PRÁCA S PRESNOU REÁLNOU ARITMETIKOU

Ročníkový projekt
(Zimný semester)

Meno a priezvisko: Damián Regeš

Školiteľ: doc. Dr. Tomáš Plachetka

Akademický rok: 2023/2024

OBSAH

ÚVOD

Počítanie s číslami je samozrejme hlavnou úlohou počítača. Ale oveľa závažnejším problémom je zaznamenanie a počítanie s potenciálne nekonečnými presnými reálnymi číslami v konečných bežných počítačoch. Štandardne sa v počítačoch počíta s určitou presnosťou, a preto sa výsledky výpočtov zaokrúhľujú. Toto zaokrúhľovanie, ale môže v komplikovaných výpočtoch spôsobiť veľkú chybu konečného výsledku od naozajského reálneho výsledku. Špeciálne vo výpočtovo náchylných oblastiach, ako sú fyzika alebo medicína, takáto chyba môže mať závažné následky. Preto sme sa rozhodli zaoberať sa s Exact Real Arithmetic (ERA), teda s presnou reálnou aritmetikou. Táto aritmetika počíta s číslami bez straty presnosti pri aritmetických operáciách, ktoré sú bežne spôsobené zaokrúhľovaním medzivýsledkov pri výpočtoch. Konkrétne sa budeme zaoberať rôznymi už implementovanými knižnicami zaoberajúcimi sa ERA.

Cieľ práce

Cieľom tohto ročníkového projektu je v prvom rade získať skúsenosti, ktoré budú použité pri vypracovaní bakalárskej práce. Tento projekt sa bude zameriavať na problematiku Exact Real Arithmetics, s hlavným cieľom porovnaním funkcionalít existujúcich knižníc, ktoré sa venujú danej problematike. Sekundárnym cieľom je porovnanie výkonu daných knižníc a ich reálne využitie v numericky náročnom programe. Ďalšími cieľmi projektu sú porozumenie implementácií daných knižníc, prípadné rozšírenie ich funkcionality.

1 Popis ERA knižníc

Existuje mnoho knižníc v mnohých programovacích jazykov zaoberajúcim sa problematikou ERA. My sme sa rozhodli zaoberať najmä knižnicami napísaných v programovacích jazykoch C a C++.

1.1 xr

Prvou a jedinou knižnicou implementujúcu ERA napísanou v jazyku C, ktorou sme sa zaoberali, je knižnica XR od autora Keith Briggs. Oproti alternatíve – multiple-precision floating-point, nie je nutné v knižnici nastavovať presnosť pred samotným výpočtom a zároveň si môžeme byť istý (podľa autora) konečným výsledkom. Naopak intervalová aritmetika je v tomto ohľade lepšia, ale pri nej je ťažšie obnoviť výpočet s väčšou presnosťou. Naopak v prípade tejto knižnice nie je vopred zadaná presnosť a teda neprebieha žiaden výpočet, kým nie je zadaná konečná požiadavka na výstup. Autor vyvinul podobné kódy aj v pythone a aj v C++. (Briggs, 2013)

1.1.1 Popis implementácie

Funkcie v knižnici xr pracujú s presnými reálnymi číslami pomocou celočíselnej reprezentácie vynájdenej Boehm a kol. a ďalej rozvíjané Ménisssier-Morain. V tomto systéme je reálne číslo x reprezentované ako funkcia x: {Z +} -> Z, splnená pre všetky n >= 0 a pre nejaké pevne dané celé číslo B > 0: $|B^nX – x(n)| < 1$, n = 1, 2, 3,..., kde X je reálna reprezentácia čísla x. Všetky výpočty tak závisia od hodnoty B – granularity. Táto hodnota môže byť zmenená, ale predvolene je nastavená na hodnotu 2. (Keith, 2013)

1.1.2 Funkcie

Inicializácia a mazanie

xr_set_b(int b), xr_get_b() – nastavenie a zistenie hodnoty granularity

xr_t xr_init(long n, long d) – inicializácia reálneho čísla ako zlomok n/d

xr_free(const xr_t x) – zmazanie čísla x a teda uvoľnenie pamäte

Unárne aritmetické operácie

xr_abs(const xr_t x), xr_neg(x), xr_recip(x), xr_sqr(x), xr_sqrt(x), xr_root(const xr_t x, long n) – absolútna hodnota, záporná hodnota, obrátená hodnota, druhá mocnina, druhá odmocnina, n-tá odmocnina

Binárne aritmetické operácie

xr_add(const xr_t x, const xr_t y), xr_sub(x,y), xr_mul(x,y), xr_div(x,y) – sčítanie, odčítanie, násobenie a delenie dvoch presných reálnych čísiel

Ostatné funkcie a aritmetické operácie

xr_iadd(const long n, const xr_t x), xr_isub(n,x), xr_imul(n,x) – sčítanie, odčítanie a násobenie celého čísla a presného reálneho čísla

xr_divi(const xr_t x, const long y), xr_powi(x,n), xr_root(x,n) – delenie, umocnenie a odmocnenie presného reálneho čísla celým číslom

xr_cmp(xr_t x, xr_t y) – porovnanie čísiel, vracia 1, ak x > y, vracia -1, ak y > x, nevie rozhodnúť rovnosť;

xr_log2_bound(xr_t x) – vráti celé číslo k také, že $| x / 2^k | < 1$

xr_near_int(xr_t x) – vracia hornú alebo dolnú celú časť čísla

Transcedentálne funkcie

xr_exp(const xr_t x) – vracia Eulerovo číslo umocnené na x

xr_pi(void) - vracia hodnotu pí

Vypísanie hodnoty

xr_get_d(const xr_t x,const int n), char* xr_get_str(x,n), xr_print(x,n), xr_print_nl(x,n) – funkcie snažiace sa získať n správnych desatinných miest x, bez záruky správnosti, funkcie neskončia, ak x = 0, (xr_get_d, interne používa double-precision, teda podlieha obmedzeniu)
xr_dotdump(const char* dotfilename, const xr_t x, const int showcache), zapíše súbor dotfilename.dot vo formáte graphviz dot, na konverziu na grafické znázornenie vnútornej dátovej štruktúry

1.1.3 Aplikácia knižnice

```
#include "xr.h"
int main() {
  xr_t x;
  x=xr_exp(xr_mul(xr_pi(),xr_sqrt(xr_init(163,1))));
  printf("%d\n",xr_cmp(x,xr_near_int(x)));
  return 0;}
```

1.2 RealLib

RealLib je balík na výpočet reálnych čísiel v C++. Posledná verzia RealLib3 umožňuje používateľovi pracovať s reálnymi číslami (Real) alebo na úrovní aproximácií k reálnym číslam (Estimate). Triedy (Real a Estimate) reprezentujú reálne čísla prostredníctvom popisov umožňujúce nekonečne presné výpočty. (Lambov, 2015)

1.2.1 Popis implementácie – Real

Tvorba reálneho čísla Real je vlastne tvorba objektu – inštancie triedy Real. Funkcia aplikovaná na objekt vytvorí nový objekt s referenciou na predchádzajúci objekt. Pri vytvorení reálneho čísla sa odporúča v tejto knižnici použiť inicializáciu pomocou stringu, nie pomocou double (bolo by použitá konštanta s presnosťou double). Pre rýchlejšie počítanie knižnica vnútorne najprv využíva double, až v prípade potrebnej väčšej presnosti použije vlastnú vnútornú aritmetiku. Niektoré implementácie funkcií závisia od implementácií stdlib funkcií sin, cos, log, exp, asin, ftoa teda ich presnosť nie je garantovaná. (Lambov, 2015)

1.2.2 Funkcie a operácie – Real

Inicializácia knižnice

void InitializeRealLib(unsigned precStart = MachineEstimatePrecision, unsigned precMax = 100000, unsigned numEstAtStart = 1000) – inicializuje knižnicu: počiatočná presnosť, maximálna pracovná presnosť, počiatočné vyhradené miesto pre aproximizáciu

#define MachineEstimatePrecision 4 – predvolená inicializácia presnosti

unsigned FinalizeRealLib() – alokovaná pamäť sa uvoľní, zahodia sa aproximizácie, vráti sa použitá presnosť

unsigned ResetRealLib(unsigned precStart) – začne sa pracovať s novou presnosťou

unsigned GetCurrentPrecision() – vráti momentálnu presnosť

Vytváranie a ničenie objektov Real

Real(x) – tvorba objektu, kde x môže byť dané ako double, string, referencia na Real objekt alebo Oracle funkcia

Real objekt sa ničí vymazaním smerníka alebo ak je premenná mimo rozsahu

Unárne aritmetické operácie s Real

Real Real::operator – () const – negácia

Real::Real& operator += (const Real &rhs), -=; *=, /= - skrátené formy pre sčítanie, odčítanie, násobenie, delenie a následne priradenie vypočítanej hodnoty

Real recip(const Real &arg), abs(), sqrt(), rsqrt() – prevrátená hodnota, absolútna hodnota, odmocnina, prevrátená odmocnina

Binárne aritmetické operácie s Real

Real operator + (const Real &lhs, const Real &rhs), -, *, / - sčítanie, odčítanie, násobenie a delenie Real čísel

Ostatné funkcie a operácie s Real

Real log(const Real &arg), exp() – prirodzený logaritmus, umocnené Eulerovo číslo

