

Přesnost a rychlost výpočtu

Jan Faigl

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Lecture 10

B3B36PRG – Programming in C

Přehled témat

- Část 1 – Přesnost výpočtu
Přesnost výpočtů a numerická stability
- Část 2 – Rychlost výpočtu (programu)
Maticové násobení
Rychlost výpočtu
Paralelní výpočet
- Část 3 – Implementace domácích úkolů
Diskutovaná témata

Přesnost výpočtů

Part I

Část 1 – Přesnost výpočtu

Přesnost výpočtů

Přesnost výpočtu - Příklad součtu dvou čísel

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     double a = 1e+10;
6     double b = 1e-10;
7
8     printf("a : %24.121f\n", a);
9     printf("b : %24.121f\n", b);
10    printf("a+b: %24.121f\n", a + b);
11
12    return 0;
13 }
14
15 clang sum.c && ./a.out
16 a : 100000000000.000000000000000000
17 b : 0.000000000000000000000000000000
18 a+b: 100000000000.000000000000000000
```

lec10/sum.c

Přesnost výpočtů

Zdroje a typy chyby

- Chyby matematického modelu - matematická aproximace fyzikální situace.
- Chyby vstupních dat.
- Chyby numerické metody.
- Chyby zaokrouhlovací.

- Absolutní chyba aproximace
 $E(x) = \hat{x} - x$, \hat{x} přesná hodnota, x aproximace.
- Relativní chyba $RE(x) = \frac{\hat{x} - x}{x}$.

Přesnost výpočtů

Přesnost výpočtu - Příklad dělení dvou čísel

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     const int number = 100;
6     double dV = 0.0;
7     float fV = 0.0f;
8
9     for (int i = 0; i < number; ++i) {
10        dV += 1.0 / 10.0;
11        fV += 1.0 / 10.0;
12    }
13
14    printf("double value: %lf ", dV);
15    printf("float value: %lf ", fV);
16
17    return 0;
18 }
19
20 clang division.c && ./a.out
21 double value: 10.000000 float value: 10.000002
```

lec10/division.c

Přesnost výpočtů

Podmíněnost numerických úloh

- Podmíněnost úlohy $C_p = \frac{\text{relativní chyba výstupních údajů}}{\text{relativní chyba vstupních údajů}}$.
- Dobře podmíněná úloha $C_p \approx 1$.
- Výpočet je dobře podmíněný, je-li málo citlivý na poruchy ve vstupních datech.
- Numericky stabilní výpočet - vliv zaokrouhlovacích chyb na výsledek je malý.
- Výpočet je stabilní, je-li dobře podmíněný a numericky stabilní.

Přesnost výpočtů

Přesnost výpočtu - strojová přesnost

- Strojová přesnost ϵ_m - nejmenší desetinné číslo, které přičtením k 1.0 dává výsledek různý od 1, pro $|v| < \epsilon_m$, platí

$$v + 1.0 == 1.0.$$

Symbol `==` odpovídá porovnání dvou hodnot (test na ekvivalenci).

- Zaokrouhlovací chyba - nejmeně ϵ_m .
- Přesnost výpočtu - aditivní chyba roste s počtem operací v řádu $\sqrt{N} \cdot \epsilon_m$.
 - Často se však kumuluje preferabilně v jedno směru v řádu $N \cdot \epsilon_m$.

Přesnost výpočtů

Možnosti zvýšení přesnosti

- Reprezentace racionálních čísel - podíl dvou celočíselných hodnot, např. *Homogenní souřadnice*.
- „Libovolná přesnost“ - speciální knihovny, např. `gmp` až do výše volné paměti.
<https://gmplib.org/manual/index>
 - Souřadnice x,y - 7511164176768 346868669952 3739567104 ~ 2008,57; 92,76.

Přesnost výpočtu

Součin dvou velkých čísel knihovnou gmp - 1/2

- V HW04B je uveden příklad (995663 - 995669)⁸ jako prvočíselný rozklad čísla
932865073719992059629773513614789388266580305083920591925740371392254317064584855785088915745761.
<https://cv.fel.cvut.cz/wiki/courses/b3b36prg/hw/hw02>
- Použijme knihovnu gmp pro mocninu a součin dvou čísel, #include<gmp.h>.
 - Typ celých čísel mpz_t, pomocné funkce mpz_init_set_str(), mpz_init(), gmp_printf() a mpz_clears() a operace mpz_pow_ui() a mpz_mul().

