C2115
Praktický úvod do superpočítání

IX. lekce

Petr Kulhánek, Tomáš Bouchal

kulhanek@chemi.muni.cz

Národní centrum pro výzkum biomolekul, Přírodovědecká fakulta,
Masarykova univerzita, Kotlářská 2, CZ-61137 Brno

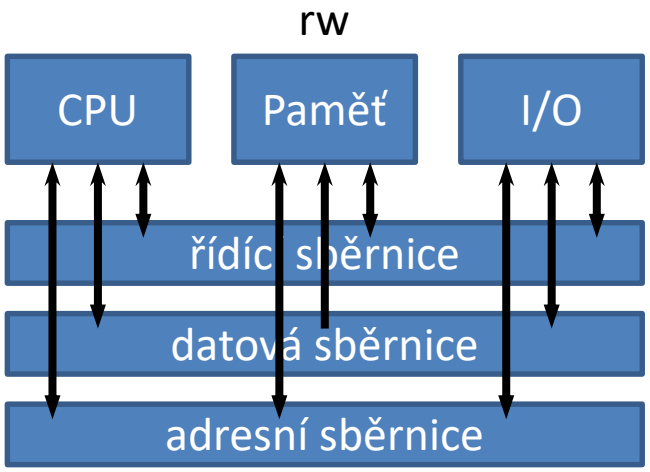
Obsah

- **Architektura počítače**
  CPU, paměť, grafická systém, disky, síť, periferie
- **Reprezentace číselných hodnot v číslicové technice**
  celá čísla, reálná čísla
- **Od problému k výsledku**
  algoritmus, zdrojové kódy, překlad, spouštění programu, programovací jazyky

Architektura počítače

Přehled

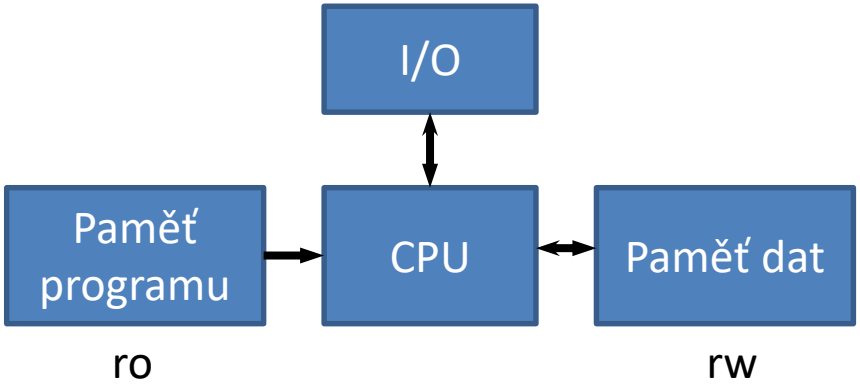
1945 von Neumannova architektura



- program může sebe modifikovat
- program a data nelze načítat současně

John von Neumann, původem maďarský matematik, působící ve spojených státech

1944 Hardwardská architektura

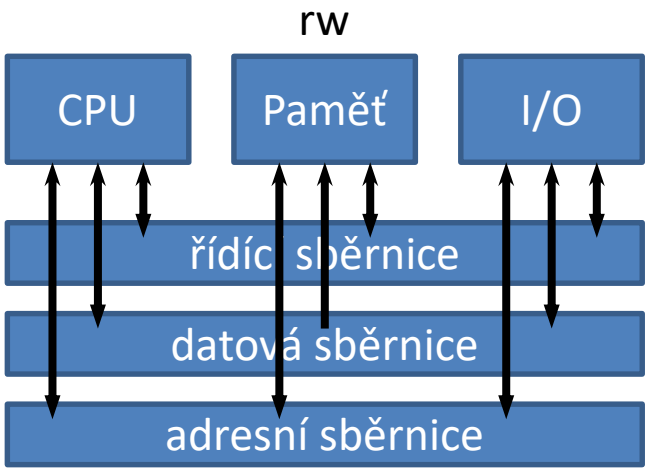


- program se nemůže modifikovat
- program a data se mohou načítat současně

Harvard Mark I - počítač složeny z relé, 24 bitové instrukce

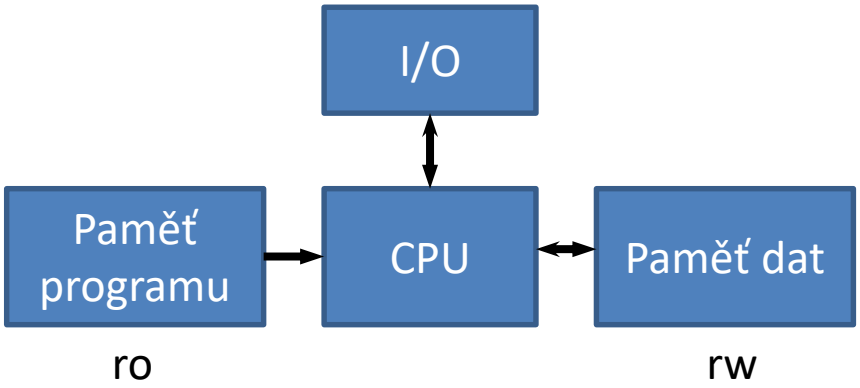
Přehled

1945 von Neumannova architektura



- program může sebe modifikovat
- program a data nelze načítat současně

1944 Hardwardská architektura



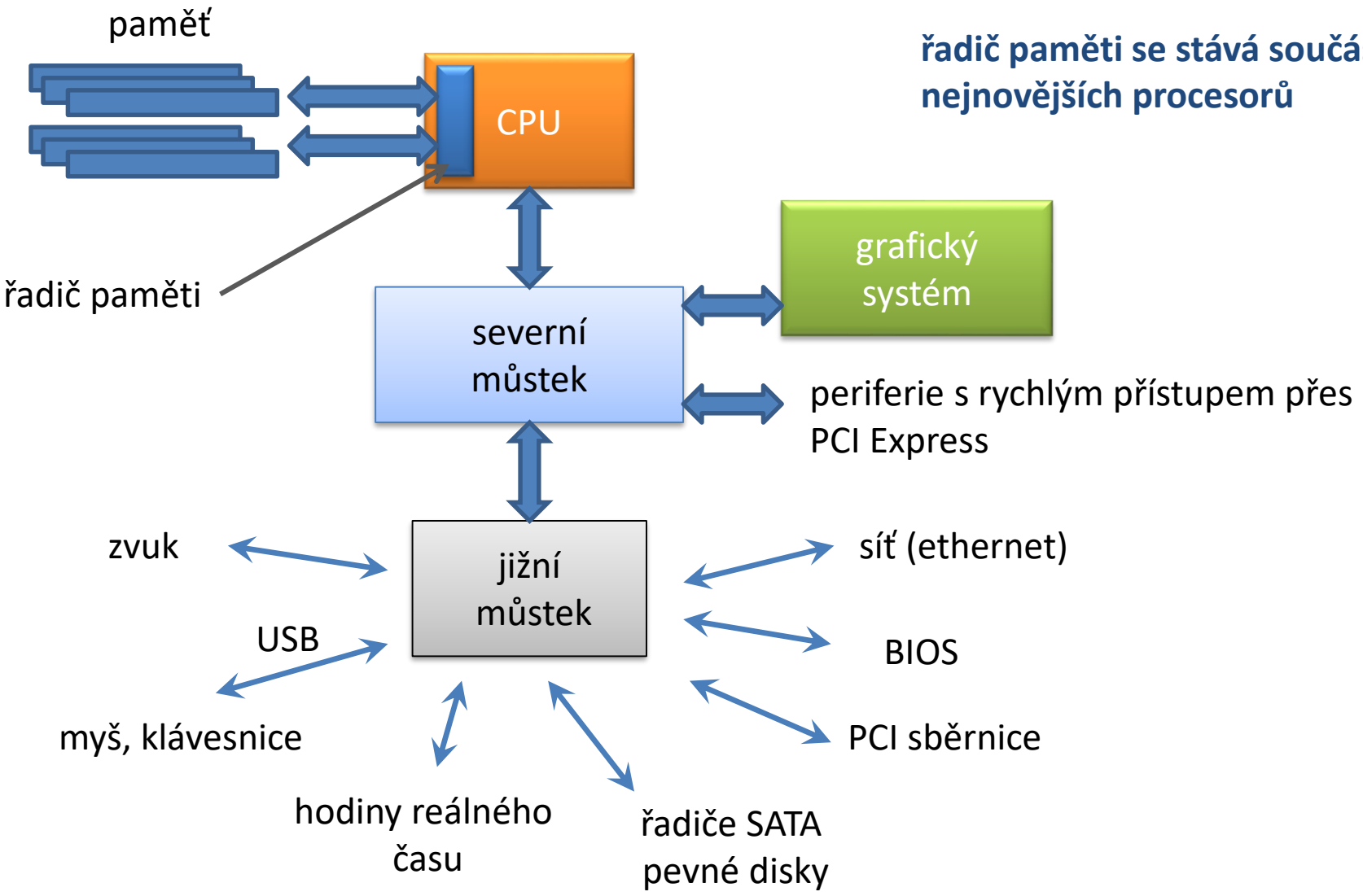
- program se nemůže modifikovat
- program a data se mohou načítat současně