Real sin(const Real &arg), cos, tan – sínus, kosínus, tangens

Real asin(const Real &arg), acos(), atan(), Real atan2(const Real &y, const Real &x) – arkussínus, arkuskosínus, arkustangens, arkustangens zlomku y/x

bool Real::IsNegative() const, IsPositive(), IsNonZero(), ForceNonZero() – zisťuje, či je číslo záporné, kladné, nenulové – môže vracať probable zero

bool operator < (const Real &lhs, const Real &rhs), >, != – porovnanie menší, väčší, nerovnaký

double Real::AsDouble() const, AsDecimal(char *buffer, unsigned len) – konverzia na double (nie vždy presne zaokrúhlené) a konverzia na desatinné číslo – vytvorí sa desatinná reprezentácia čísla s dĺžkou len (nie vždy presne zaokrúhlené)

Real Konštanty

Extern const Real Pi, Ln2 - konštanty pí a Ln2

Vypísanie hodnoty a načítanie hodnoty Real

std::istream& operator >>(std::istream &in, Real &r) – načítanie konečného reťazca a tvorba reálneho čísla
std::ostream& operator <<(std::ostream &out, const Real& r) – výpis reálneho čísla, nemusí byť pre nekonečné čísla presne a korektne vypísané (kvôli nekonečným binárnym číslam, ktoré môžu mať rôzne reprezentácie v desiatkovej sústave)
setprecision(x) – nastavenie presnosti desatinných čísel
...

1.2.3 Aplikácia knižnice s Real

```
#include <iostream>
#include <iomanip>
#include "Real.h"
using namespace std;
```

```
using namespace RealLib;
void main() {
        InitializeRealLib();
        Real a, b(4);
        b = Pi / b;
        a = sin(b);
        cout << "sin(Pi/4) is " << a << endl;
        FinalizeRealLib();
        return 0;
}
```

1.3 iRRAM

Balík iRAAM je komplexný balík, napísaný v jazyku C++, ponúkajúci možnosť počítať a pracovať s presnou reálnou aritmetikou. Balík ponúka množstvo tried na prácu s: Dyadic Numbers (racionálne číslo s mocninou dvojky v menovatli), Lazy Booleans (lenivé boolean), **Real Numbers (reálne čísla)**, Matrices of real numbers (matice s reálnymi číslami), Sparse matrices of real numbers (riedke matice s reálnymi číslami), Intervals of real numbers (intervaly reálnych čísiel), Complex numbers (komplexné čísla), Integer numbers (celé čísla), Rational numbers (racionálne čísla). (Müller, 2015)

1.3.1 Popis implementácie – REAL

Reálne číslo sa vytvorí vytvorením inštancie triedy REAL. Vnútorne je objekt reprezentovaný hodnotou a chybou. Informáciu o chybe v objekte REAL je možné získať (geterror) alebo aj nastaviť (seterror).

1.3.2 Funkcie a operácie – REAL

Inicializácia knižnice

V knižnici iRRAM sú dva možné módy použitia:

a) Main mode: používateľ definuje funkciu void compute(void); namiesto zvyčajnej funkcie int main(...). Táto funkcia bude použitá ako hlavná funkcia iRRAM slúžiaca na všetky iterácie potrebné na implementáciu ERA.

b) V režime knižnice môže používateľ používať iRRAM ako nástroj na výpočet diskrétnych funkcií interne pomocou ERA viď 1.3.3

Vytváranie objektov REAL

REAL(x) – tvorba objektu, kde x môže byť dané ako int, double, long, string či INTEGER, DYADIC, RATIONAL, REAL number

Unárne aritmetické operácie s Real

REAL operator - (const REAL& x) – negácia

REAL& operator += (REAL& x, const REAL& y), *= – skrátená forma pre sčítanie, násobenie a následne priradenie vypočítanej hodnoty

REAL sqrt(const REAL& x), root(), square(), abs(), round(), round2() – odmocnina, n-tá odmocnina, druhá mocnina, absolútna hodnota, zaokrúhlenie na celé číslo (vracia REAL), zaokrúhlenie na celé číslo (vracia long)

...

Binárne aritmetické operácie s Real

REAL operator + (const REAL& x, const REAL& y), -, *, /, ^ – sčítanie, odčítanie, násobenie, delenie, umocnenie dvoch REAL čísel alebo jedného REAL čísla a jedného long / int čísla

REAL operator << (const REAL& x, long n), >> – n bitový posun doľava a doprava

Ostatné funkcie a operácie s Real

REAL power(const REAL& x, const REAL& y), modulo(), maximum(), minimum() – umocnenie, modulo, maximum a minimum z dvoch čísiel

REAL scale (const REAL& x, long k) – pomer

REAL exp(const REAL& x), log() – Eulerovo číslo umocnené na x, logaritmus

REAL sin(const REAL& x), cos(), tan(), cotan(), sec(), cosec() – sínus, kosínus, tangens, kotangens, sekans, kosekans

REAL asin(const REAL& x), acos(), atan(), acotan(), asec(), acosec() – arkusínus, arkuskosínus, arkustangens, arkuskotangens, arkussekans, arkuskosekans

REAL sinh(const REAL& x), cosh(), tanh(), coth(), sech(), cosech() – hyperbolický sínus, hyperbolický kosínus, hyperbolický tangens, hyperbolický kotangens, hyperbolický sekans, hyperbolický kosekans

REAL asinh(const REAL& x), acosh(), atanh(), acotth(), asech(), acosech() – hyperbolický arkusínus, hyperbolický arkuskosínus, hyperbolický arkustangens, hyperbolický arkuskotangens, hyperbolický arkussekans, hyperbolický arkuskosekan

LAZY_BOOLEAN operator < (const REAL& x, const REAL& y), <=, >, >= – porovnanie menší, menší rovný, väčší, väčší rovný

REAL strtoREAL(char* s, char** endptr), atoREAL() – konverzia reťazca na REAL objekt

void precision_policy (long policy) – nastavenie manuálne aktuálnej presnosti

REAL Konštanty

REAL pi (), euler (), ln2 () – konštanty Pí, Eulerovo číslo a Ln2

Vypísanie a načítanie hodnoty REAL

void rwrite (const REAL& x, const long p, const long w), swrite(), rshow()

void rscanf (const char *format, void * input) – prečítanie hodnoty

void rprintf (const char *rformat, ...) – vypísanie hodnoty

1.3.3 Aplikácia knižnice s REAL

```
#include “iRRAM/lib.h“
using namespace iRRAM;
int iRRAM_compute(const int& dummy) {
        REAL a = 7;
        a = sqrt(a);
        cout << setRwidth(100) << a;
        return 0;
}
int main(int argc, char **argv){
        iRRAM_initialize(argc, argv);
        return iRRAM_exec(iRRAM_compute, 0);
}
```

1.4 Porovnanie

Všetky tri skúmané knižnice (tj. xrc, RealLib, iRRAM) ponúkajú možnosť počítať s presnou reálnou aritmetikou (ERA).

Knižnica xr napísaná v jazyku C ponúka základné aritmetické operátory a porovnanie. Možnou nevýhodou knižnice je absencia implementácie bežne používaných goniometrických funkcií, hoci knižnica ponúka konštantu pí. Ďalšou nevýhodou je nemožnosť nastaviť aktuálnu presnosť ani zistiť momentálnu používanú presnosť.

RealLib knižnica postavená na jazyku C++ ponúka základné aritmetické operácie, porovnania a goniometrické funkcie. Ponúka možnosť počiatočnej inicializácie potrebnej presnosti, taktiež je možné zistiť aktuálnu používanú presnosť vo výpočtoch. Nevýhodou je absencia implementácie niektorých funkcií (napr. hyperbolických, hoci je ich možné ľahko implementovať.

Knižnica iRRAM postavená na jazyku C++ je zo všetkých troch knižníc najkomplexnejšia. Ponúka viacero typov čísiel, medzi ktorými ponúka aj nami zaoberaný typ – reálne číslo. Okrem základných aritmetických operácií a porovnaní ponúka množstvo funkcií, medzi ktorými patria goniometrické a hyperbolické. Tiež je možné vypísať a nastaviť momentálnu presnosť počítania. Veľkou nevýhodou je nemožnosť priamo počítať s reálnou presnosťou v hlavnej funkcií main.

2 Testy a porovnanie ERA knižnice

Prvým krokom bola inštalácia virtuálneho boxu (Oracle VM VirtualBox) so systémom Debian (64-bit) s vyhradenými 4 jadrami procesora a s vyhradenou 8GB RAM. VM bol nainštalovaný na systéme s procesorom Intel i5-8250U. Vo virtual boxe bola vykonáná všetka prácu na tomto ročníkovom projekte, teda inštalácia knižníc a programovanie programov. Graf štruktúry základných priečinkov je umiestnený v prílohe 1.

