodnutelnosti úlohy U, tedy schopnost určit (přesněji říci ANO - NE), zda-li jazyk L (přesněji množina řetězců jazyka L tvaru $w \in \Sigma^*$), jsou daným Turingův strojem přijímány (tedy stroj přejde do stavu q_F). Na základě popsaného modelu jsme schopni definovat třídy složitosti.

Definice B.4.2. Rozhodovací úloha U leží ve třídě problému P, právě pokud existuje deterministický Turingův stroj (přechodová funkce poskytuje pro daný vstup a stav TS pouze jednu možnost přechodu, což platí pro všechny konfigurace), který rozhodne jazyk L v polynomiálním čase.

Příkladem úloh z třídy P je nalezení nejkratší cesty v acyklickém grafu mezi uzly a a b, kde délka je nejvýše k nebo nalezení minimální kostry grafu, jejíž cena je maximálně rovna k.

Definice B.4.3. Rozhodovací úloha U leží ve třídě problému NP, právě pokud existuje nedeterministický Turingův stroj (přechodová funkce umožňuje přejít z jednoho stavu pro daný vstup do více různých stavů), který rozhoduje jazyk L v polynomiálním čase.

Jednoduše lze říci, že výsledek lze ověřit v polynomiálním čase. Příkladem úloh z třídy NP může být problém obchodního cestujícího (TSP problem [32]). Existence řešení $P = NP$, tedy možnosti, že lze všechny nepolynomiální řešení problému převést na polynomiální je jednou z otevřených otázek současné matematiky a informatiky [2].

[1] ověřit řešení některých problémů z teorie čísel lze provést v polynomiálním čase, ale nalezení řešení je možné pouze v nepolynomiálním čase, čehož využívá současná kryptografie

[2] více lze nalézt: http://www.claymath.org/about/mission.php

Vzhledem k tomu, že v práci se nachází pravděpodobnostní algoritmus, je vhodné uvést i teorii k třídě *RP*.

Definice B.4.4. Randomizovaný Turinguv stroj (*RTS*), je nedeterministický vícepáskový stroj *TS*, skládající se nejméně ze dvou pásek, první páska je totožná s předcházející definicí *TS*, druhá páska (lze pouze číst) obsahuje symboly $\Sigma \in \{0,1\}$ s pravděpodobností 0.5 pro každý ze symbolů 0,1.

Nyní se bude uvedena třída *RP*.

Definice B.4.5. Jazyk *L* patří do třídy *RP* (randomizovaně polynomiální) [23], pokud existuje *RTS* takový, že:

- pokud $w \notin L$, tak *RTS* skončí v koncovém stavu s pravděpodobností $p(w) = 0$,
- pokud $w \in L$, tak *RTS* skončí v koncovém stavu s pravděpodobností $p(w) \geq 0,5$.

Mezi nejvýznamnější úlohy z této třídy patří Millerův-Rabinův test prvočíselnosti, který je zmíněn v kapitole 8.

Příloha C

Druhy paralelních systémů a jejich výkonostní analýza

C.1 Úvod do druhů paralelních systémů

Za paralelní systém je považován systém, kde může probíhat více procesů současně. Paralelismu se používá ke zvýšení výkonu číslicového systému jiným způsobem než je jeho technologická inovace. Vhodným rozdělením problému na více nezávislých podproblémů, jejichž řešení může probíhat současně, se získá celkové řešení za kratší čas. Tedy lze hovořit o nárůstu výkonu číslicového systému. Podle úrovně složitosti (granularity, zrnitosti) paralelního procesu se dělí paralelní systémy na následující úrovně:

1. úroveň příkazů a instrukcí,
2. úroveň cyklů a iterací,
3. úroveň podprogramů,
4. úroveň části úloh a programů,
5. úroveň nezávislých úloh a programů.