v dnešních počítačích se kombinují obě architektury

John von Neumann, původem maďarský matematik, působící ve spojených státech

Harvard Mark I - počítač složeny z relé, 24 bitové instrukce

Typické schéma počítače

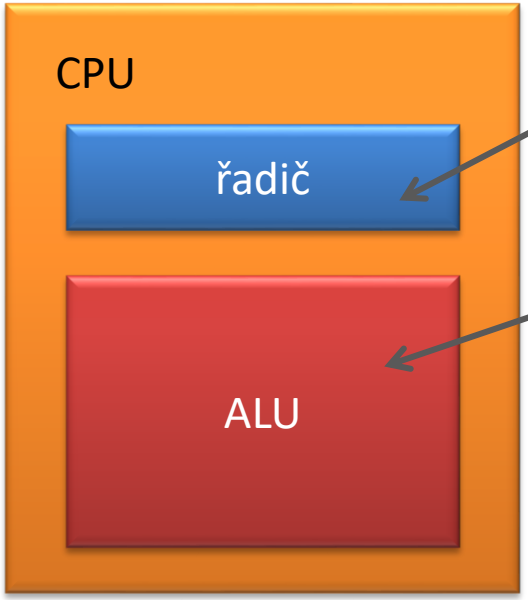


řadič paměti se stává součástí nejnovějších procesorů

CPU

Procesor též **CPU** (anglicky **Central Processing Unit**) je základní součástí počítače; jde o velmi složitý sekvenční obvod, který **vykonává strojový kód** uložený v operační paměti počítače. Strojový kód je složen z jednotlivých strojových instrukcí počítačových programů nahraných do operační paměti.

www.wikipedia.org



načítá strojové instrukce a data a připravuje jejich zpracování v ALU

ALU (arithmetic and logic unit), vykonává aritmetické operace, vyhodnocuje podmínky

sekvenční zpracovávání strojových instrukcí je řízeno vnitřním hodinovým taktem

Reprezentace číselných hodnot

Celá čísla

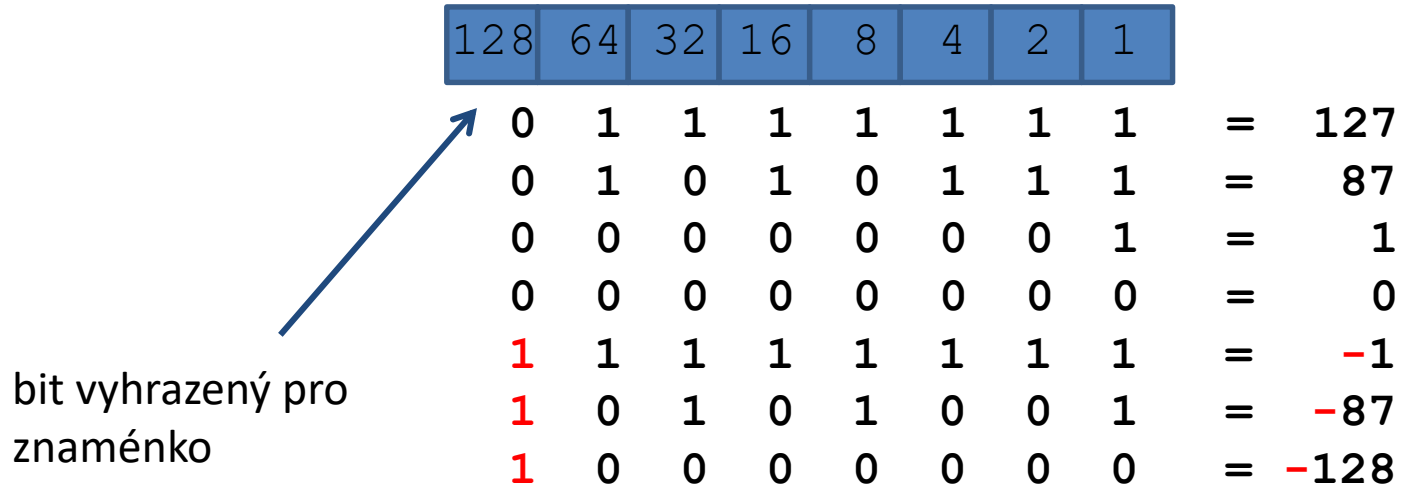
Nejmenší jednotkou informace v číslicové technice je jeden **bit**. Z bitů se skládají slova. Nejmenším slovem je **byte**, který obsahuje 8 bitů.

Jeden byte může popsat celá čísla s rozsahem od 0 do 255.

128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	1

= 87

Lze vyjádřit i celá čísla se znaménkem. V tomto případě je jeden bit vyhrazen pro znaménko, zbývající bity pro číslo. Existuje několik možností implementace. Intel architektura využívá **dvojkový doplněk**, který vede k rozsahu od -128 do 127.



Celá čísla, II

Celé čísla s větším dynamickým rozsahem lze vyjádřit pomocí větších slov typicky složených ze čtyř bajtů (32 bitové slovo) nebo osmi bajtů (64 bitové slovo).

32 bitové celé číslo bez znaménka:	0 až 4.294.967.295
32 bitové celé číslo se znaménkem:	−2.147.483.648 až 2.147.483.647
64 bitové celé číslo bez znaménka:	0 až 18.446.744.073.709.551.615
64 bitové celé číslo se znaménkem:	−9.223.372.036.854.775.808 až 9.223.372.036.854.775.807

Při práci s celými čísly je nutné brát v potaz, že s nimi **nelze vyjádřit libovolně velké číslo** a je nutné se důsledně vyvarovat možnosti **podtečení** nebo **přetečení** hodnoty.

Reálná čísla

Reálná čísla se vyjadřují v následujícím formátu (formát s **pohyblivou čárkou**, angl. **floating point**):

$$X = (-1)^s . (1 + Q) . 2^E$$

exponent

mantisa

$$Q = m_1 \frac{1}{2^1} + m_2 \frac{1}{2^2} + m_3 \frac{1}{2^3} + m_4 \frac{1}{2^4} \ldots$$

m_1, m_2, m_3 jsou bity mantisy

V číslicové technice se reálná čísla nejčastěji vyjadřují ve formátu definovaném standardem **IEEE 754**.

typ	šířka	mantisa	exponent
jednoduchá přesnost (single precision)	32	23	8
dvojnásobná přesnost (double precision)	64	52	11

Reálná čísla, II

typ	rozsah	přesnost
jednoduchá přesnost (single precision)	$\pm 1{,}18\times10^{-38}$ až $\pm 3{,}4\times10^{38}$	přibližně 7 desetinných míst
dvojnásobná přesnost (double precision)	$\pm 2{,}23\times10^{-308}$ až $\pm 1{,}80\times10^{308}$	přibližně 15 desetinných míst

Speciální kombinací hodnoty mantisy a exponentu, lze vyjádřit následující **speciální hodnoty**:

0	kladná nula
-0	záporná nula
NaN	not a number, např. výsledek dělení nulou
+Inf	kladné nekonečno (číslo je příliš velké pro vyjádření)
-Inf	záporné nekonečno (číslo je příliš velké pro vyjádření)

Při práci s reálnými čísly je nutné dbát ohled na šíření **zaokrouhlovacích chyb**, v logických porovnáváních **není vhodné** používat operátory **rovná se** a **nerovná se**, kromě situace porovnávání s nulou.