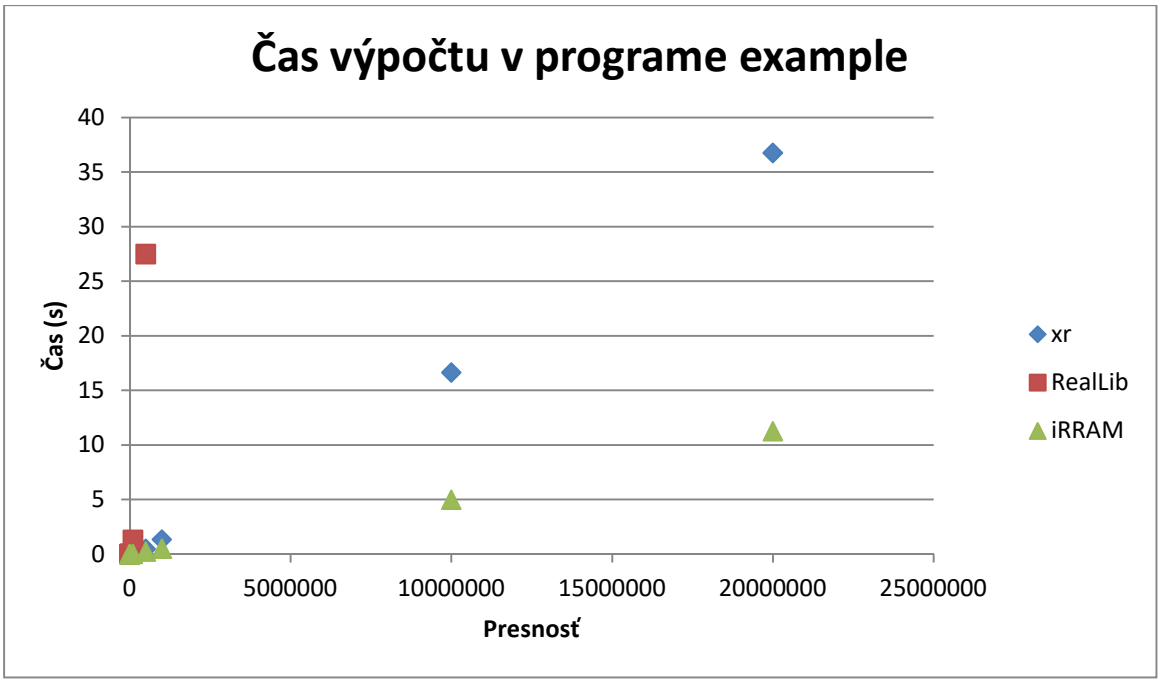
2.1 Testovacie programy

Vytvorili sme rôzne testovacie programy s použitím hore opísaných knižníc. Týmto spôsobom sme mohli otestovať nie len prácu s danými knižnicami, ich možnosti a správanie, ale aj ich rýchlosť pri rôznych použitých presnostiach. Každý program má vyhradený priečinok s rovnomenným názvom s makefile. Kompilácia a generácia alebo vymazanie spustiteľného súboru je teda možné jednoduchým make príkazom. Spustenie vygenerovaného programu sa spúšťa pomocou príkazu: ***./program x***, kde *program* je názov programu a *x* je požadovaná výsledná presnosť. V prípade, že hodnota x nie je zadaná, presnosť je predvolené zadaná na 32 cifier. Po spustení programu, program vypíše výsledok na štandardnú konzolu s danou presnosťou. Zároveň v niektorých prípadoch aj čas behu výpočtu programu uvedený v sekundách.

2.1.1 Test – example

Prvým vytvoreným programom bol program na výpočet jednoduchého príkladu s iracionálnym číslo s nekonečným desatinným rozvojom. Program vypíše výsledok príkladu $\frac{1}{7} + \sqrt{2}$ s vopred danou presnosťou. V nasledujúcej tabuľke a v grafe sú uvedené časy výpočtov (v sekundách) s danou presnosťou v jednotlivých knižniciach.

knižnica/presnosť	1	10	100	10^4	10^5	$5*10^5$	10^6	10^7	$2*10^7$
xr	0.000112	0.000166	0.000200	0.003276	0.07527	0.4715	1.346	16.64	36.76
RealLib	0.000128	0.000137	0.000246	0.0380	1.317	27.49			
iRRAM	0.000234	0.000238	0.000209	0.000950	0.02385	0.229	0.5	4.977	11.26

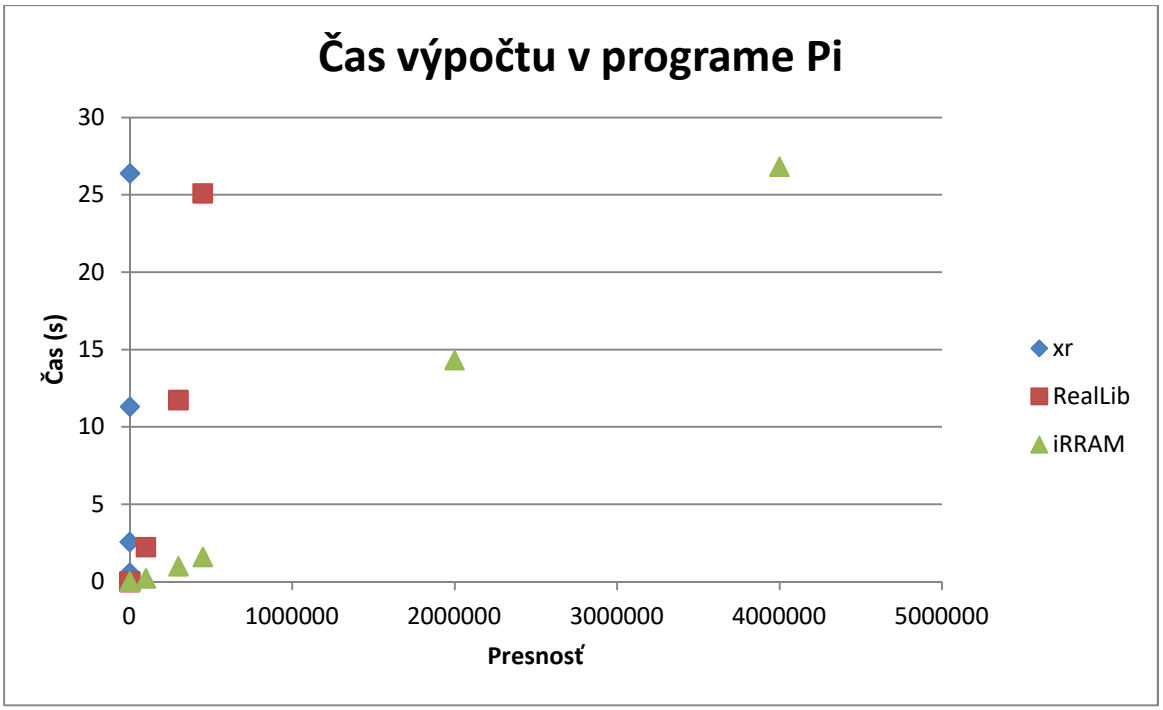


Pri presnostiach 10^0, 10^1, 10^2 rozdiel medzi knižnicami je zanedbateľný. Pri presnosti 10^4 už je rozdiel merateľný, knižnica RealLib je značne pomalšia a už pri presnosti $5*10^5$ výpočet trval približne 27 sekúnd, to je 50 – 100x pomalšie než s knižnicami iRRAM a xr. Pri ďalších meraniach knižnici xr trval výpočet 2 – 4x dlhšie než pomocou knižnice iRRAM.

2.1.2 **Test – Pi**

Z dôvodu už implementovanej konštanty s nekonečným desatinným rozvojom – pí v knižniciach, rozhodli sme sa odmerať rýchlosť daných implementácií v jednotlivých knižniciach.

knižnica/ presnosť	1	10	250	500	1 000	1 500	10^5	$3*10^5$	$4.5*10^5$	$2*10^6$	$4*10^6$
xr	0.000110	0.001172	0.577	2.571	11.315	26.393					
RealLib	0.000053	0.000060	0.0037	0.004235	0.01024	0.01472	2.236	11.74	25.10		
iRRAM	0.000166	0.000260	0.00283	0.000357	0.00049	0.000705	0.2243	1.005	1.607	14.32	26.82