Mocnina unsigned integer a násobení - multiplication.

- Knihovna nemusí být součástí operačního systému, proto může být nutně specifikovat cestu k hlavičkovému souboru a vlastní knihovně (-lgmp).
 - Můžeme zadat cestu ručně při kompilaci (nebo do Makefile).
 - Alternativně můžeme použít nástroj pkg-config (nebo pkgconf).

<https://www.freedesktop.org/wiki/Software/pkg-config/> <http://pkgconf.org/>

- Argumenty pro překlad (CFLAGS).


```
$ pkgconf --cflags gmp
-I/usr/local/include
```
- Argumenty pro linkování (LDFLAGS).


```
$ pkgconf --libs gmp
-L/usr/local/lib -lgmp
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 11 / 36

Přesnost výpočtu

Součin dvou velkých čísel knihovnou gmp - 2/2

```
1 #include <stdio.h> 26 gmp_printf("Xz x Xz\n", n1, n2);
2 #include <stdlib.h> 27
3 28 mpz_mul(result, n1, n2);
4 #include <gmp.h> 29 gmp_printf("Xz\n", result);
5 30
6 const char* resultSrc = 31 printf("Result from HW04\nXz\n", resultSrc);
7 "932865073719992059629773513614789388266580305083" 32
8 "920591925740371392254317064584855785088915745761"; 33
9 34 mpz_clears(n1, n2, result, NULL);
10 35 return ret;
11 int main(int argc, char *argv[]) 36 }
12 {
13 int ret = EXIT_SUCCESS; $ ./demo-gmp-mpz
14 mpz_t n1, n2, result; n1: 995663
15 mpz_init_set_str(n1, "995663", 10); n2: 995669
16 mpz_init_set_str(n2, "995669", 10); 995663*8 x 995669*8
17 mpz_init(result); 96582612429460787982699926255695296863400309121 x
18 gmp_printf("n1: Xz\n", n1); 96587268686826115153703708226203156648104775841
19 gmp_printf("n2: Xz\n", n2); 93286507371999205962977351361478938826658030508392059
1925740371392254317064584855785088915745761
21 gmp_printf("Xz Xz x Xz Xz\n", n1, 8, n2, 8); Result from HW04
22 93286507371999205962977351361478938826658030508392059
23 mpz_pow_ui(n1, n1, 8); 1925740371392254317064584855785088915745761
24 mpz_pow_ui(n2, n2, 8);
25 lecl0/gmp/demo-gmp-mpz.c
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 12 / 36

Přesnost výpočtu

Racionální čísla knihovny gmp - 1/3

- „Libovolné přesnosti“ reprezentace, např. souřadnic v rovině jako výsledek operací výpočetní geometrie, můžeme realizovat podílem dvou („libovolné velkých“) celých čísel.
 - Souřadnice x,y - 7511164176768 346868669952 3739567104 ~ 2008,57; 92,76.
- Knihovna gmp k tomuto účelu poskytuje typ mpq_t, kromě typu necelého čísla mpf_t, který využijeme pro převod mpq_t na celé číslo typu double.


```
49 double mpq2d(const mpq_t *op)
50 {
51 double ret;
52 mpf_t v;
53 mpf_init(v);
54 mpf_set_q(v, *op);
55 ret = mpf_get_d(v);
56 mpf_clear(v);
57 return ret;
58 }
```