Cvičení 1

1. Proměnná typu **signed char** (bajt se znaménkem) obsahuje číslo 127. Jakou hodnotu bude mít proměnná pokud ji zvýšíme o jedničku?
2. Proměnná typu **unsigned char** (bajt bez znaménka) obsahuje číslo 88. Jak se číselná hodnota změní, pokud se provede posunutí bitové reprezentace čísla o jednu pozici doprava nebo doleva? Jaký matematický význam má daná operace.
3. Jaký bude výsledek součtu reálných čísel reprezentovaných v dvojnásobné přesnosti a majících hodnotu:

   $0{,}1346978.10^{-12}$

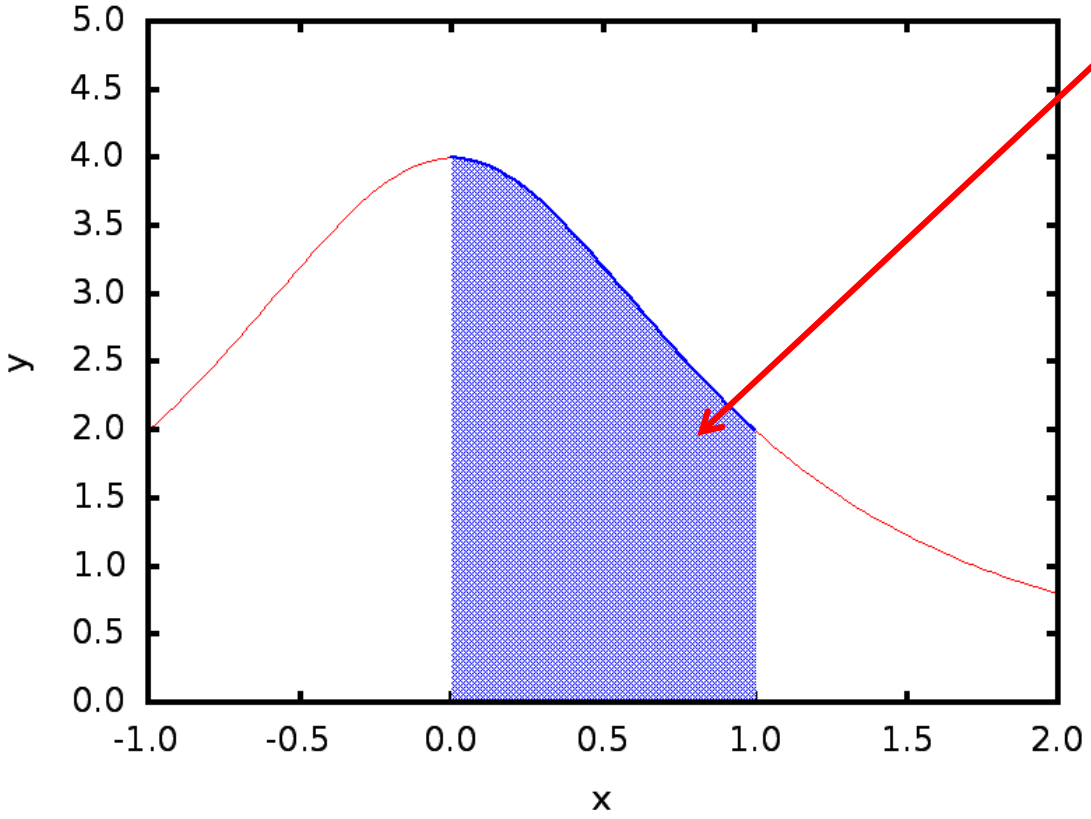
   $1{,}2312657.10^{6}$
4. Co je to big-endian a little-endian? Uveďte architektury, které daný typ endianity používají. Jaký vliv má endianita na přenos binárních dat?

Společné cvičení: převod číselných hodnot z dekadické do dvojkové a hexadecimální soustavy

Numerická integrace

Cvičení LIII.3

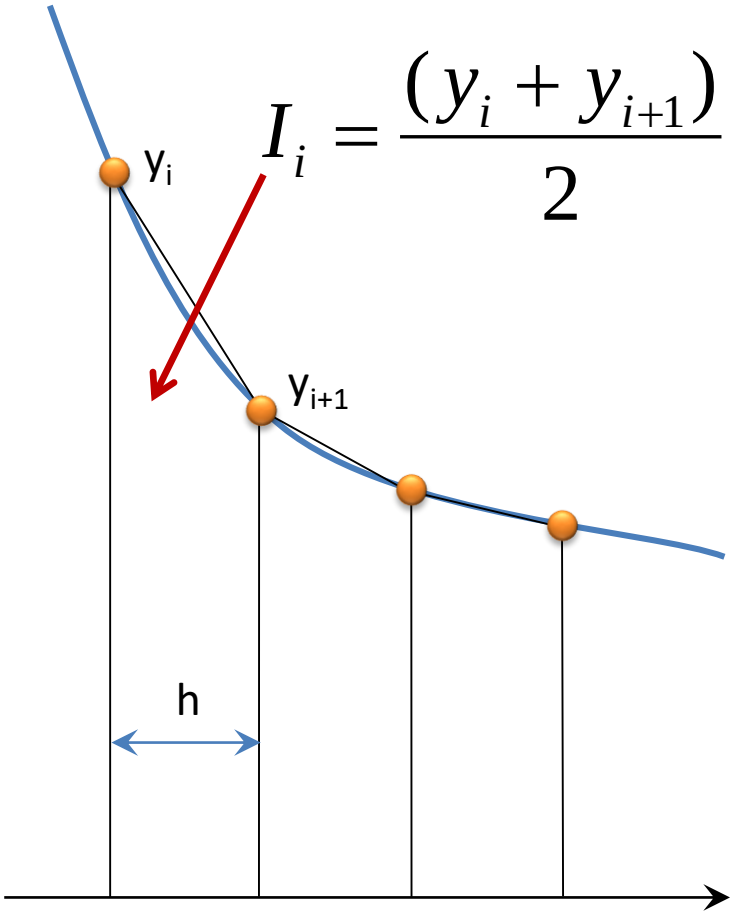
1. Napište program, který vypočte určitý integrál uvedený níže. K integraci použijte lichoběžníkovou metodu.



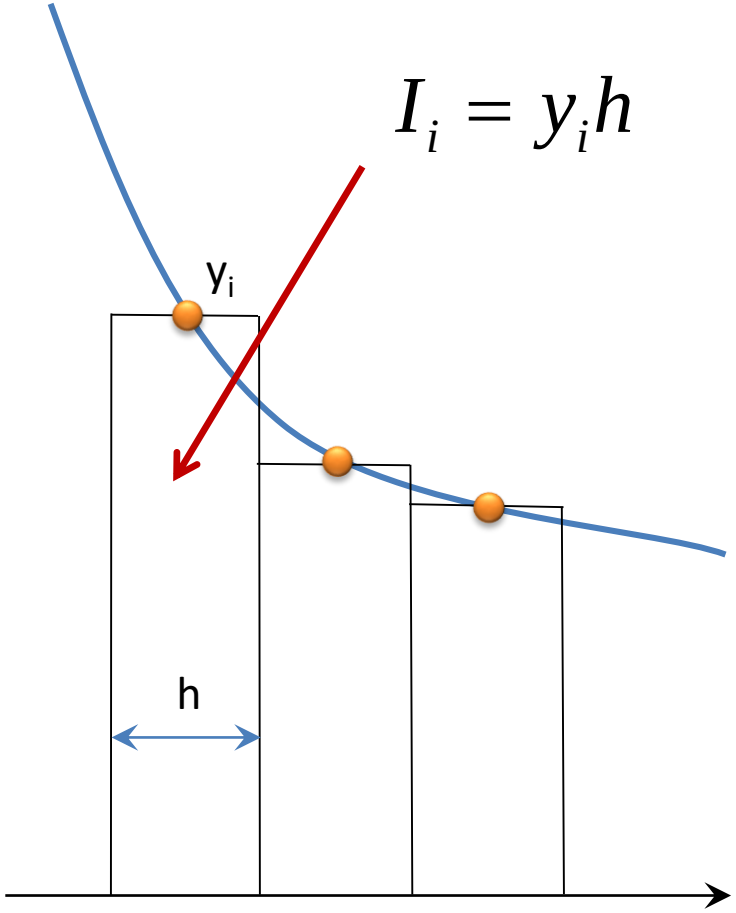
$$I = \int_0^1 \frac{4}{1+x^2} dx$$

určitý integrál je plocha pod křivkou v rozsahu integračních mezí

Lichoběžníková vs obdélníková metoda



lichoběžníková metoda



obdélníková metoda

Sekvenční implementace

obdélníková metoda

```
program integral

  implicit none
  integer(8)                :: i
  integer(8)                :: n
  double precision          :: h,v,y,x
  !-----------------------------------------------
  n = 2000000000
  h = 1.0d0/n
  v = 0.0d0

  do i=1,n
    x = (i-0.5d0)*h
    y = 4.0d0/(1.0d0+x**2)
    v = v + y*h
  end do

  write(*,*) 'integral = ',v

end program integral
```