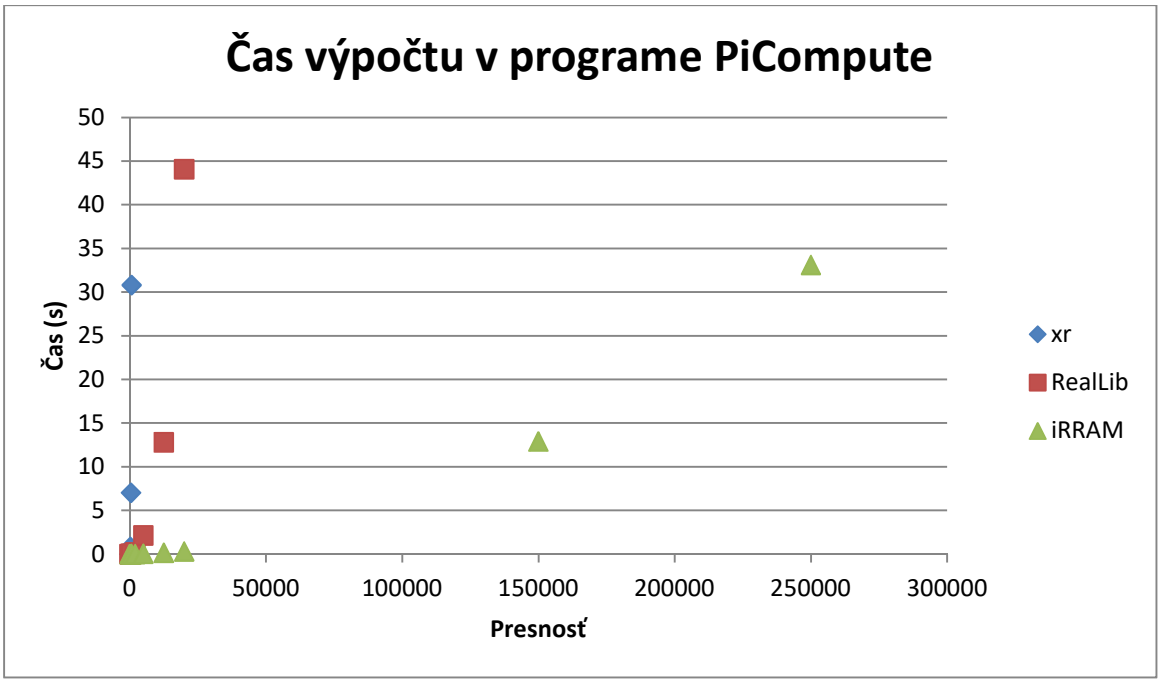


Pri malých presnostiach pí je výpočet a výpis medzi knižnicami nepostrehnuteľný. No už pri 1 000 desatinných miest čísla pí výpis v knižnici xr trval niekoľko sekúnd. Podobnú dobu výpisu dosahovala knižnica RealLib pri sto tisícových presnostiach a knižnica iRRAM pri miliónových presnostiach.

2.1.3 **Test – PiCompute**

Z dôvodu možných rozdielnych algoritmoch použitých v jednotlivých knižniciach na výpočet konštanty pí, sme sa rozhodli implementovať jednotný algoritmus v daných knižniciach. Použili sme Gaussov-Legenrov algoritmus (viď en.wikipedia.org/wiki/Gauss-Legendre_algorithm), s kvadratickou konvergenciou, teda platné cifry sa zdvojnásobujú každou iteráciou algoritmu.

knižnica/ presnosť	1	10	250	500	700	2 000	5 000	$1.25*10^4$	$2*10^4$	$1.5*10^5$	$2.5*10^5$
xr	0.000272	0.00159	0.810916	7.028	30.81						
RealLib	0.000064	0.000163	0.003466	0.01109	0.03871	0.2069	2.145	12.81	44.08		
iRRAM	0.000314	0.000343	0.001164	0.002829	0.006176	0.0187	0.04751	0.152	0.3164	12.898	33.12



Obdobne ako pri počítaní už implementovaného pí v knižniciach doba výpočtu pri malých presnostiach je zanedbateľná. Väčšie rozdiely sú pri väčších požadovaných presnostiach. Najrýchlejší výpočet s rovnakou presnosťou prebiehal v knižnici iRRAM, potom v knižnici RealLib a najpomalší výpočet bol v knižnici xr. Implementácia Gaussovho-Legenrovho algoritmu je vo všetkých knižniciach pomalšia, než už ich vnútorná implementácia konštanty pí.

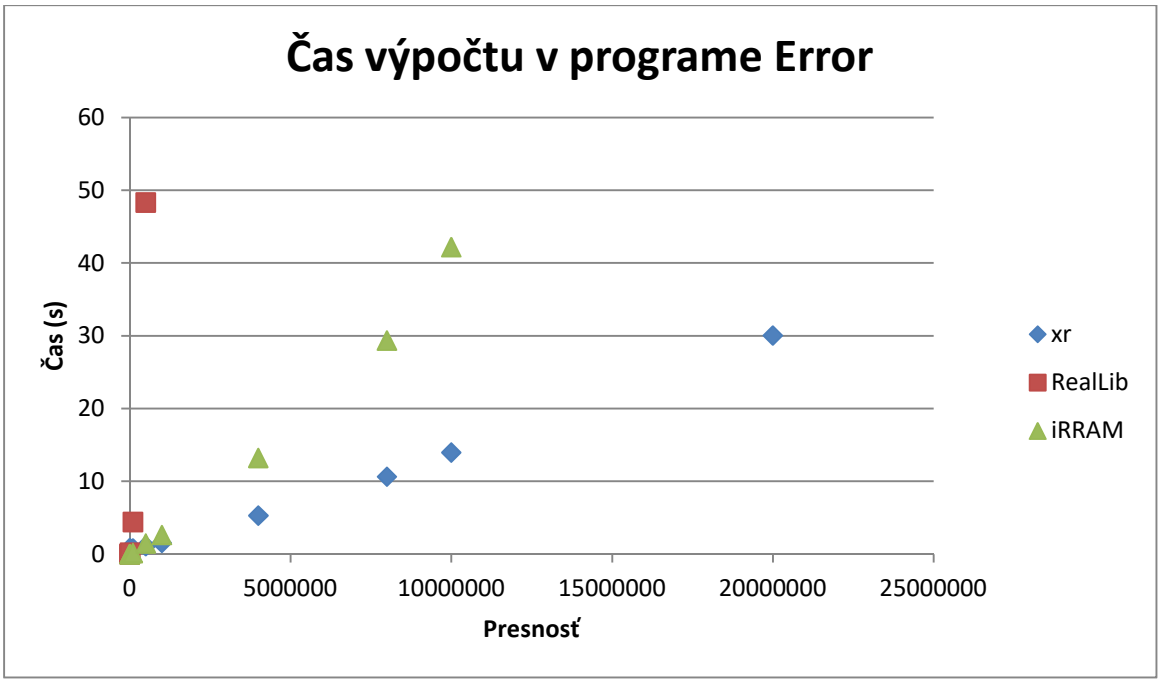
2.1.4 **Test – Error**

Jedným z problémov, ktorý by mali riešiť ERA knižnice, je problém so zaokrúhľovaním čiastkových medzivýsledkov, ktoré spôsobia veľkú konečnú chybu vo výpočte. Príklad takého výpočtu sme naprogramovali podľa algoritmu nájdeného v prezentácií (viď https://keithbriggs.info/documents/xr-kent-talk-pp.pdf) od tvorcu xr knižnice. Teda algoritmus vyzerá nasledovne: $x_0 = 0.9$

$x_{k+1} = 3.999*x_k*(1-x_k)$, vypočítame x_{53}

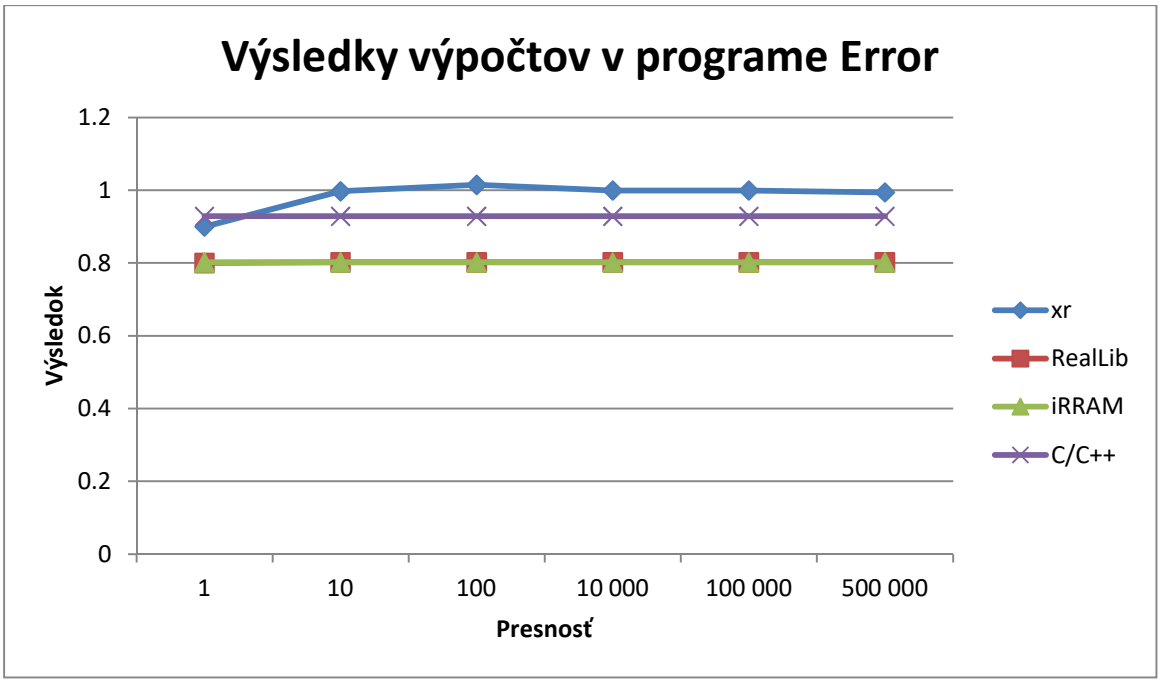
V nasledujúcej tabuľke a v grafe sú uvedené časy výpočtov (v sekundách) s danou presnosťou v jednotlivých knižniciach.

knižnica/ presnosť	1	10	100	10^4	10^5	$5*10^5$	10^6	$4*10^6$	$8*10^6$	10^7	$2*10^7$
xr	0.7541	0.7732	0.7850	0.7499	0.7857	1.097	1.543	5.299	10.64	13.97	30.04
RealLib	0.000482	0.000436	0.000823	0.2156	4.422	48.33					
iRRAM	0.000129	0.000266	0.000298	0.01035	0.1564	1.466	2.625	13.21	29.35	42.19	



V nasledujúcej tabuľke a v grafe sú uvedené začiatočné cifry konečného výsledku s danou presnosťou v jednotlivých knižniciach.

knižnica/ presnosť	1	10	100	10^4	10^5	$5*10^5$
xr	0.9	0.997005	1.014628	0.999419	0.999419	0.99419
RealLib	0.8	0.801919	0.801919	0.801919	0.801919	0.801919
iRRAM	0.801919	0.801919	0.801919	0.801919	0.801919	0.801919



V tomto špecifickom prípade bola najpomalšou knižnicou práve RealLib, ktorej výpočet pri presnosti pol milióna trval vyše 40 sekúnd. Podobnú dobu trval výpočet v iRRAM až pri presnosti 10 miliónov. Najrýchlejší výpočet bol pri použitý knižnice xr, kde pri presnosti 10 miliónov výpočet trval 13 sekúnd.