Vývoj paralelních systémů probíhal od nižších úrovní k vyšším. Předkládaná práce se zabývá architekturou založenou na úrovních 1 až 3. Obecně nejpoužívanější klasifikací paralelních systémů v úrovni 1 je Flynntova klasifikace.

C.2 Flynntova taxonomie (klasifikace) paralelních systémů

Flynntova taxonomie je používána od roku 1966. Popisuje pouze von Neumannovské architektury, tedy nezabývá se redukčními[1] a data-flow architekturami[2]. Používané architektury rozlišuje podle toku instrukcí a toku dat, vzájemnou kombinací těchto toků se získají následující skupiny:

[1] redukční (stromová) architektura - při výpočtu jsou uzly stromu postupně redukovány mezivýsledky, kdy konečná fáze výpočtu představuje kořen stromu

[2] data-flow architektura - lze si jí představit jako orientovaný graf, kdy do do uzlu grafu vstupují operandy a vystupuje výsledek

- SISD (Single Instruction stream, Single Data stream) je počítač, zpracovávající data sériově podle jednoho programu. Představiteli této skupiny jsou počítače ze 40. let dvacátého století, které byly postaveny podle von Neumannova návrhu.
- SIMD (Single Instruction stream, Multiple Data stream) je počítač používající více stejných jednotek (procesorů) řízených společným programem. Přitom data zpracovávaná v jednotlivých procesorech jsou různá. Tedy existuje jedna instrukce pro více procesorů zpracovávající jiná data. Často se tak označují počítače s vektorovými procesory.
- MIMD (Multiple Instruction stream, Multiple Data stream) je multiprocesorový systém, kdy každý procesor zpracovává vlastní program a pracuje s jinými daty než ostatní procesory. Takto označené jsou většinou distribuované systémy spolupracující skrze zasílání zpráv.
- MISD (Multiple Instruction stream, Single Data stream) je skupina počítačů, kde řetězce procesorů pracují podle různých programů na společných datech a ty si postupně předávají. Označují se jako počítače s řetězcovými procesory.

Klasická Flynnova klasifikace byla později doplněna ještě o několik dalších typů (např.) :

- MSIMD (Multiple Single Instruction stream, Multiple Data stream) je systém, kde pracuje několik podsystémů SIMD nezávisle na sobě. Každý z podsystémů se řídí jiným programem, řízení systému je analogické systému MIMD.
- SPMD (Same Program, Multiple Data Stream) je to modifikace systému SIMD. Všechny procesory vykonávají tentýž program, ale nezávisle na sobě, tedy bez synchronizace. Což vede k tomu, že každý procesor má svůj vlastní řadič a určuje si provádění instrukcí.

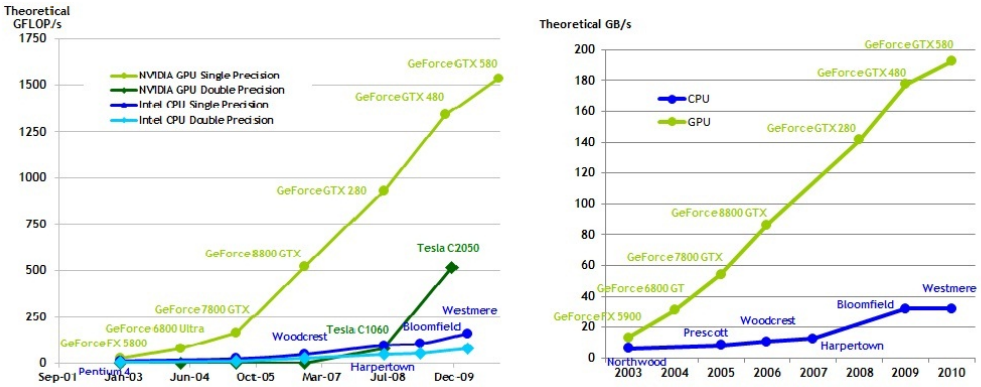
V práci je zmíněna také architektura odvozená od SIMD:

- SIMT (Single Instruction Multiple Threads) - každý z procesorů, které jsou součásti daného multiprocesoru má vlastní aritmetickou jednotku a procesoru je přiřazeno výpočetní vlákno.