lecl0/gmp/demo-gmp-mpq.c

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 13 / 36

Přesnost výpočtu

Racionální čísla knihovny gmp - 2/3

```
1 #include <stdio.h> 27 mpq_t x, y;
2 #include <stdlib.h> 28 mpq_inits(x, y, NULL);
3 #include <gmp.h> 29 mpq_set_ui(x, x1, den1);
4 double mpq2d(const mpq_t *op); 30 mpq_set_ui(y, y1, den1);
5 31
6 int main(int argc, char *argv[]) 32 mpq_canonicalize(x);
7 { 33 mpq_canonicalize(y);
8 int ret = EXIT_SUCCESS;
9 34 mpf_t xmpf, ympf;
10 35 mpf_inits(xmpf, ympf, NULL);
11 36 mpf_set_q(xmpf, x);
12 unsigned long x1 = 75111641767681; 37 mpf_set_q(ympf, y);
13 unsigned long y1 = 3468686699521; 38
14 unsigned long den1 = 37395671041; 39
15 const unsigned int digits = 21; 40 gmp_printf("mpq x,y (canonical form): X04 X04\n", x, y);
16 41 gmp_printf("mpf x,y (to Xd decimal digits): %.*f %.*f\n",
17 double xd = 1. * x1; 42 digits, digits, xmpf, ympf);
18 double yd = 1. * y1; 43 gmp_printf("mpq x,y (double .46): %.*46f %.*46f\n",
19 double den1 = 1. * den1; 44 mpq2d(x), mpq2d(y));
20 45
21 printf("unsigned long: X1u X1u\n", x1, y1, den1); 46 mpf_clears(x, y, NULL);
22 printf("double: X.01f X.01f X.01f\n", xd, yd, den1); 47 mpf_clears(xmpf, ympf, NULL);
23 48 return ret;
24 printf("double x,y (.2): %.*21f %.*21f\n", xd/den1, yd/den1);
25 printf("double x,y (.46): %.*46f %.*46f\n", xd/den1, yd/den1);
26 lecl0/gmp/demo-gmp-mpq.c
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 14 / 36

Přesnost výpočtu

Racionální čísla knihovny gmp - 3/3

- Souřadnice x,y - 7511164176768 346868669952 3739567104 ~ 2008,57; 92,76.


```
$ make
clang -c -I/usr/local/include -g demo-gmp-mpq.c -o demo-gmp-mpq.o
clang demo-gmp-mpq.o -I/usr/local/lib -lgmp -o demo-gmp-mpq
clang -c -I/usr/local/include -g demo-gmp-mpz.c -o demo-gmp-mpz.o
clang demo-gmp-mpz.o -I/usr/local/lib -lgmp -o demo-gmp-mpz
```

`lecl0/gmp/Makefile`

```
$ make info
-I/usr/local/include -g
-L/usr/local/lib -lgmp
```

`$ gmake`

```
clang -c -I/usr/local/include -g demo-gmp-mpq.c -o demo-gmp-mpq.o
clang demo-gmp-mpq.o -I/usr/local/lib -lgmp -o demo-gmp-mpq
clang -c -I/usr/local/include -g demo-gmp-mpz.c -o demo-gmp-mpz.o
clang demo-gmp-mpz.o -I/usr/local/lib -lgmp -o demo-gmp-mpz
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 15 / 36