Ako bolo predpokladané výpočet v klasickom C/C++ sa bude líšiť od využitia knižníc. Prekvapivo výsledky sa líšili aj medzi knižnicami. Výsledok pri použitý knižnice xr sa líšil aj medzi použitými presnosťami, hoci sa postupne ustaľoval. Rovnaké začiatočne cifry výsledkov vypisuje program s použitím knižnice iRRAM aj RealLib.

2.1.5 **Test – Sinus**

Ďalším testom bol program, ktorý využíval funkciu sínus. Tento program sa zaoberá malými číslami a to konkrétne, či $\sin\frac{\pi}{4} - \frac{\sqrt{2}}{2+e^{-1000\ *\ln 10}}$ (čo je ekvivalentné s: $\sin\frac{\pi}{4} - \frac{\sqrt{2}}{2+10^{-10}}$) je rovné s nulou. Najprv program vypíše na konzolu výsledok príkladu s vopred danou presnosťou, potom overí, či je výsledok príkladu rovný nule a potom znova vypíše výsledok príkladu s danou presnosťou. Skrze chýbajúcej implementácie funkcie sínus v knižnici xr sme tento program vytvorili len s použitým knižníc RealLib a iRAAM.

Pri použitý RealLib pri presnosti 10, program vypísal miesto čísla iba: ‘probable zero‘, následne vypísal hodnotu 1 – teda, že číslo nie je rovné číslu nula a potom číslo bolo vypísané s desiatimi ciframi. Programu a knižnici to trvalo približne 4 sekúnd. Od požadovanej presnosti 4990 program už pri prvom výpise vypísalo namiesto ‘probable zero‘ už vypočítané číslo.

V knižnici iRRAM pri presnosti 10 miesto konkrétneho čísla v oboch prípadoch vypísalo cifru 0, ale program zistil, že sa dané číslo nerovná nule. Celý program zbehol v čase 0.3 sekundy. Až pri výpise 9967 cifier program začal vypisovať konkrétne cifry výsledku, dovtedy vypísalo iba nulu oddelenú viacerými medzerami.

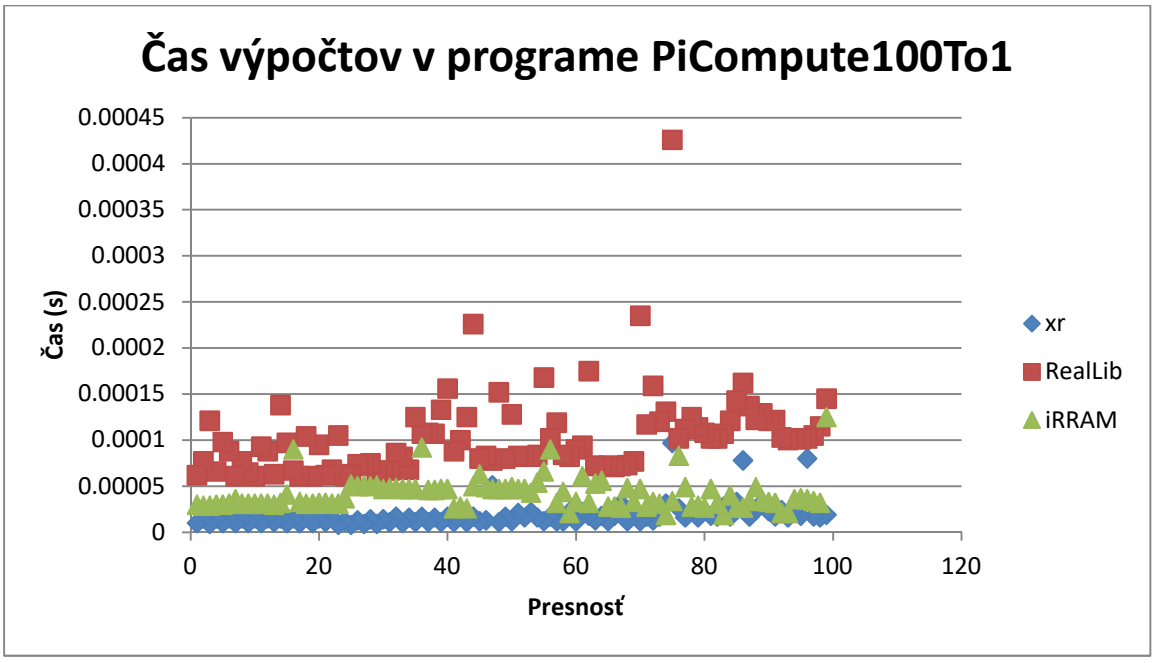
2.1.6 Testy - spustenie výpočtu a pamätanie medzivýsledkov

Ďalšou nami skúmanou oblasťou bolo zistenie, kedy v programe začína výpočet a či pri rôznych presnostiach si program interne v type čísla ukladá presnosť a číslo s doposiaľ počítanou presnosťou. Pre tieto testy využijeme vytvorené programy Error a PiCompute.

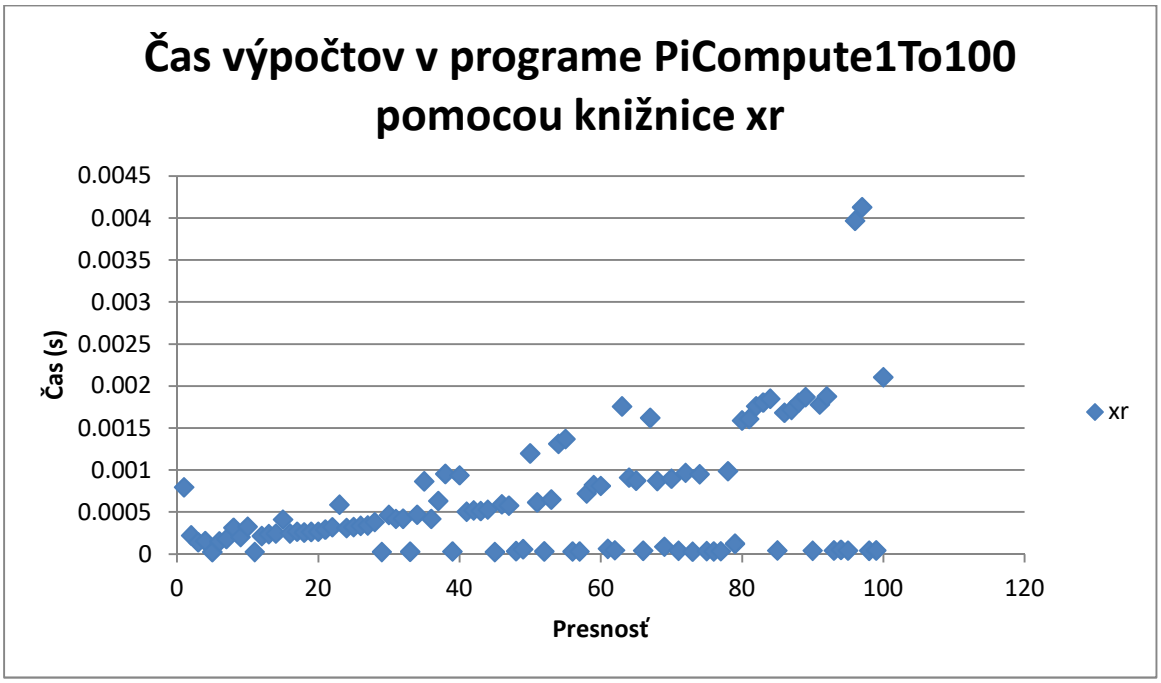
Najprv sme modifikovali program error, tak že sa na konci programu sa výsledok nevypíše. Tento program sme nazvali errorNoOut. Vo všetkých knižniciach prebehol tento program veľmi rýchlo bez vplyvu nastavenej presnosti. Usudzujeme teda, že hlavný výpočet s danou presnosťou začína až pri požiadavke na získanie konečného počtu cifier premennej.