Protože na architektuře SIMT jsem provedl několik testů vůči implementaci na architektuře MIMD, stručně její vlastnosti shrnu v následující kapitole.

C.3 Architektura SIMT

Poptávka po vysoce výkonných systémech schopných zpracovávat 3D grafiku vedla k vývoji paralelních multiprocesorových GPU s vysokou propustností. Vzhledem ke zvětšujícím se rozdílům ve výkonu mezi CPU a GPU v plovoucí řádové čárce za sekundu (FLoting-point OPerations per Second) - FLOPS (obrázek propustností pamětí (gigabitů za sekundu) C.3.1), byla snaha využít GPU i pro paralelní úlohy mimo 3D grafiku.



Obrázek C.3.1: Teoretický výkon a propustnost GPU

Toho bylo dosaženo nejprve pomocí programovacích jazyků CUDA (fy nVidia) a ATI Stream (fy ATI), později byl vytvořen standard OpenCL, který požadavky na paralelní programování moderních architektur slučoval.

Rozdíl výkonů mezi CPU a GPU je dán specializací GPU na náročné a vysoce paralelní výpočty, které jsou potřebné při renderování 3D grafiky. Schématická podoba CPU a GPU je znázorněna na obrázku C.3.2.

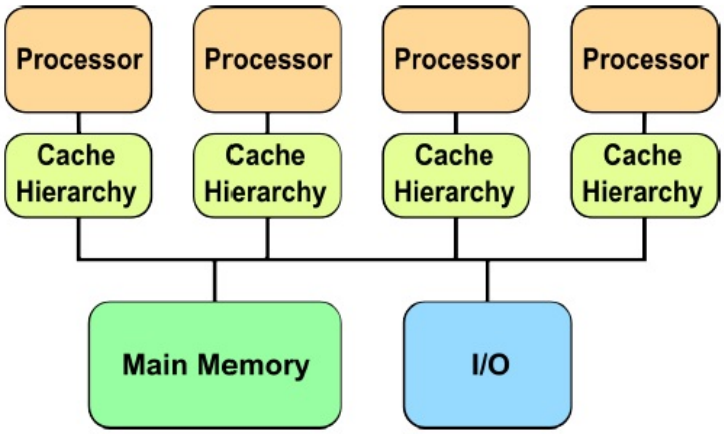
Obrázek C.3.2: CPU vs GPU

Z obrázku je zřejmé, že architektura GPU se hodí především pro datový paralelismus, kdy stejný program je paralelně spuštěn na velkém počtu datových elementů - což značí velkou intenzitu aritmetických operací a velkou paměťovou propustnost architektury. Protože jsou spuštěny stejné instrukce na všech datových elementech, není zapotřebí tak sofistikované řídící logiky jako v případě CPU. Přístup k paměti je skryt pod intenzitou prováděných operací, proto může být vyrovnávací paměť mnohem menší než v případě CPU.

Jak bude ukázáno dále v publikovaných testech, architektura SIMT i přes své uplatnění v řadě současných aplikací, se nejeví jako vhodná v případě výpočtu RNS. Tento důvod je dán především jednoduchostí návrhu instrukčních sad v proudových procesorech, které neumožňují efektivní práci nad bitovými datovými buňkami, druhým problémem je pomalost globální paměti, jež je použita pro čtení a zápis výsledků provedených aritmetických operací.

C.4 Architektura MIMD

Tato architektura existovala dříve pouze na úrovni clusterů, v dnešní době lze nalézt i v domácích počítačích. Struktura lze popsat následujícím diagramem, který ukazuje, že jednotlivé procesory jsou spojeny s hierarchickou pamětí, která umožňuje snížit datovou zátěž na sběrnici mezi hlavní pamětí a jednotlivými procesory.