Přesnost výpočtu

Makefile s pkg-config a gmp

```
1 CFLAGS+=$(shell pkg-config --cflags gmp) 17 $(DEMO_MPQ): $(DEMO_MPQ).o
2 LDFLAGS+=$(shell pkg-config --libs gmp) 18 $(CC) $(C) $(LDFLAGS) -o $@
3 19
4 CFLAGS+=-g 20 $(DEMO_MPZ): $(DEMO_MPZ).o
5 DEMO_MPQ=demo-gmp-mpq 21 $(CC) $(C) $(LDFLAGS) -o $@
6 DEMO_MPZ=demo-gmp-mpz 22
7 TARGETS+=$(DEMO_MPQ) $(DEMO_MPZ) 23 %.o : %.c
8 24 $(CC) -c $(CPPFLAGS) $(CFLAGS) $(C) -o $@
9 25
10 bin: $(TARGETS) 26 clean:
11 27 $(RM) $(DEMO_MPQ) $(DEMO_MPZ) *.o
12 lecl0/gmp/Makefile
13 info:
14 @echo $(CFLAGS)
15 @echo $(LDFLAGS)
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 16 / 36

Přesnost výpočtu

Reprezentace necelých čísel – IEEE 754

- Reálné číslo x se zobrazuje ve tvaru $x = (-1)^s \cdot \text{mantisa} \cdot 2^{\text{exponent} - \text{bias}}$. **Základ 2.** IEEE 754, ISO/IEC/IEEE 60559:2011
- Mantisa je normalizována na první jedničku vlevo (v soustavě o dvojkovém základu).
- float – 32 bitů (4 bajty): s – 1 bit znaménko (+ nebo -), exponent – 8 bitů, tj. 256 možností. mantisa – 23 bitů \approx 16,7 milionu možností.

- double – 64 bitů (8 bajtů).
 - s – 1 bit znaménko (+ nebo -).
 - exponent – 11 bitů, tj. 2048 možností.
 - mantisa – 52 bitů \approx 4,5 bilióny možností (4 503 599 627 370 495).
- bias umožňuje reprezentovat exponent vždy jako kladné číslo.
 - Lze zvolit, např. $\text{bias} = 2^{e-1} - 1$, kde e je počet bitů exponentu.

<http://www.root.cz/clanky/norma-ieee-754-a-pribuzni-formaty-plovouci-radove-tecky>

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 17 / 36

Přesnost výpočtu

Příklad reprezentace float hodnot dle IEEE 754

- Chyba reprezentace -256.75 vs -256.74.
- Infinity (0x7f800000), -Infinity (0xff800000), a NaN (0x7fffffff).

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 18 / 36

Přesnost výpočtu

Příklady reprezentace hodnot typu float

- Reprezentace čísla 85,125 (float)
 - 85 odpovídá $1010101_{(2)}$.
 - 0,125 odpovídá 001
 - $0,125 \cdot 2^{-1} = 0,25$ | 0
 - $0,125 \cdot 2^{-2} = 0,50$ | 0
 - $0,125 \cdot 2^{-3} = 1,00$ | 1
 - 85,125 odpovídá $1010101,001_{(2)} = 1,010101001_{(2)} \cdot 2^6$.
 - Bias pro float je 127.
 - Exponent je $127 + 6 = 133$.
 - 133 odpovídá $10000101_{(2)}$.
 - Normalizovaná mantisa je $010101001_{(2)}$, kterou doplníme nulami na 23 bitů (zprava, je to desetinné číslo).
 - 0-1000 0101-0101 0100 1000 0000 0000 0000.
 - 01000010 10101010 01000000 00000000.
 - V šestnáctkové soustavě to je 0x3d 0xxx 0xxx 0xxx, tedy 0x42aa4000.
- Reprezentace čísla 0,1 (float)
 - 0,1 má periodický rozvoj
 - $0,1 \cdot 2 = 0,2$ | 0
 - $0,2 \cdot 2 = 0,4$ | 0
 - $0,4 \cdot 2 = 0,8$ | 0
 - $0,8 \cdot 2 = 1,6$ | 1
 - $0,6 \cdot 2 = 1,2$ | 1
 - $0,2 \cdot 2 = 0,4$ | 0
 - Opakuje se 0011, 23-bitů tak reprezentuje menší hodnotu.
 - $0,1_{(10)} \sim 0,0001 1001 1001 1001 1001 1001 100_{(2)} = 1,0011001100110011001100_{(2)} \cdot 2^{-4}$.
 - Exponent je $127 - 4 = 123$ odpovídá $01111011_{(2)}$.
 - Normalizovaná mantisa $\approx 1001 1100 1100 1100 1100 1100$.
 - 0-0111 1011-100 1100 1100 1100 1100 1100.
 - 00111011 1001100 11001100 11001100.
 - V šestnáctkové soustavě to je 0x3d 0xxx 0xxx 0xxx, tedy 0x3dcccc.
 - Prakticky je 0,1 převedeno na o něco větší číslo 0x3dcccc, protože absolutní chyba je menší.

`lecl0/floats.c`

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 19 / 36

Přesnost výpočtů

Sečítání mnoha malých necelých čísel - 1/2

- Na příkladu součtu dvou velmi odlišných čísel (např. $1 \times 10^{10} + 1 \times 10^{-10}$) dochází z důvodu omezené reprezentace mantisy k zaokrouhlovací chybě.
- V případě naivní implementace součtu velkého počtu (např. 1×10^{-20}) může dojít vlivem zaokrouhlování k významné chybě.

lec10/addition.c

```
// small value to be sum
float v = 1e-20f; //float literal
// 1073741824 is 2^30 values (1e9)
const size_t power = 30;
size_t n = 1l<<power;

// multiplication factor for print
const double k = 1e11;

float sum_naive(size_t n, float v) {
    float r = 0;
    for (size_t i = 0; i < n; ++i) {
        r += v[i];
    }
    return r;
}

float sum_alter(size_t n, float v, size_t power) {
    float r = 0;
    const size_t order = power - 1;
    size_t k = 2;
    for (size_t l = 1; l < order; ++l, k *= 2) {
        for (size_t i = 0; i < n; i += k) {
            v[i] = v[i] + v[i+k/2];
        }
    }
    k /= 2;
    for (size_t i = 0; i < n; i += k) {
        r += v[i];
    }
    return r;
}

double sum1 = v*n * k;
double sum2 = sum_naive(n, values) * k;
float sum3 = sum_alter(n, values, power) * k;