Cvičení 2

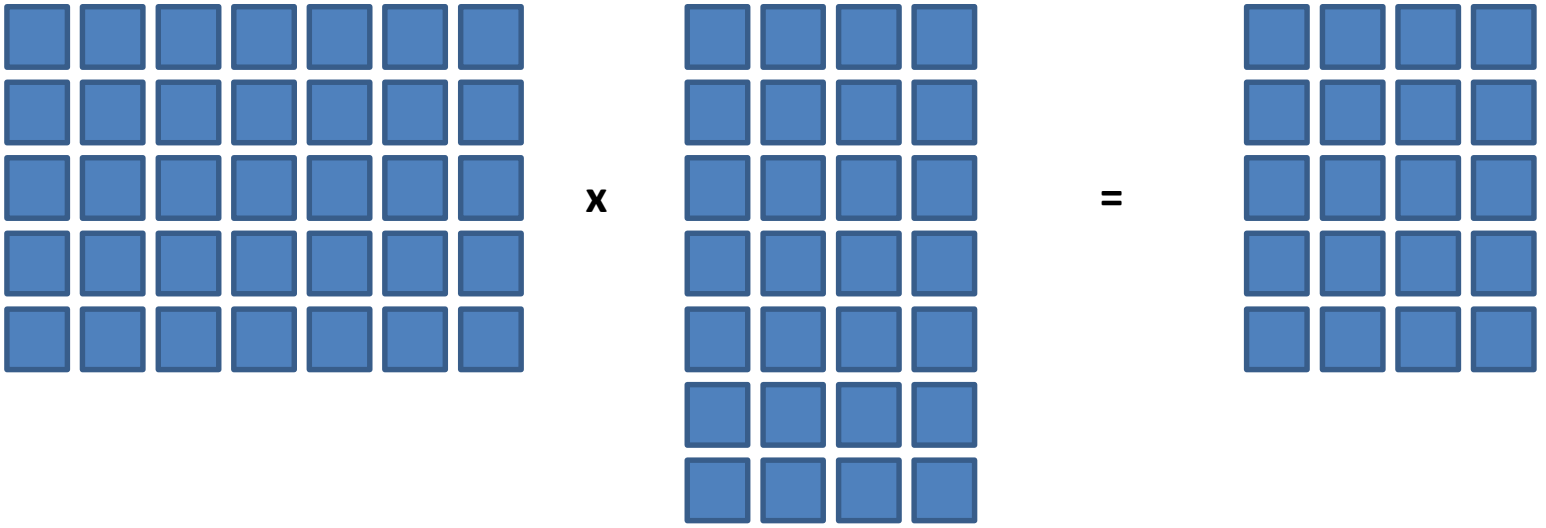
1. Zkompilujte program **integral.f90** z adresáře **/home/kulhanek/Data/C2115/Lesson12/integral/single** s optimalizací **-O3**
2. Určete dobu běhu aplikace potřebnou pro integraci funkce. K měření doby použijte program **/usr/bin/time**.
3. Jaký vliv má na přesnost výpočtu hodnota n (velikost h)?

Násobení matic

Obsah

- **Násobení matic**
  implementace, komplexita, výpočetní výkon, cvičení
- **Vysvětlení získaných výsledků**
  architektura počítače a její úzká hrdla
- **Použití optimalizovaných knihoven**
  BLAS, LAPACK, LINPACK, porovnání výsledků, cvičení

Násobení matic

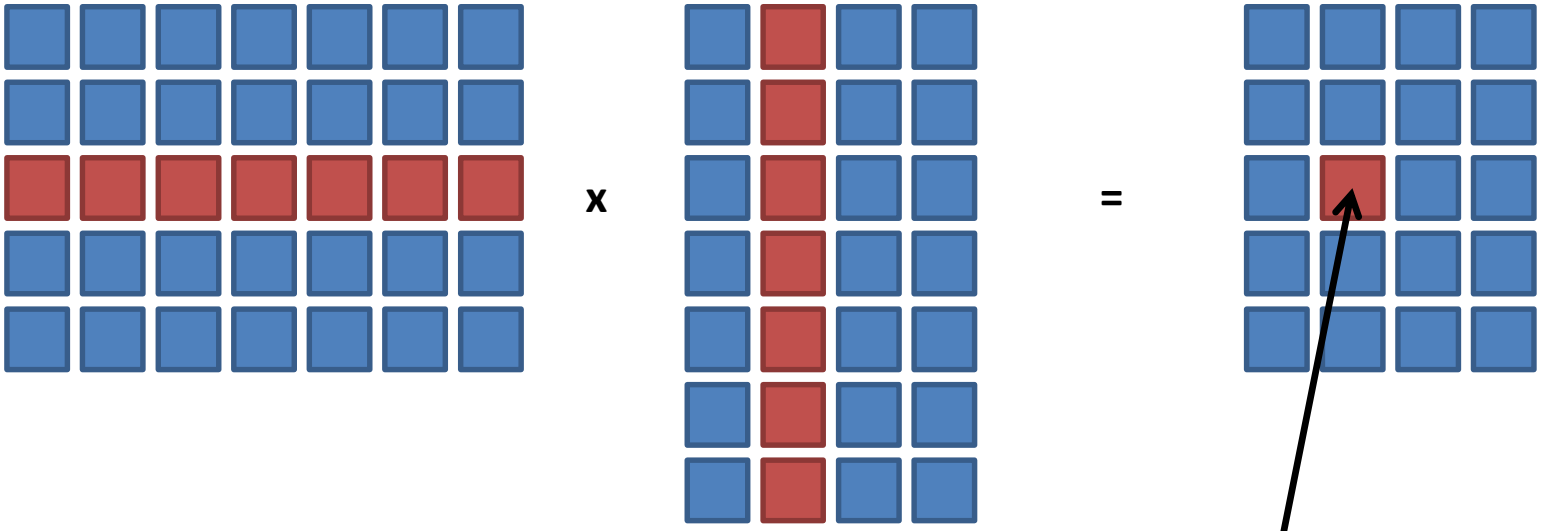


A(n,m) **B**(m,k) **C**(n,k)

Využití:

- hledání vlastních čísel a vektorů čtvercových matic (kvantová chemie)
- řešení soustavy lineárních rovnic (QSAR, QSPR)
- transformace (posunutí, rotace, škálování - zobrazení a grafika)

Násobení matic



A(n,m) **B**(m,k) **C**(n,k)

$$C_{ij} = \sum_{l=1}^{m} A_{il} B_{lj}$$

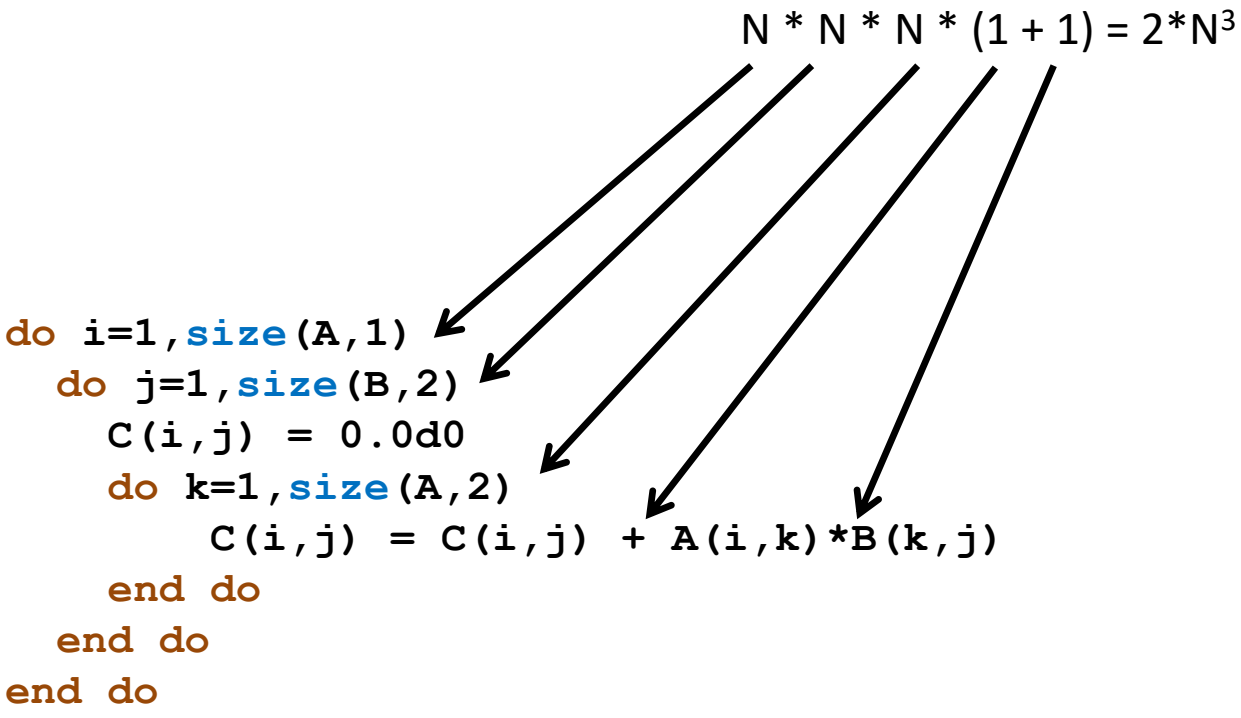
prvek výsledné matice **C** je skalárním součinem vektorů tvořených řádkem *i* matice **A** a sloupcem *j* matice **B**