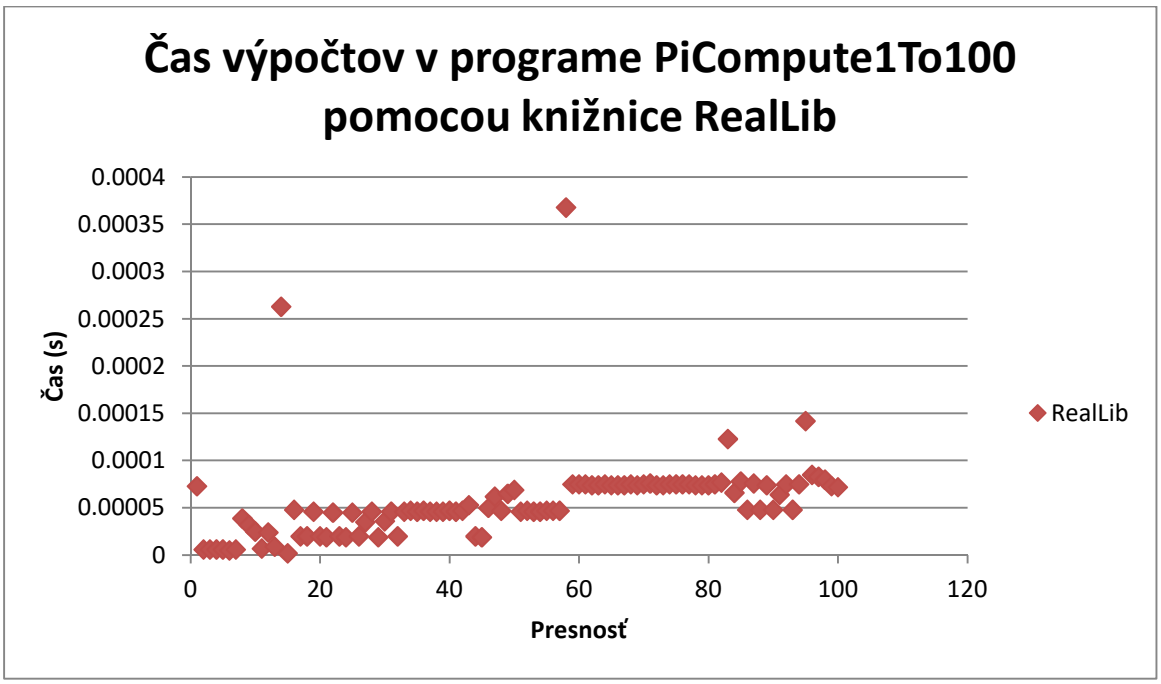
Ďalej bol upravený program piCompute tak, aby program postupne vypísal výsledne vypočítané pí od 1 po 100 cifier vrátané potrebného času na vypísanie daného čísla. V programoch s využitím knižníc iRRAM a RealLib program vypisuje aj použitú presnosť vo výpočte. Týmto spôsobom môžeme zistiť, kedy vo výpočte sa zvyšuje presnosť a či si program zapamätáva doposiaľ vypočítané číslo.

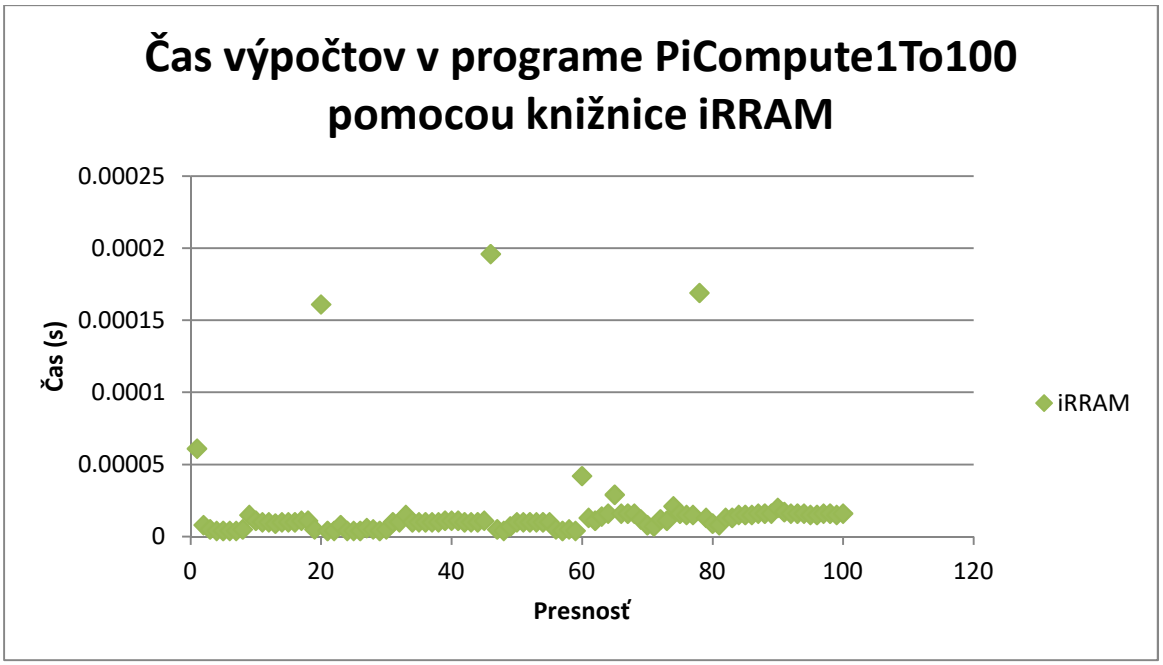
Program piCompute100To1 vypočíta a následne vypíše postupne prvých 100 cifier pí, potom prvých 99 cifier pí atď. Vo všetkých troch knižniciach trval výpočet a výpis prvých 100 cifier najdlhšie a to konkrétne s xr – 0.013173s, s RealLib – 0.000835s a s iRRAM 0.000100s. Potom už výpis ostatných cifier prebehol veľmi rýchlo v priemere s xr 0.000018 s, s RealLib 0.000104 s a s použitým iRRAM 0.000040 s. Z toho vyplýva, že doposiaľ vypočítanú hodnotu a aj presnosť premennej si knižnica interne pamätá. V nasledujúcom grafe sú uvedené časy výpisov s presnosťou, okrem prvého výpisu so sto ciframi.



Program piCompute1To100 vypočíta a následne vypíše postupne prvú cifru pí, potom prvá dve cifry pí atď. Vo všetkých knižniciach pri určitej dosiahnutej presnosti výpis s danou presnosťou trval dlhšie než zvyčajne. V týchto bodoch v knižniciach iRRAM aj RealLib sa zvýšila aj použitá presnosť pri výpočtu. V prípade knižnice xr nie je celkom jasné, kedy sa interne v programe zvýšila presnosť. Pomocou knižnice RealLib sa presnosť zvyšovala pri požadovaných 14 a 58 číslic čísla pí. V prípade iRRAM sa presnosť zvýšovala pri požadovaných 20, 46 a 78 cifier pí. V nasledujúcich grafoch sú uvedené časy výpisov pri vyžadovanej presnosti s použitým jednotlivých knižníc.







2.2 Tvorba Macra

Jedným z problémov, pri tvorbe tých istých programov vo viacerých knižniciach je nutnosť poznať názov a signatúru jednotlivých funkcií. Väčším problémom je písanie obdobného kódu pre každú knižnicu zvlášť a pri možnej zmene správania programu je nutné prerobiť jednotlivé zdrojové súbory. Týmto spôsobom nielenže je časovo náročnejšie pre programátora udržiavať programy, ale zvyšuje sa šanca na vytvorenie prípadnej chyby. Oveľa lepším riešením, ktoré sme implementovali, je tvorba spoločného makra, teda určitého rozhrania, ktoré dokáže pokryť potrebnú funkcionalitu daných knižníc. Potom môžeme efektívne písať kód a až pri kompilácií sa rozhodnúť, ktorú z knižníc využijeme.

Dané makro s názvom MACRO-REAL.h a programy využívajúce dané makro sa nachádzajú v priečinku Programy. Vytvorené makro ponúka nasledujúce funkcie:

MyReal initVar(int b) – inicializácia premennej
void init() – inicializácia knižnice
int exec(int(*f)) – spustenie výpočtu v funkcií f
void finish() – finalizácia knižnice

MyReal myAdd(MyReal a, MyReal b) – sčítanie
MyReal mySub(MyReal a, MyReal b) – odčítanie
MyReal myMul(MyReal a, MyReal b) – násobenie
MyReal myDiv(MyReal a, MyReal b) – delenie
MyReal mySqrt(MyReal a) – druhá odmocnina