Přímé násobení - výsledek 1.073 741 789 925 364 287 228 1.
Naivní součet - výsledek 0.022 737 367 544 323 205 947 9.
Sečítání po dvojicích - výsledek 1.073 741 793 632 507 324 218 8.
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 20 / 36

Přesnost výpočtů

Sečítání mnoha malých necelých čísel - 2/2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float* init_values(size_t n, float v);
5 float sum_naive(size_t n, float v);
6 float sum_alter(size_t n, float v, size_t power);
7
8 int main(void)
9 {
10     float v = 1e-20f; // small value to be sum
11     const size_t power = 30; // try 3 vs. 30
12     size_t n = 1l<<power; // 1073741824 is 2^30 values
13     const double k = 1e11;
14
15     float vvalues = init_values(n, v);
16
17     double sum1 = v * n * k;
18     double sum2 = sum_naive(n, values) * k;
19     float sum3 = sum_alter(n, values, power) * k;
20
21     printf("Sum of %lu numbers of the value %.22f\n", n, v);
22     printf("Sum1 (multiplication): %.22f\n", sum1);
23     printf("Sum2 (naive) : %.22f\n", sum2);
24     printf("Sum3 (alter) : %.22f\n", sum3);
25     free(values);
26     return EXIT_SUCCESS;
27 }
```

\$ clang addition.c -o addition && ./addition
Sum of 1073741824 numbers of the value 0.0000000000000000000100
Sum1 (multiplication): 1.0737417899253642872281
Sum2 (naive) : 0.0227373675443232059479
Sum3 (alter) : 1.0737417936325073242188

\$ calc "1e-20 * 2^30 * 1e11"
1.073741824

lec10/addition.c

- Implementujte s využitím knihovny `gmp`.

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 21 / 36

Maticové násobení Rychlost výpočtu Paralelní výpočet

Part II

Část 2 – Rychlost výpočtu (programu)

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 22 / 36

Maticové násobení - Naivně

```
1 void simple_multiply(const int n, const double *a, const double *b, double *c)
2 {
3     for (int i = 0; i < n; ++i) {
4         for (int j = 0; j < n; ++j) {
5             double prod = 0;
6             for (int k = 0; k < n; ++k) {
7                 prod += a[i * n + k] * b[k * n + j];
8             }
9             c[i * n + j] = prod;
10        }
11    }
12 }
```

- Pro přehlednost předpokládáme kompatibilní rozměry matic a správně alokované.

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 24 / 36

Maticové násobení - Naivně s transpozicí