Násobení matic, implementace

```
subroutine mult_matrices(A,B,C)

  implicit none
  double precision    :: A(:,:)
  double precision    :: B(:,:)
  double precision    :: C(:,:)
  !-------------------------------------
  integer             :: i,j,k
  !-------------------------------------------------------------

  if( size(A,2) .ne. size(B,1) ) then
    stop 'Error: Illegal shape of A and B matrices!'
  end if

  do i=1,size(A,1)
    do j=1,size(B,2)
      C(i,j) = 0.0d0
      do k=1,size(A,2)
          C(i,j) = C(i,j) + A(i,k)*B(k,j)
      end do
    end do
  end do

end subroutine mult_matrices
```

Počet operací

Za předpokladu, že matice **A** a **B** jsou čtvercové o rozměrech NxN:



Ve výpočetní technice se pro posuzování výpočetního výkonu používá hodnota udávaná ve **FLOPS (FLoating-point Operations Per Second)**, která vyjadřuje kolik operací v pohyblivé čárce dané zařízení vykoná za sekundu.

Výsledky

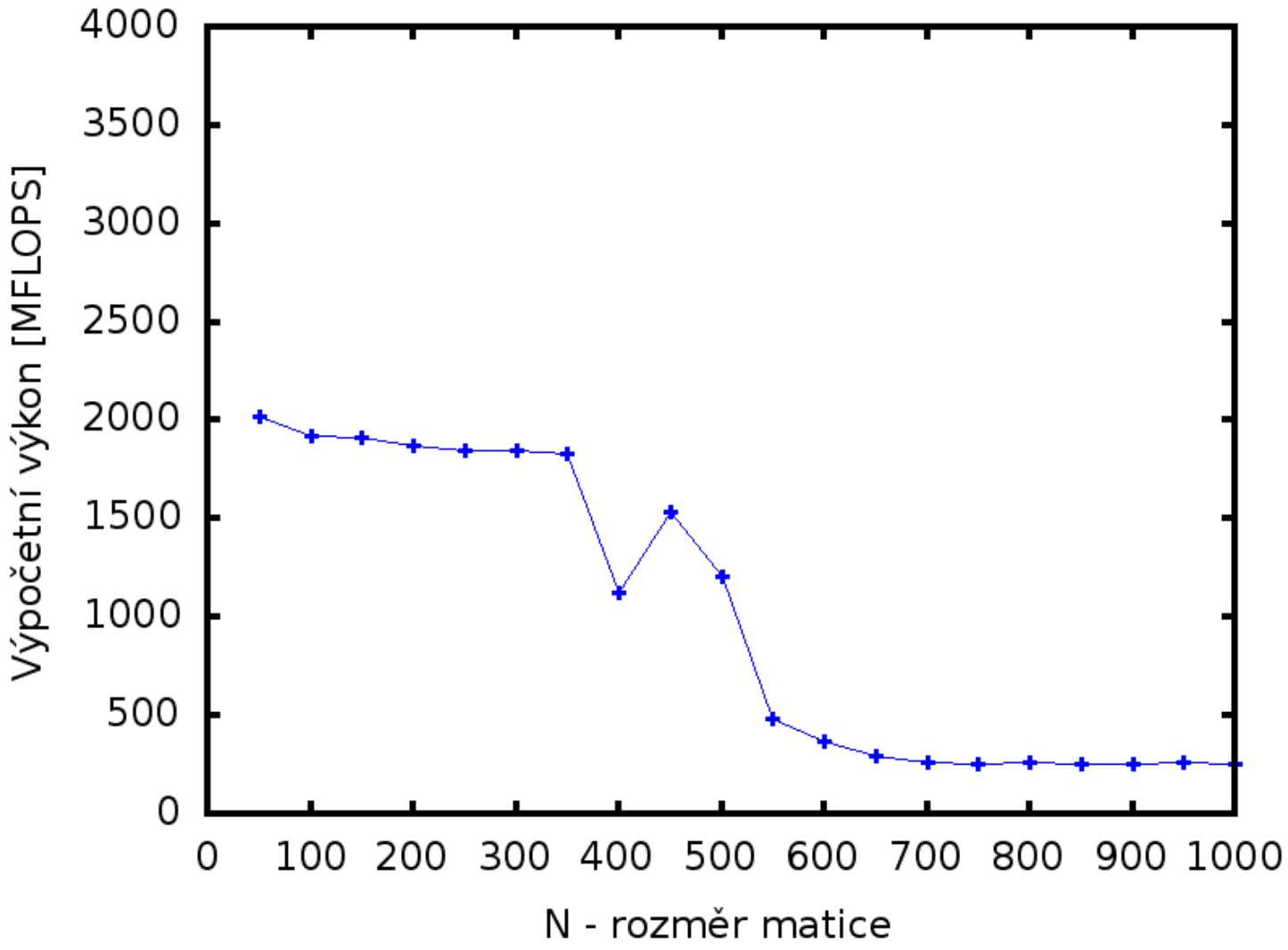
wolf21: gfortran 4.6.3, optimalizace O3, Intel(R) Core(TM) i5 CPU 750 @ 2.67GHz

N	NR	NOPs	Time	MFLOPS
50	50000	12500000000	6.1843858	2021.2
100	500	1000000000	0.5200334	1923.0
150	50	337500000	0.1760106	1917.5
200	50	800000000	0.4280272	1869.0
250	50	1562500000	0.8440533	1851.2
300	50	2700000000	1.4640903	1844.1
350	50	4287500000	2.3441458	1829.0
400	50	6400000000	5.7083569	1121.2
450	50	9112500000	5.9363708	1535.0
500	50	12500000000	10.3366470	1209.3
550	1	332750000	0.6880417	483.6
600	1	432000000	1.1600723	372.4
650	1	549250000	1.8601189	295.3
700	1	686000000	2.5881615	265.1
750	1	843750000	3.2762032	257.5
800	1	1024000000	3.8522377	265.8
850	1	1228250000	4.7883034	256.5
900	1	1458000000	5.6963577	256.0
950	1	1714750000	6.5044060	263.6
1000	1	2000000000	7.9444962	251.7