Obrázek C.4.1: Architektura MIMD

Při testech a implementaci RNS (FP RNS) se během zpracování práce využívat především tento typ architektury, kdy lze jednotlivá modula spouštěn nezávisle na jednotlivých procesorech a společné sběrnice pro komunikaci mezi procesory se využívá pouze při rekonstrukci čísla do standardní architektury.

C.5 Úvod do výkonnostní analýzy paralelních systémů

Z důvodu zaměření práce na paralelní architektury jsou zde uvedeny nejčastější metriky používané pro porovnání paralelních výpočtů (Amdahlův zákon [3], Gustafsonův zákon [61], Karp - Flattova metrika viz kapitola C.8, Iso - efektivní metrika viz kapitola C.9).

V metrikách se vychází z následujících pojmů:

Definice C.5.1. Celkový čas sériového výpočtu $T_s(n)$, závisí na velikosti problému n a je definován jako součet času výpočtu sériové části $\sigma(n)$ a času výpočtu paralelní části $\phi(n)$:

$$T_s(n) = \sigma(n) + \phi(n). \tag{C.5.1}$$

Definice C.5.2. Režie komunikace paralelního výpočtu $\kappa(n,p)$ je závislá na velikosti problému n a počtu procesorů p používaných k řešení problému. Je součtem režijních časů paralelizace (celkový čas inicializace paralelního výpočtu $T_{ini}(n,p)$, celkový čas synchronizace procesorů $T_{syn}(n,p)$, celkový čas komunikace mezi procesory $T_{kom}(n,p)$):

$$\kappa(n,p) = T_{ini}(n,p) + T_{syn}(n,p) + T_{kom}(n,p). \tag{C.5.2}$$

Definice C.5.3. Celkový čas paralelního výpočtu $T_p(n,p)$ se získá jako součet sériového času a průměrného paralelního času výpočtu na p procesorech:

$$T_p(n,p) = \sigma(n) + \frac{\phi(n)}{p}. \tag{C.5.3}$$

Definice C.5.4. Urychlení výpočtu $\psi(n,p)$ se určí se jako podíl sériového času (nebo času výpočtu na jednom paralelním procesoru) a paralelního času výpočtu (spuštěný na p procesorech):

$$\psi(n,p) = \frac{T_s(n)}{T_p(n,p)}. \tag{C.5.4}$$

Definice C.5.5. Rychlost paralelního výpočtu $R_p(n,p)$ je závislá na velikosti problému n a počtu procesorů p. Je dána poměrem počtu zpracovaných instrukcí $W(n)$ vůči celkovému času paralelního výpočtu $T_p(n,p)$C.5.5:

$$R_p(n,p) = \frac{W(n)}{T_p(n,p)}. \tag{C.5.5}$$

Definice C.5.6. Efektivnost paralelního výpočtu $\varepsilon(n,p)$ lze vyjádřit jako podíl sériového času vůči paralelnímu času vykonanému na p procesorech C.5.6:

$$\varepsilon(n,p) = \frac{T_s(n)}{p \cdot T_p(n,p)}, \tag{C.5.6}$$

a nebo jako poměr urychlení na p procesorech C.5.7:

$$\varepsilon(n,p) = \frac{\psi(n,p)}{p}. \tag{C.5.7}$$

Definice C.5.7. Reálné urychlení výpočtu $\psi_r(n,p)$ je definováno jako čas sériového výpočtu a paralelního času výpočtu s režií:

$$\psi_r(n,p) = \frac{T_s(n)}{T_p(n,p) + \kappa(n,p)}.$$

Při použití Amdahlova a Gustafsonova zákona budou následující nerovnosti zachovány:

$$\psi_r(n,p) \leq \psi(n,p). \tag{C.5.8}$$

Definice C.5.8. Efektivita paralelního výpočtu $\varepsilon_r(n,p)$ je definována jako:

$$\varepsilon_r(n,p) = \frac{T_s(n)}{p \cdot (T_p(n,p) + \kappa(n,p))}, \tag{C.5.9}$$

a splňuje nerovnostiC.5.10:

$$0 \leq \varepsilon_r(n,p) \leq 1. \tag{C.5.10}$$

Tedy není možné dosáhnout vyšší efektivity výpočtu než v případě sériového výpočtu (kde není paralelní režie).