```
1 void simple_multiply_trans(const int n, const double *a, const double *b, double *c)
2 {
3     double *bT = create_matrix(n); // allocate memory for transposed matrix
4     for (int i = 0; i < n; ++i) {
5         bT[i*n + i] = b[i*n + i];
6         for (int j = i + 1; j < n; ++j) {
7             bT[i*n + j] = b[j*n + i];
8             bT[j*n + i] = b[i*n + j];
9         }
10    }
11    for (int i = 0; i < n; ++i) {
12        for (int j = 0; j < n; ++j) {
13            double tmp = 0;
14            for (int k = 0; k < n; ++k) {
15                tmp += a[i*n + k] * bT[j*n + k];
16            }
17            c[i*n + j] = tmp;
18        }
19    }
20    free(bT);
21 }
```

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 25 / 36

Maticové násobení

Porovnání rychlosti násobení matic

Y-osa: Výpočetní čas [ms] (0 to 14000)
X-osa: Velikost matice (200 to 1400)

Legenda: Naivní (zelená), Naivní s transpozicí (modrá)

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 26 / 36

Architektura procesoru a způsob výpočtu

- Příklad násobení matic a násobení transponované matice ukazuje, že kromě instrukcí má zásadní vliv **organizace dat a přístup do paměti**.
- V moderních procesorech hraje **cache** zásadní roli společně s řetězením instrukcí, tzv. **pipelining**, a využitím specifických instrukcí.
- SIMD - Single Instruction Multiple Data
- Proniknutí do detailů fungování cache a řetězení instrukcí je náplní předmětu **Architektura počítačů (BOB35APO)**, kde máte možnost se seznámit s přicházející architekturou RISC V.

<https://cw.felk.cvut.cz/wiki/courses/b35apo/>

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 28 / 36

Optimalizace kódu

- Kromě optimalizace výsledného spustitelného kódu při překladu, je možné optimalizovat kódu za běhu nebo již existujících binární (přeložené) soubory.
 - BOLT** - Binary Optimization and Layout Tool, zrychlení o až 20%–50%
<https://arxiv.org/abs/1807.06735>
<https://dl.acm.org/doi/10.5555/3314872.3314876>
- Využití speciálních instrukcí v základních funkcích může výpočty (programy) výrazně urychlit, zejména pokud se funkce používají masivně.
 - AVX2 a EVEX instrukce (ze sady SSE4.2) ve funkcích porovnání řetězců `str{n}casecmp()` – až o 38% méně potřebného času.
03/2022 - <https://www.phoronix.com/news/Glibc-strcasecmp-AVX2-EVEX>
- V obou případech (a obecně) je vhodné rozumět principu a využít instrukce Assembleru.*

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 29 / 36

Compiler Explorer

<https://godbolt.org/z/K9r1eWqd>

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 30 / 36

Maticové násobení Rychlost výpočtu Paralelní výpočet

Compiler Explorer – Analýza optimalizovaného kódu

- Vliv optimalizace -O2 na výsledný kód, který obsahuje nedefinované chování, přetečení celého čísla. Příloha 3. přednášky.

<https://godbolt.org/z/G3GEz4vbw>

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 31 / 36

Maticové násobení Rychlost výpočtu Paralelní výpočet

Příklad použití OpenMP - Maticové násobení 1/2

- Open Multi-Processing (OpenMP) - aplikační programové rozhraní (API) multiplatformních výpočtů se sdílenou pamětí. <http://www.openmp.org>
- Direktivou preprocesoru můžeme instruovat kompilátor k vytvoření kódu paralelního výpočtu, např. paralelizace přes vnější proměnnou `i`.

```

1 void multiply(int n, int a[n][n], int b[n][n], int c[n][n])
2 {
3     int i;
4     #pragma omp parallel private(i)
5     #pragma omp for schedule (dynamic, 1)
6     for (i = 0; i < n; ++i) {
7         for (int j = 0; j < n; ++j) {
8             c[i][j] = 0;
9             for (int k = 0; k < n; ++k) {
10                c[i][j] += a[i][k] * b[k][j];
11            }
12        }
13    }
14 }

```

lec10/demo-omp-matrix.c
Pro přehlednost uvažujeme čtvercové matice stejných rozměrů.

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 33 / 36

Maticové násobení Rychlost výpočtu Paralelní výpočet

Příklad použití OpenMP - Maticové násobení 2/2

- Příklad násobení matic 1000 x 1000 s využitím OpenMP na iCore5 (2 jádra s HT ~ 4x výpočetní jednotky).
- Násobení matic 5000 x 5000 (Ryzen 9 5950X).

```

1 gcc -std=c99 -O2 -o demo-omp demo-omp-matrix.c -fopenmp
2 ./demo-omp 1000
3 Size of matrices 1000 x 1000 naive
4 multiplication with O(n^3)
5 c1 == c2: 1
6 Multiplication single core 9.33 sec
7 Multiplication multi-core 4.73 sec
8
9 OMP_NUM_THREADS=2 ./demo-omp 1000
10 Size of matrices 1000 x 1000 naive
11 multiplication with O(n^3)
12 c1 == c2: 1
13 Multiplication single core 9.48 sec
14 Multiplication multi-core 6.23 sec

```

- i7-6700K:
 - 1x vlákno 0.80s;
 - 2x vlákna 0.39s;
 - 4x vlákna 0.24s.

```

1 OMP_NUM_THREADS=16 ./demo-omp 5000
2 Size of matrices 5000 x 5000 naive
3 multiplication with O(n^3)
4 Multiplication single core 256.00 sec
5 Multiplication multi-core 18.05 sec

```

lec10/demo-omp-matrix.c

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 34 / 36

Diskutovaná témata

Shrnutí přednášky

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 35 / 36

Diskutovaná témata

- Numerická přesnost.
- Knihovna `gmp`.
- Maticové násobení a organizace paměti.
- Rychlost výpočtu a architektura procesoru.
- Paralelní výpočty OpenMP.

Jan Faigl, 2024 B3B36PRG – Lecture 10: Přesnost a rychlost výpočtu 36 / 36