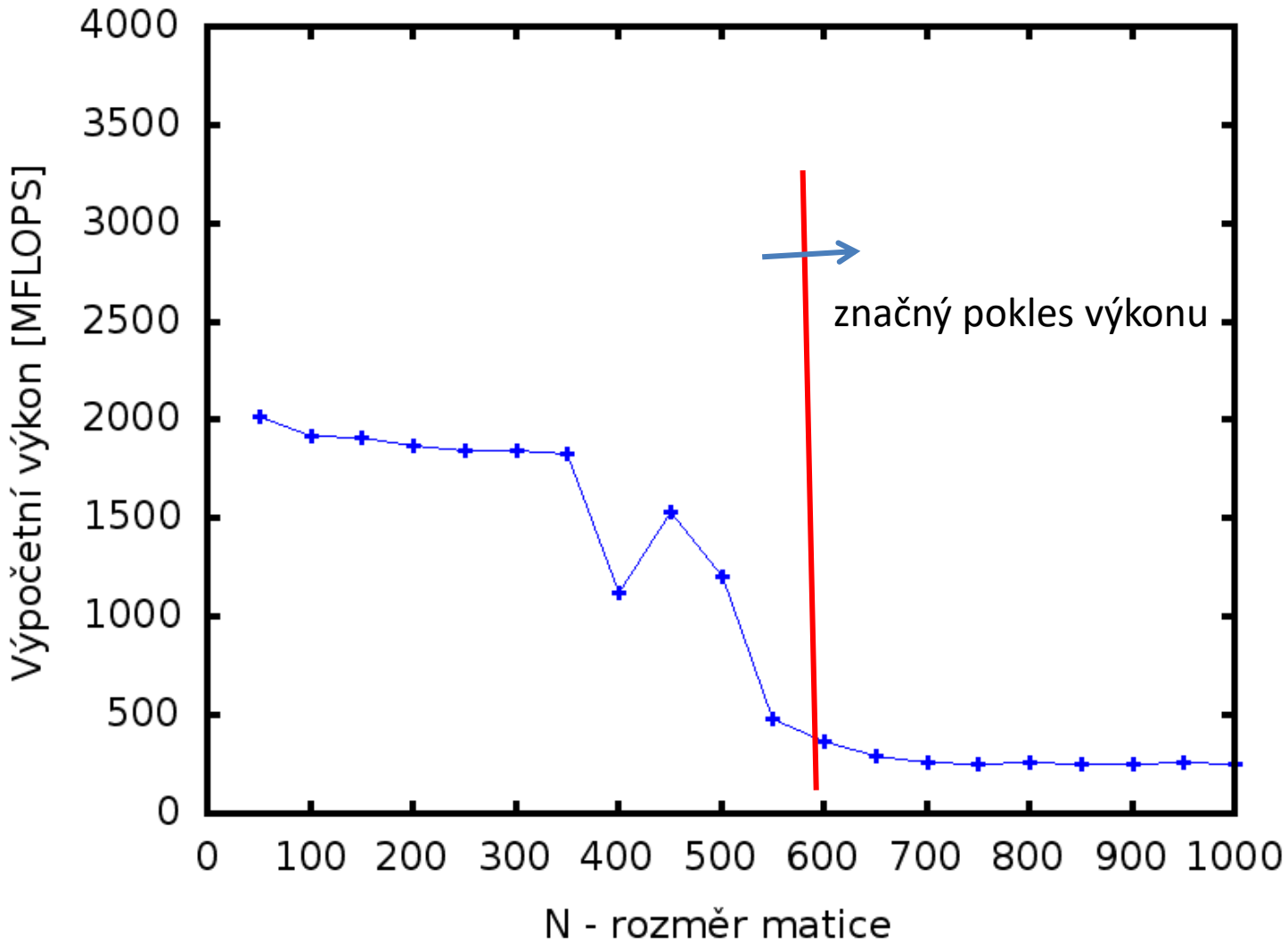
Legenda:

N – rozměr matice
NR – počet opakování
NOPs – počet operaci v FP
Time – doba vykonávání v s
MFLOPS – výpočetní výkon

Výsledky


Výpočetní výkon [MFLOPS]
4000
3500
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700
800
900
1000
N - rozměr matice

Výsledky


4000
3500
3000
2500
2000
1500
1000
500
0
Výpočetní výkon [MFLOPS]
0
100
200
300
400
500
600
700
800
900
1000
N - rozměr matice
značný pokles výkonu

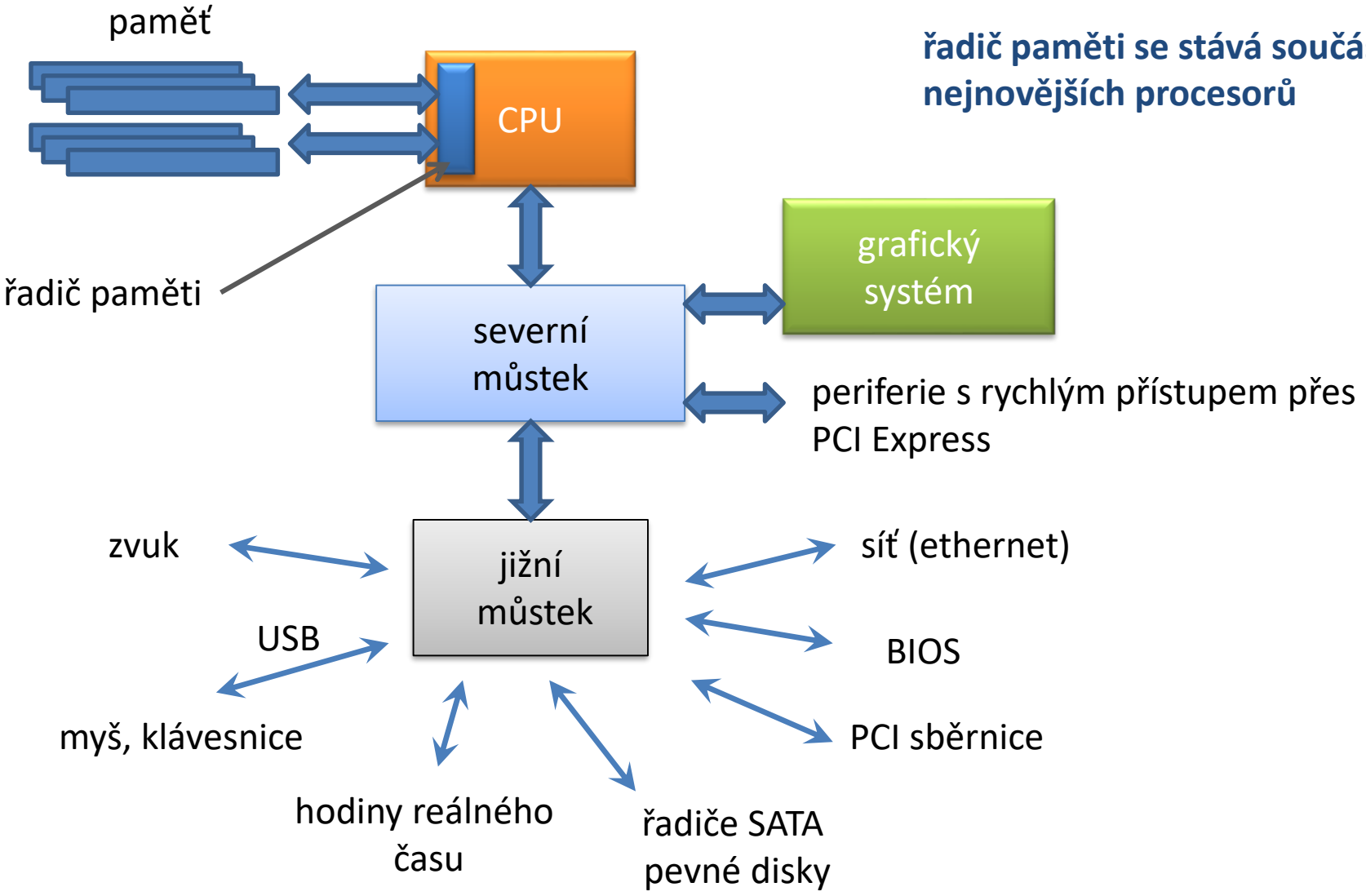
Cvičení 3

Zdrojové kódy jsou umístěny v: **/home/kulhanek/Data/C2115/Lesson10**

1. Zkompilujte program **mult_mat_naive_dp.f90** kompilátorem **gfortran**, použijte **-O3** optimalizaci.
2. Program spusťte a získanou závislost výpočetního výkonu v závislosti na rozměru matice zobrazte ve formě grafu (použijte interaktivní režim programu gnuplot).
3. Postupně určete výpočetní výkon pro optimalizační úrovně **-O3**, **-O2**, **-O1** a **-O0**. Získané závislosti zobrazte v jednom grafu včetně popisu os a legendy (použijte neinteraktivní režim programu gnuplot).
4. Zkompilujte program **mult_mat_naive_sp.f90** kompilátorem **gfortran**, použijte **-O3** optimalizaci.
5. Porovnejte výpočetní výkon pro jednoduchou (**sp**) a dvojitou (**dp**) přesnost. Získané závislosti zobrazte v jednom grafu včetně popisu os a legendy (použijte neinteraktivní režim programu gnuplot).
6. Diskutujte získané výsledky.