C.6 Amdahlův zákon

Zákon byl publikován v roce 1967 a je významný především stanovením teoretické meze, které lze paralelním výpočtem dosáhnout za předpokladu konstantního rozložení sériové a paralelní části výpočtu.

Věta C.6.1. *Označme funkci $f(n)$ vyjadřující poměr času sériového výpočtu vůči celkovému času výpočtuC.6.1:*

$$f(n) = \frac{\sigma(n)}{\sigma(n) + \phi(n)}. \tag{C.6.1}$$

***Amdalovým zákonem** se nazývá vztah vyjadřující urychlení výpočtu pomocí funkce $f(n)$:*

$$\psi(n,p) == \frac{1}{f(n) + \frac{1-f(n)}{p}}.$$

Důkaz. Vzorec C.6.1 lze upravit na tvarC.6.2:

$$\sigma(n) = \frac{f(n)}{1-f(n)} \cdot \phi(n), \tag{C.6.2}$$

Na základě výše uvedeného vzorce a vztahů C.5.1, C.5.2, C.5.3 je vyjádřeno urychlení výpočtu následovně:

$$\psi(n,p) = \frac{T_s(n)}{T_p(n,p)} = \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p}} = \frac{\frac{f(n)}{1-f(n)} \cdot \phi(n) + \phi(n)}{\frac{f(n)}{1-f(n)} \cdot \phi(n) + \frac{\phi(n)}{p}}, \tag{C.6.3}$$

po úpravách vzniká následující tvar Amdahlova zákona

$$\psi(n,p) = \frac{1}{f(n) + \frac{1-f(n)}{p}}. \tag{C.6.4}$$

□

Na závěr je třeba zmínit, že možnosti paralelizace výpočtu (vlivem sériových částí) jsou vždy omezené a proto urychlení výpočtu problému $\psi(n,p)$ bude vždy menší než $1/f(n)$ (při $p \to \infty$).

C.7 Gustafsonův zákon

Amdahlův zákon se zabývá především urychlením (minimalizací času) řešení problému o neměnící se velikosti jeho složek (podíl sekvenční a paralelní části ve výpočtu se s počtem procesorů nemění).

V dalších letech se však při praktických testech ukázalo, že lze najít případy, kdy Amdahlův zákon neplatí. Proto byl v roce 1988 Gustafsonem publikován jeho zákon, který je identický s Amdalovým [62], avšak konstantou není velikost problému (Amdahl), ale čas jeho řešení. Říká jaká velikost problému v závislosti na počtu procesorů, je časově přijatelná.

Věta C.7.1. *Označme funkci $\alpha(n)$ vyjadřující poměr času sériové části k celkové:*

$$\alpha(n) = \frac{\sigma(n)}{\sigma(n) + \frac{\phi(n)}{p}}. \tag{C.7.1}$$

***Gustafsonovým zákonem** se nazývá vztah pro zrychlení tvaru:*

$$\psi(n,p) = \alpha(n) + p \cdot (1 - \alpha(n)). \tag{C.7.2}$$

Důkaz. Ze vztahu C.7.1 se vyjádří $\sigma(n), \phi(n)$ následovně:

$$\sigma(n) = \alpha(n) \cdot (\sigma(n) + \frac{\phi(n)}{p}),$$

$$\phi(n) = p \cdot (1 - \alpha(n)) \cdot (\sigma(n) + \frac{\phi(n)}{p}).$$

Pro zrychlení bude platit:

$$\psi(n,p) = \frac{T_s(n)}{T_p(n,p)} = \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p}} = \frac{(\alpha(n) + p \cdot (1 - \alpha(n))) \cdot (\sigma(n) + \frac{\phi(n)}{p})}{\sigma(n) + \frac{\phi(n)}{p}}. \tag{C.7.3}$$