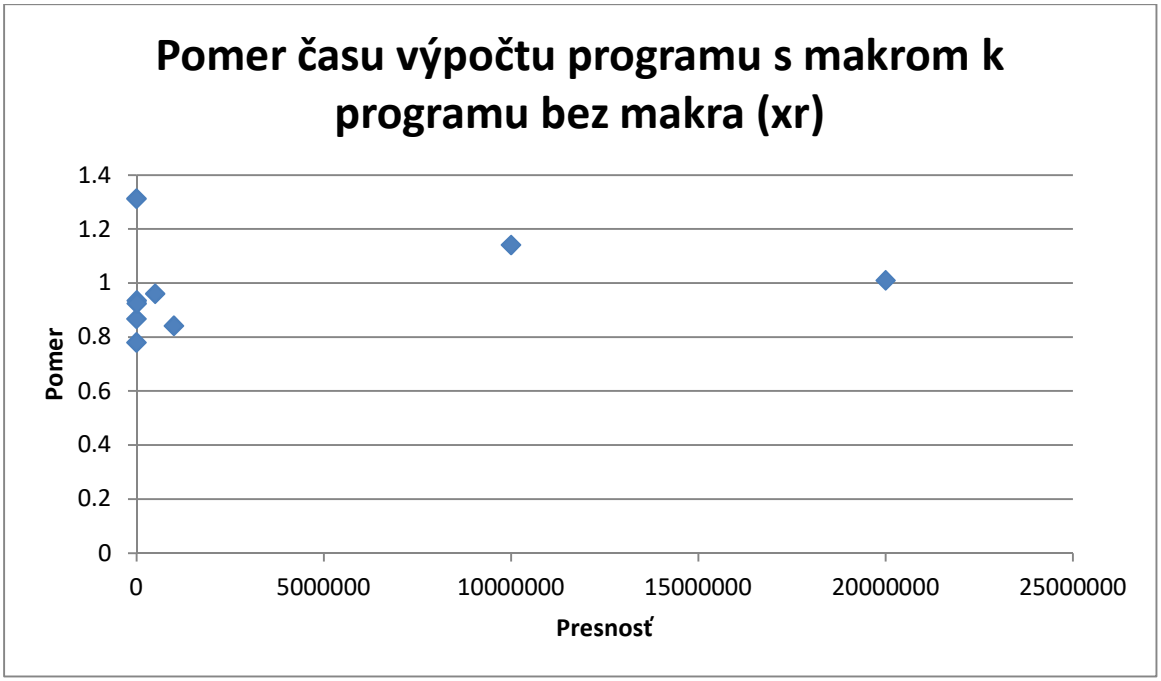
void myPrint(MyReal a, long long precision) – výpis čísla na štandardnú konzolu s danou presnosťou

Pre jednoduchšiu prácu s makrom sme vytvorili šablónu – pattern program spolu s makefile. Pre následnú kompiláciu vo všetkých knižniciach stačí v hlavnom priečinku napísať príkaz *make* (kompilácia vo všetkých knižniciach) alebo *make xr* / *realLib* / *iRRAM* (kompilácia v jednotlivých knižniciach).

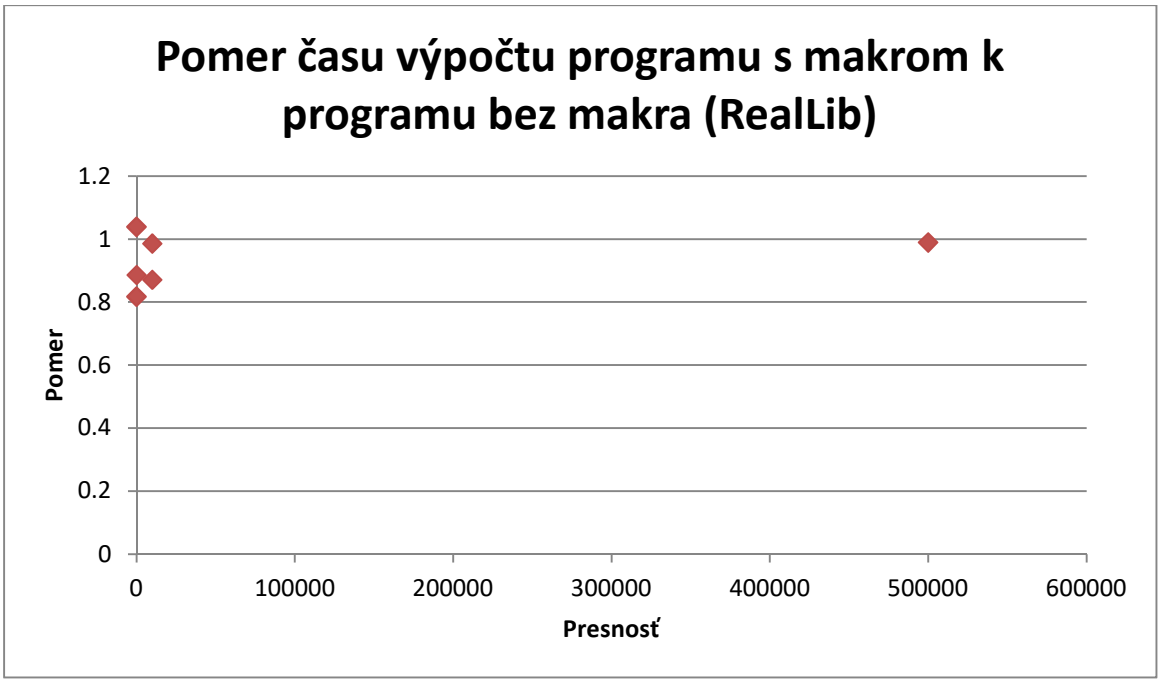
2.2.1 Test – example pomocou makra

Pomocou makra a šablóny bol znova vytvorený program example (viď 2.1.1). Mohli sme tak porovnať rozdiel v rýchlosti medzi použitím makra a priamočiarym použitím knižníc. V nasledujúcich tabuľkách sú uvedené časy výpisov s použitím a bez použitia makra, za jednotlivými tabuľkami nasleduje graf pomeru programu s makrom k programu bez makra.

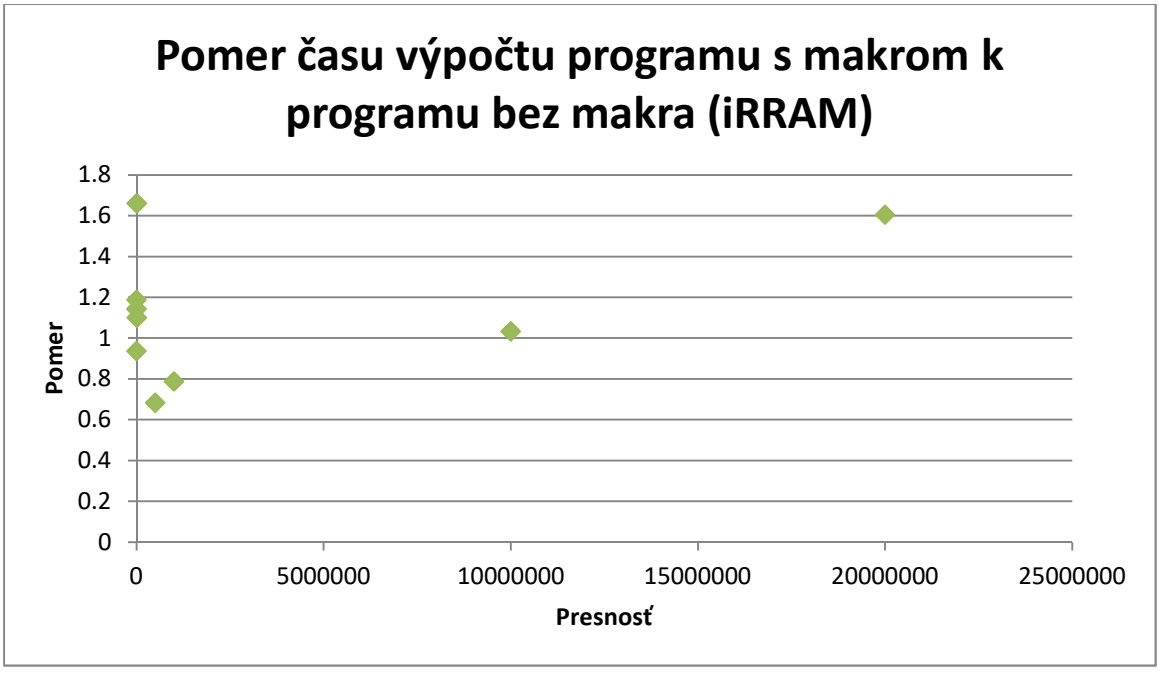
knižnica/presnosť	1	10	100	10^4	10^5	$5*10^5$	10^6	10^7	$2*10^7$
xr (makro)	0.000147	0.000144	0.000156	0.003064	0.0696	0.4528	1.133	18.99	37.13
xr	0.000112	0.000166	0.000200	0.003276	0.07527	0.4715	1.346	16.64	36.76



knižnica/presnosť	1	10	100	10^4	10^5	$5*10^5$
RealLib (makro)	0.000133	0.000112	0.000218	0.03746	1.147	27.20
RealLib	0.000128	0.000137	0.000246	0.0380	1.317	27.49



knižnica/presnosť	1	10	100	10^4	10^5	$5*10^5$	10^6	10^7	$2*10^7$
iRRAM (makro)	0.000278	0.000223	0.000239	0.001578	0.02626	0.1565	0.3940	5.143	18.08
iRRAM	0.000234	0.000238	0.000209	0.000950	0.02385	0.229	0.5	4.977	11.26



Pri použití makra alebo s priamym použitím knižníc sa rýchlosť programu líšila ± o 20 – 60%, v priemere približne o 15% pomalšie. Takýto zvláštny rozdiel mohol byť spôsobený viacerými faktormi priamo nesúvisiacimi s programov využívajúcim makro ako

je napríklad momentálna zaťaženosť procesora. Podstatné je, že rýchlosť programu sa rádovo nespomalila.