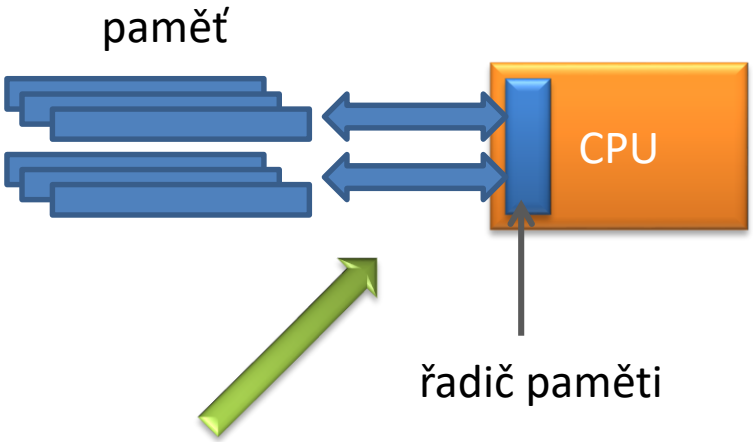
Architektura počítače

Architektura, celkový pohled



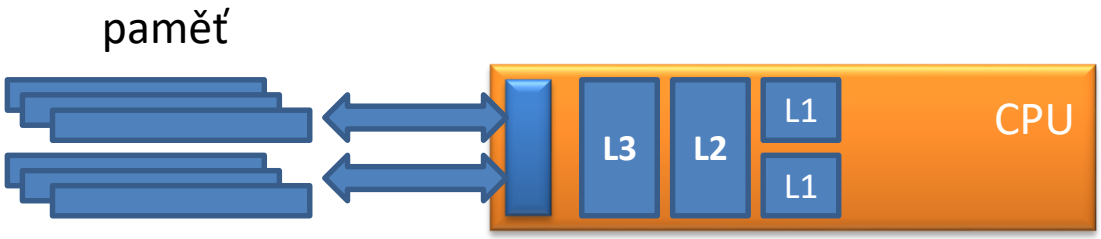
řadič paměti se stává součástí nejnovějších procesorů

Architektura, úzké hrdlo



úzké hrdlo: rychlost přenosu dat mezi pamětí a CPU je pomalejší než rychlost s jakou je CPU data schopno zpracovávat

Hierarchický model paměti

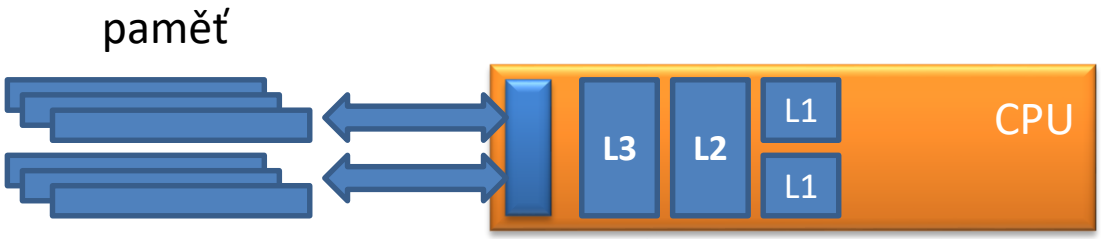


rychlá mezipaměť (cache), různé úrovně s různými přístupovými rychlostmi

wolf21 – přenosové rychlosti (memtest86+, http://www.memtest.org/)

Typ	Velikost	Rychlost
L1	32kB	89 GB/s
L2	256 kB	35 GB/s
L3	8192 kB	24 GB/s
paměť	8192 MB	12 GB/s

Hierarchický model paměti



rychlá mezipaměť (cache), různé úrovně s různými rychlostmi

wolf21 – přenosové rychlosti (memtest86+, http://www.memtest.org/)

Typ	Velikost	Rychlost
L1	32kB	89 GB/s
L2	256 kB	35 GB/s
L3	8192 kB	24 GB/s
paměť	8192 MB	12 GB/s

Jakmile velikost problému přesáhne velikost mezipaměti CPU, **rychlost určujícím krokem** se stává rychlost přenosu dat mezi fyzickou pamětí a CPU.

N=600

600x600x3x8 = 8437 kB

A,B,C double precision

Knihovny pro lineární algebru

BLAS

The BLAS (**Basic Linear Algebra Subprograms**) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

LAPACK

LAPACK is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

http://netlib.org

Optimalizované knihovny

Optimalizované knihovny BLAS a LAPACK

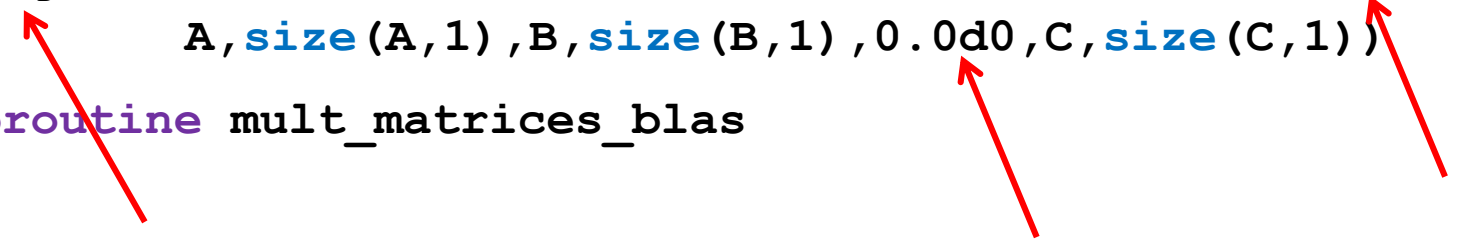
- optimalizované dodavatelem hardware
- ATLAS http://math-atlas.sourceforge.net/
- MKL http://software.intel.com/en-us/intel-mkl
- ACML http://developer.amd.com/tools/cpu-development/amd-core-math-library-acml/
- cuBLAS https://developer.nvidia.com/cublas

Optimalizované knihovny FFT (Fast Fourier Transform)

- optimalizované dodavatelem hardware
- MKL http://software.intel.com/en-us/intel-mkl
- ACML http://developer.amd.com/tools/cpu-development/amd-core-math-library-acml/
- FFTW http://www.fftw.org/
- cuFFT https://developer.nvidia.com/cufft

Násobení matic pomocí BLAS - dp

```
subroutine mult_matrices_blas(A,B,C)

  implicit none
  double precision    :: A(:,:)
  double precision    :: B(:,:)
  double precision    :: C(:,:)
!-----------------------------------------------------

  if( size(A,2) .ne. size(B,1) ) then
    stop 'Error: Illegal shape of A and B matrices!'
  end if

  call dgemm('N','N',size(A,1),size(B,2),size(A,2),1.0d0, &
             A,size(A,1),B,size(B,1),0.0d0,C,size(C,1))

end subroutine mult_matrices_blas
```



F77 rozhraní BLAS knihovny neobsahuje informace o typech argumentů. Programátor musí zadat všechny argumenty ve správném pořadí a typu!!!!