Po úpravách lze získat tvar:

$$\psi(n,p) = \alpha(n) + p \cdot (1 - \alpha(n)), \tag{C.7.4}$$

což je Gustafsonův zákon. □

Na základě vztahů C.7.1, C.7.2 lze odvodit stanovit následující závěry:

- pokud $\sigma(n) \gg \phi(n)$ pak $\alpha(n) \approx 1$ a $\psi(n,p) \approx 1$
- pokud $\sigma(n) \ll \phi(n)$ pak $\alpha(n) \approx 0$ a $\psi(n,p) \approx p$

Rozdíl mezi Gustafsonovým a Amdahlovým zákonem je především rozdílný pohled na urychlení výpočtu paralelních architektur. Amdahl předpokládá, že rozsah paralelní a sériové části se nemění a v této situaci je zrychlení limitováno, naproti tomu Gustafson předpokládá, že podíl sériové části výpočtu bude s velikostí problému n klesat a urychlení není omezeno.

C.8 Karp - Flatt metrika

V roce 1990 byla publikována Karp - Flatt metrika jako doplnění Amdahlova a Gustafsonova zákona, která je měřítkem paralelizace implementace v paralelním systému.

Definice C.8.1. Sekvenční frakcí výpočtu nazýváme vztah:

$$e(n) = \frac{\sigma(n)}{T_s(n)}. \tag{C.8.1}$$

Věta C.8.2. ***Karp-Flattovou metrikou*** *nazýváme vztah:*

$$e(n,p) = \frac{\frac{1}{\psi(n,p)} - \frac{1}{p}}{1 - \frac{1}{p}}.$$

Důkaz. Ze vztahu C.8.1 se vyjádří $\sigma(n)$:

$$\sigma(n) = T_s(n) \cdot e(n).$$

S využitím $T_s(n) = \sigma(n) + \phi(n)$ lze i $\sigma(n)$ zapsat pomocí sekvenční frakce výpočtu:

$$\phi(n) = T_s(n) \cdot (1 - e(n)).$$

Nyní bude vyjádřen celkový čas paralelního výpočtu pomocí výše uvedených vztahů:

$$T_p(n,p) = \sigma(n) + \frac{\phi(n)}{p} = e(n) \cdot T_s(n) + \frac{(T_s(n) \cdot (1 - e(n)))}{p}.$$

Rovnici se vydělí $T_s(n)$ a vznikne:

$$\frac{T_p(n,p)}{T_s(n)} = e(n) + \frac{(1-e(n))}{p}.$$

Vzhledem ke zrychlení lze rovnici přepsat na:

$$\frac{1}{\psi(n,p)} = e(n) + \frac{(1-e(n))}{p}.$$

Vyjádřením$e(n)$ se získá:

$$e(n,p) = \frac{\frac{1}{\psi(n,p)} - \frac{1}{p}}{1 - \frac{1}{p}},$$

čímž je věta dokázána. □

Na základě Karp - Flattova vztahu je možno určit režii paralelního výpočtu. Vztah se používá také pro detekci zdrojů neefektivnosti paralelní implementace.

C.9 Iso - metriky

Tyto metriky jsou důležité pro stanovení škálovatelnosti paralelního systému, čímž se rozumí schopnost řešit problém o vzrůstající velikosti n se zvyšujícím se počtem procesorů p (případně jejich pamětí), bez ztráty efektivnosti.