2.2.2 Test – rekurzia a makro

V dôsledku knižnice iRRAM a jej začatia výpočtu pomocou funkcie iRRAM_compute, celý výpočet s reálnou presnou aritmetikou by mal byť zahrnutý v funkcií compute. Preto sme sa rozhodli testovať rekurziu funkcie compute. Program – recursion pomocou rekurzie vypočíta hodnotu $\frac{1}{3^{10}}$ postupným delením jednotky číslom 3. Následne po dopočítaní a opätovným zavolaním funkcie compute, v danej funkcií výsledok program vypíše na štandardný konzolový výstup.

Predpokladali sme, že knižnice xr a RealLib budú v tomto prípade fungovať bez problémov, pretože v ich prípade nie je nutné sústreďovať začatie výpočtu do špecifickej funkcie. Tento predpoklad sa naozaj potvrdil, teda stačí iba inicializovať knižnicu. Problémom sa stala knižnica xr, kde sa presnosť pri výpise skrátila na väčšiu polovicu (teda miesto správneho výpisu 0.3333333333e0 program vypísal 0.3333334922e0). Príčina tohto dôvodu nebola nájdená, ale problém pravdepodobne súvisí s funkciou div. Korektnosť respektíve neúplná garancia výpisu bola už spomenutá autorom v dokumentácií knižnice xr. Presná reálna aritmetika s použitím knižnice iRRAM je vykonaná nie len v prvom zavolaní funkcie compute, ale aj v funkciách volaných v funkcií compute, teda aj v rekurzívnych volaniach funkcie compute. Taktiež je možné zavolať vykonávanie funkcie compute viackrát z hlavnej funkcie main.

3 Výsledok a zhrnutie

Naším jeden semestrovým snažením bolo zameranie sa na existujúcu implementáciu presnej reálnej aritmetiky – ERA. Porovnávali sme knižnice písane v programovacom jazyku C a C++. Jedinou porovnávanou knižnicou v jazyku C bola knižnica xr. Táto knižnica hoci obsahuje základné matematické operácie, bez zložitých funkcií (ako sú goniometrické), bola v jednoduchých príkladoch oveľa rýchlejšia než ďalšia knižnica RealLib. V jednom teste, konkrétne v error bola dokonca zo všetkých knižníc najrýchlejšia. Problémom pre knižnicu xr je presný výpis výsledku, ak pri počítaní daného výsledku bolo použité delenie (viď 2.2.2), ale aj pri iných rôznych výpisoch (viď 2.1.4).

Knižnica RealLib na rozdiel od knižnice xr je písaná v jazyku C++ a obsahuje nielen základné matematičke operácie a funkcie, obsahuje tiež dôležité goniometrické funkcie. Je taktiež možné zistiť a nastaviť aktuálnu presnosť použitú vo výpočtoch. Nevýhodou je absencia komplexnejších funkcií (napríklad hyperbolických) a pomalší výpočet výsledku s veľkou presnosťou.

Najmohutnejšou porovnávanou knižnicou je knižnica iRRAM. Táto knižnica písaná v jazyku C++ je zo všetkých knižníc najrozsiahlejšia, pretože okrem reálnych čísel obsahuje aj mnohé iné číselné typy (napríklad komplexné číslo). S použitým tejto knižnice vo všetkých programoch okrem jedného (viď 2.1.4) prebehol výpočet s veľkou presnosťou rádovo rýchlejšie než s využitým inej knižnice. Veľkým obmedzeným s využitým tejto knižnice je jej špecifické sústredenie výpočtu do funkcie.

Vytvorili sme makro – určité rozhranie s podporou základných matematických operácií. Pomocou tohto makra vieme abstrahovať od konkrétnych knižníc a teda môžeme mať iba jeden zdrojový kód a následne rozhodnúť, ktorú knižnicu využijeme na kompiláciu. Toto riešenie zjednodušuje tvorbu a údržbu kódu programu.

Celkovo hodnotíme našu prácu ako úspešne zvládnutú. Vytvorili sme rôzne testy pre tri rôzne implementácie presnej reálnej aritmetiky. Pomocou testov a dokumentácie sme mohli určiť ich výhody a nevýhody. Pre jednoduchšiu prácu s rôznymi knižnicami sme vytvorili makro. Ďalším možným pokračovaním projektu je rozšírenie funkcionality makra a to konkrétne pridanie porovnávania alebo goniometrických funkcií. Projekt ďalej môže smerovať ku konkrétnemu výpočtovo náročnému programu využívajúci presnú reálnu aritmetiku.

BIBLIOGRAFICKÉ ODKAZY

Briggs, K. (12. 05 2013). *Keith Briggs:: xrc (exact reals in C)*. Cit. 27. 12 2023. Dostupné na Internete: Keith Briggs: https://keithbriggs.info/xrc.html

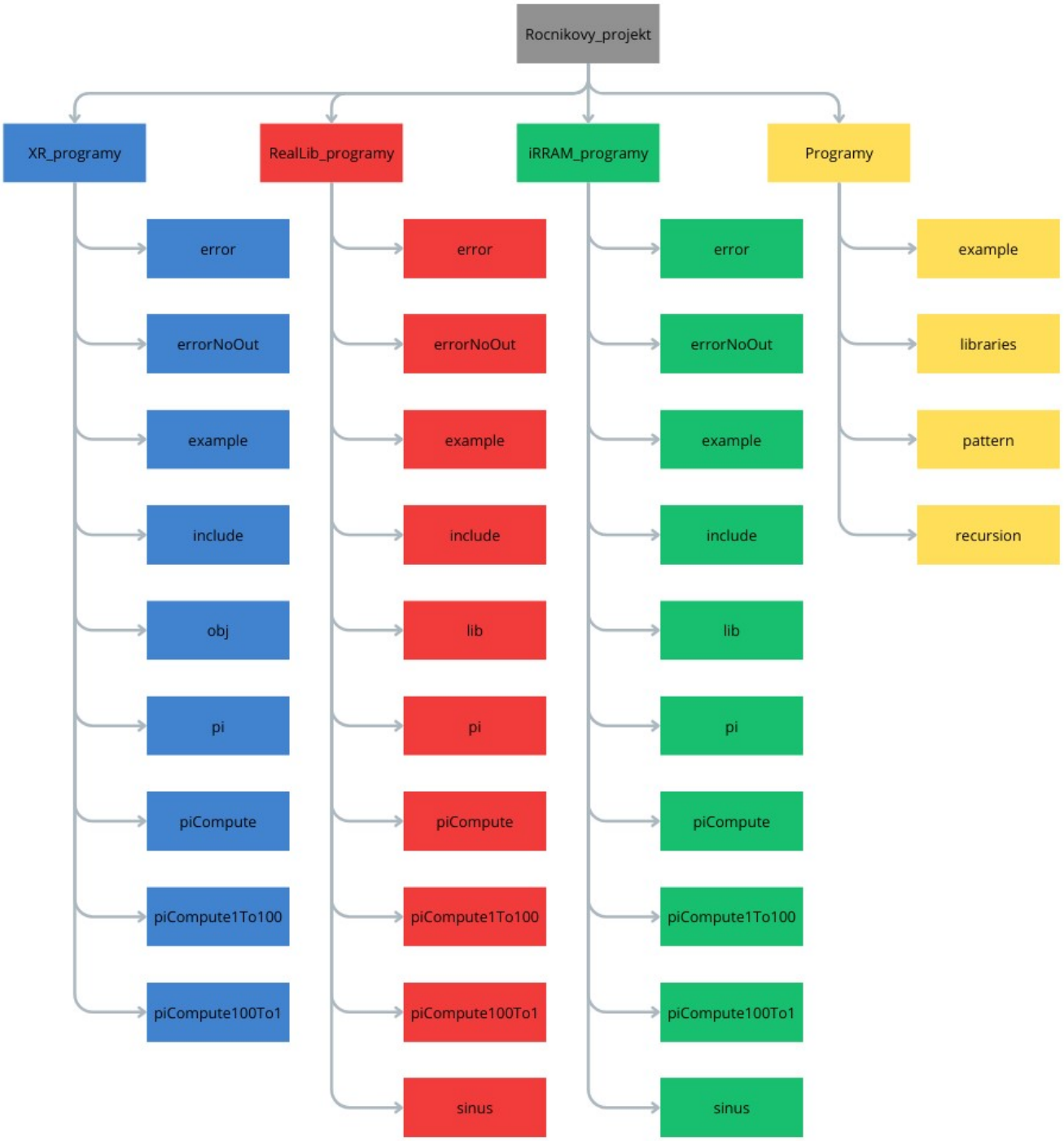
Keith, B. (12. 05 2013). *Keith Briggs:: xrc manpage*. Cit. 27. 12 2023. Dostupné na Internete: Keith Briggs: https://keithbriggs.info/xr.1.html

Lambov, B. (5. 4 2015). *GitHub - blambov/RealLib: Library for exact real number computations*. Cit. 27. 12 2023. Dostupné na Internete: GitHub - blambov/RealLib:: https://github.com/blambov/RealLib

Müller, N. (1. 5 2015). *GitHub - norbert-muller/iRRAM: Exact real arithmetic in C++*. Cit. 27. 12 2023. Dostupné na Internete: GitHub - norbert-muller/iRRAM:: https://github.com/norbert-mueller/iRRAM/tree/master

PRÍLOHY

Príloha 1 – Štruktúra projektu



Obr. 1 Základná priečinková štruktúra projektu