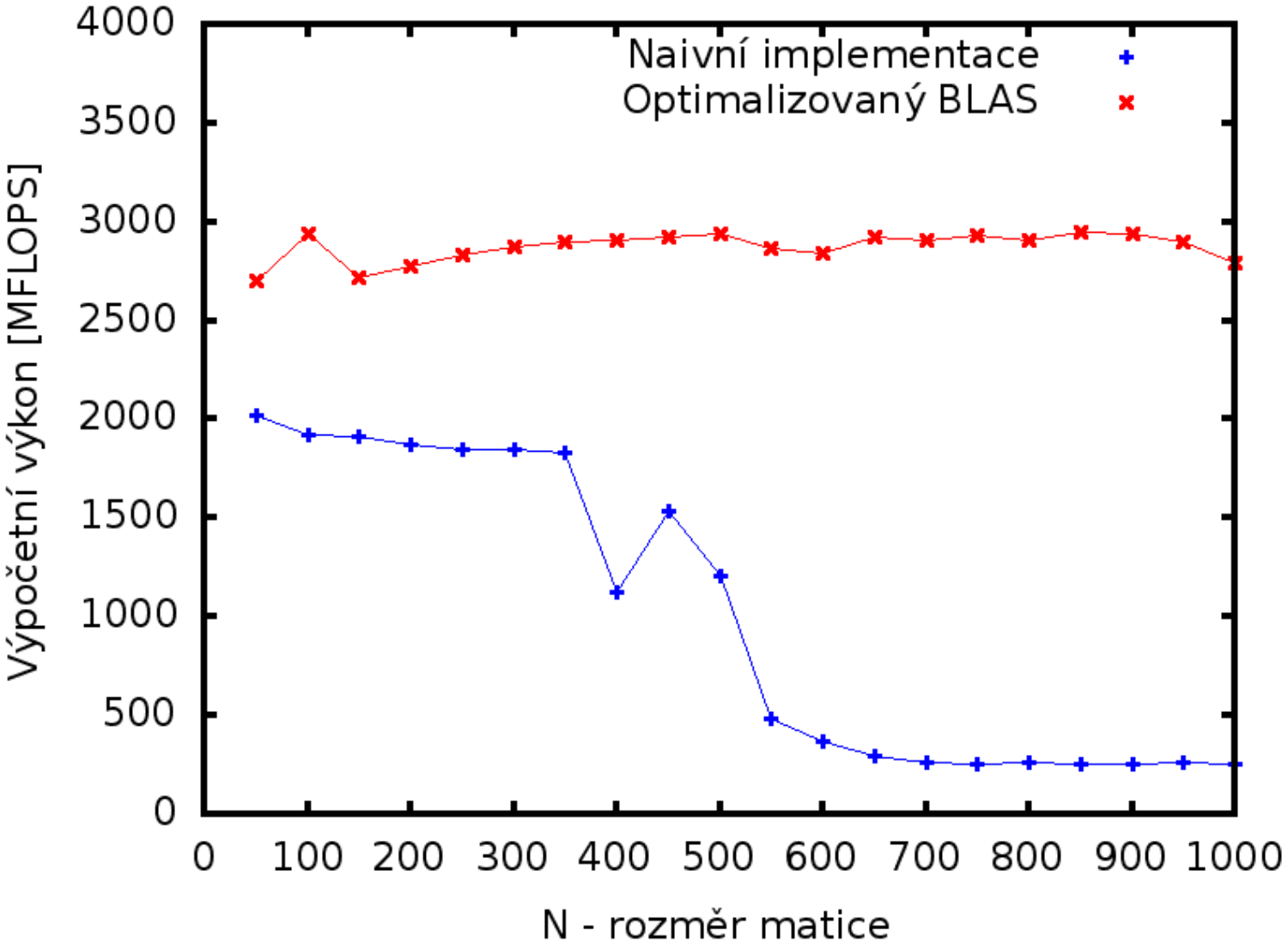
Násobení matic pomocí BLAS - sp

```
subroutine mult_matrices_blas(A,B,C)

  implicit none
  real(4)     :: A(:,:)
  real(4)     :: B(:,:)
  real(4)     :: C(:,:)
!-----------------------------------------------------

  if( size(A,2) .ne. size(B,1) ) then
    stop 'Error: Illegal shape of A and B matrices!'
  end if

  call sgemm('N','N',size(A,1),size(B,2),size(A,2),1.0, &
             A,size(A,1),B,size(B,1),0.0,C,size(C,1))

end subroutine mult_matrices_blas
```

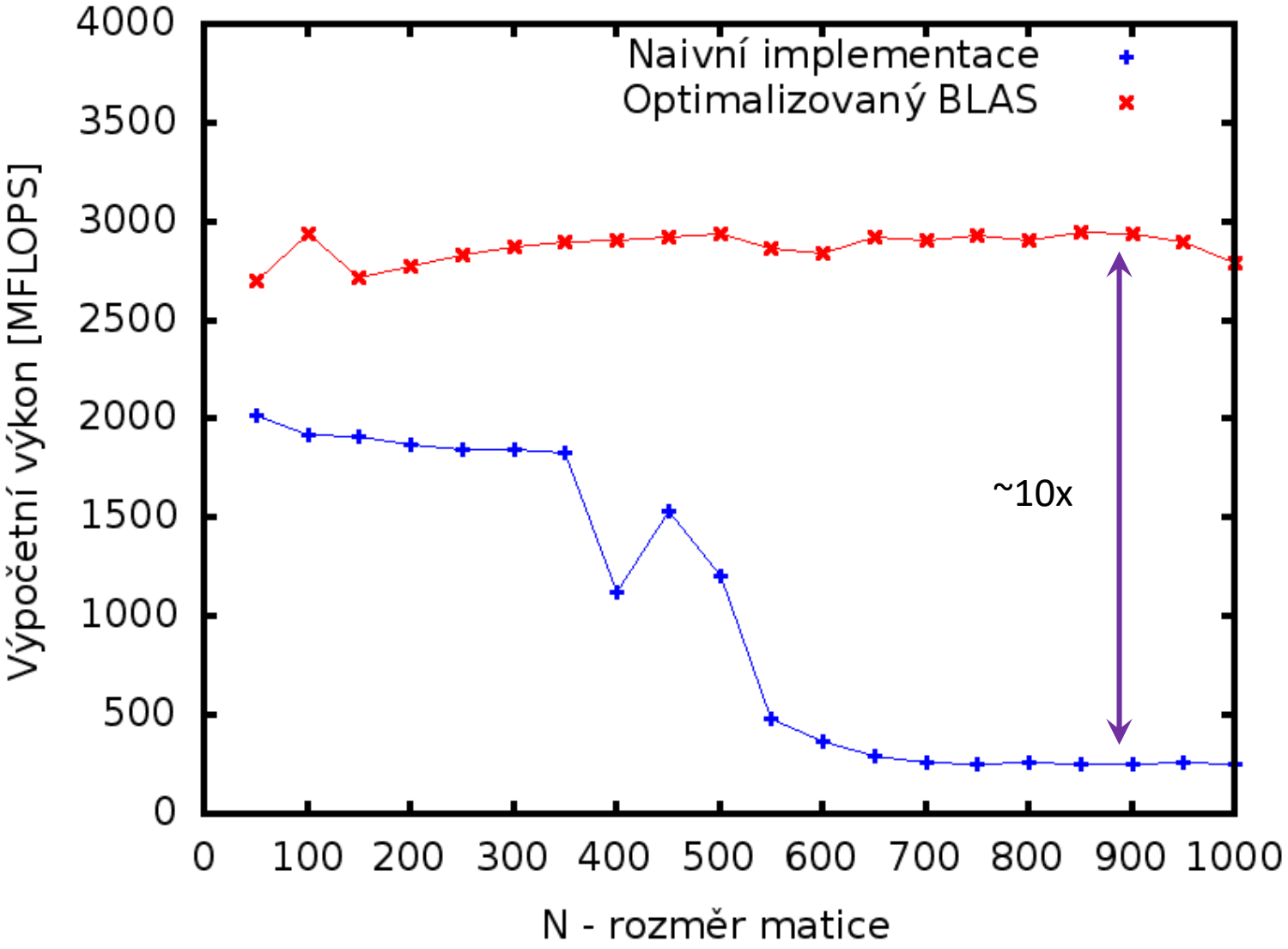
Kompilace:

```
$ gfortran -03 mutl_mat.f90 -o mult_mat -lblas
```

Naivní *vs* optimalizované řešení



Naivní *vs* optimalizované řešení



Závěr

K řešení problémů je vždy vhodné využít **existující softwarové knihovny**, u kterých lze očekávat, že jsou pro daný problém a **hardware značně optimalizované**.

Cvičení 4

Zdrojové kódy jsou umístěny v: **/home/kulhanek/Data/C2115/Lesson10**

1. Zkompilujte program **mult_mat_blas_dp.f90** kompilátorem **gfortran**, použijte **-O3** optimalizaci.
2. Program spusťte a získanou závislost výpočetního výkonu v závislosti na rozměru matice zobrazte ve formě grafu (použijte interaktivní režim programu gnuplot).
3. Určete výpočetní výkon pro optimalizační úrovně **-O3** a **-O0**. Pozorovaný rozdíl diskutujte.
4. Zkompilujte program **mult_mat_blas_sp.f90** kompilátorem **gfortran**, použijte **-O3** optimalizaci.
5. Porovnejte výpočetní výkon pro jednoduchou (**sp**) a dvojitou (**dp**) přesnost jak pro naivní implementaci tak i pro implementaci využívající knihovnu BLAS. Získané závislosti zobrazte v jednom grafu včetně popisu os a legendy (použijte neinteraktivní režim programu gnuplot).
6. Diskutujte získané výsledky.