Iso - efektivní metrika

Definice C.9.1. Celkový čas výpočtu p procesorů nevykonaného sekvenčním algoritmem se značí $T_0(n,p)$ a platí pro něj:

$$T_0(n,p) = (p-1) \cdot \sigma(n) + p \cdot \kappa(n,p).$$

Věta C.9.2. *Označme konstantu* $C_0 = \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)}$, *kde* $\varepsilon(n,p)$ *je efektivita.* ***Iso - efektivní relací*** *nazveme vztah:*

$$T_s(n) \geq C_0 \cdot T_0(n,p).$$

Důkaz. Vzorec pro urychlení $\psi(n,p)$ se rozšíří počtem procesorů p:

$$\psi(n,p) = \frac{p \cdot (\sigma(n) + \phi(n))}{p \cdot \sigma(n) + \phi(n) + p \cdot \kappa(n,p)}.$$

Předcházející rovnice se přepíše do tvaru:

$$\psi(n,p) = \frac{p \cdot (\sigma(n) + \phi(n))}{\sigma(n) + \phi(n) + (p-1) \cdot \sigma(n) + p \cdot \kappa(n,p)}.$$

Dosazením $T_0(n,p)$ do vzorce pro urychlení se dostane:

$$\psi(n,p) = \frac{p \cdot (\sigma(n) + \phi(n))}{\sigma(n) + \phi(n) + T_0(n,p)}.$$

Dělením počtem procesorů p se získá vzorec pro efektivitu:

$$\varepsilon(n,p) = \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n) + T_0(n,p)}.$$

Vydělením čitatele i jmenovatele výrazem $\sigma(n) + \phi(n)$ se dostane:

$$\varepsilon(n,p) = \frac{1}{1 + \frac{T_0(n,p)}{\sigma(n)+\phi(n)}}.$$

Dosazením T_s do vzorce se efektivita vyjádří jako:

$$\varepsilon(n,p) = \frac{1}{1 + \frac{T_0(n,p)}{T_s}}.$$

Vzhledem k T_s se získá vztah vztah:

$$T_s(n) = \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} T_0(n,p),$$

kde zlomek $\frac{\varepsilon(n,p)}{1-\varepsilon(n,p)}$ je nahrazen konstantou C_0. Tedy pro zachování stejné úrovně efektivity je nutné, aby s počtem procesorů p rostla i velikost problému n a byla splněna nerovnost:

$$T_s(n) \geq C_0 \cdot T_0(n,p).$$

□

Definice C.9.3. Předpoklad, že iso - efektivní metrika pro problém o složitosti n, řešená na paralelní architektuře je funkce závislá na počtu procesorů p, tedy $f(p)$.

Pro problém o složitosti n platí (složitost sériového algoritmu výpočtu je vyšší než u paralelního, což odpovídá delšímu času výpočtu):

$$n \geq f(p).$$

Definice C.9.4. Funkce $M(n)$, $(M(f(p)))$ je označena jako paměť procesoru potřebná pro uložení problému o složitosti n, $(f(p))$.

Pro velikost paměti bude platit vztah:

$$M(n) \geq M(f(p)).$$

Pro množství paměti potřebné na každý procesor platí:

$$\frac{M(n)}{p} \geq \frac{M(f(p))}{p}.$$

Protože celkové množství paměti je lineární funkcí počtu procesorů, je možné napsat:

$$f_s(p) = \frac{M(f(p))}{p}.$$

Definice C.9.5. Funkce $f_s(p)$ se nazývá funkcí škálovatelnosti paralelního systému o počtu procesorů p.

Funkce $f_s(p)$ určuje, jak se musí paměť procesoru pro problém o velikosti n zvýšit, aby bylo dosaženo stejné iso - efektivnosti při jeho řešení. Na základě složitosti funkce škálovatelnosti lze učinit několik závěrů:

- pokud bude $f_s(p) \in O(1)$, pak je velikost paměti (potřebné na řešení problému) na každý procesor konstantní a systém je nejlépe škálovatelný,
- pokud bude $f_s(p) \in O(p)$, pak velikost paměti na každý procesor roste lineárně a za určitých okolností se dá dosáhnout dobré škálovatelnosti (avšak je možno narazit na limit paměti, pak klesne efektivita),
- pokud bude $f_s(p) \in O(log(p))$ nebo $f_s(p) \in O(p \cdot log(p))$ bude závěr obdobný jako u předcházejícího případu.

Obecně tedy platí, že čím nižší je složitost funkce $f_s(p)$, tím lépe je systém škálovatelný.

Tvrzení C.9.6. Pro iso - paměť platí:

$$\frac{M(n)}{p} = konst.$$

Iso - paměť říká jak velký musí být řešený problém s ohledem na počet procesorů, aby velikost paměti použité pro výpočet byla konstantní.

Tvrzení C.9.7. Pro iso - čas platí následující vztah:

$$\sigma(n) + \frac{\phi(n)}{p} + \kappa(n,p) = konst.$$

Iso - čas určuje, jak se mění rozsah problému n a jeho sériových a paralelních částí, aby byl čas konstantní.

Vztah mezi efektivností a časem běhu výpočtu

Definicí efektivnosti $\varepsilon_p(n,p)$ paralelního systému dle C.5.6, lze získat:

$$\varepsilon_p(n,p) = \frac{\frac{T_s(n)}{p}}{\frac{T_s(n)}{p} + \frac{T_0(n,p)}{p}}.$$

Na základě výše uvedeného vztahu je možno odvodit následující závěry:

- pokud bude iso - časová funkce konstantní:

  $$\frac{T_s(n)}{p} + \frac{T_0(n,p)}{p} = konst.,$$

  pak také efektivita bude konstantní a paralelní systém je škálovatelný,
- pokud čas paralelního výpočtu je funkcí (n/p), pak iso - časová a iso - efektivní funkce roste lineárně s počtem procesorů a paralelní systém je škálovatelný,
- iso - časová funkce roste lineárně, pouze pokud má algoritmus paralelního výpočtu lineární složitost $O(n)$,
- pokud iso - časová funkce roste lineárně, pak také iso - efektivní funkce roste lineárně a systém je škálovatelný,
- pokud iso - efektivita roste lineárně a časová složitost roste lineárně, pak také iso - čas roste lineárně a systém je škálovatelný.

Tedy závěrem lze říci, že systém je dobře škálovatelný, pouze pokud paralelní výpočet má nejhůře lineární výpočetní složitost. V implementovaném FP RNS je využíváno algoritmů jejichž časová složitost bývá nejčastěji $O(log\,n)$ (algoritmy pro bitové sčítání, odčítání a násobení jsou složitosti) nebo$O(n)$ v případě rekonstrukce čísla (ale tuto složitost lze snížit až na $O(log\,n)$ při použití paralelní redukce).

C.10 Reálná rychlost operací prováděných počítačem

V případě volby vhodných algoritmů pro implementovaný paralelní systém je efektivní se orientovat podle počtu a druhu použitých aritmetických operací C.10.1, které musí jednotlivé procesory vykonat.

operace	váha
$+,-,\times$	1
:	10
$\sqrt{}$	30
sin, cos	50
exp	55

Tabulka C.10.1: Operace s plovoucí čárkou

Protože některé operace s plovoucí čárkou jsou náročnější než jiné, používá se váha, která charakterizuje jejich náročnost ve srovnání se sčítáním [12], které má váhu 1.

Algoritmus pro multiplikativní násobení, který jsem navrhl a implementoval 5.8, využívá pouze operace $+, -, \times$ a bitové posuny, což značně urychluje výpočet, neboť není potřeba používat operaci $/$. Výhodou výpočtu ODR pomocí metody Taylorovy řady je také transformace zadání pomocí blokového schématu, umožňující zjednodušit složitější goniometrické funkce *sinus*, *cosinus* na operace $+, -, \times$.