

# C2115

# Praktický úvod do superpočítání

IV. lekce

Petr Kulhánek, Jakub Štěpán

[kulhanek@chemi.muni.cz](mailto:kulhanek@chemi.muni.cz)

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

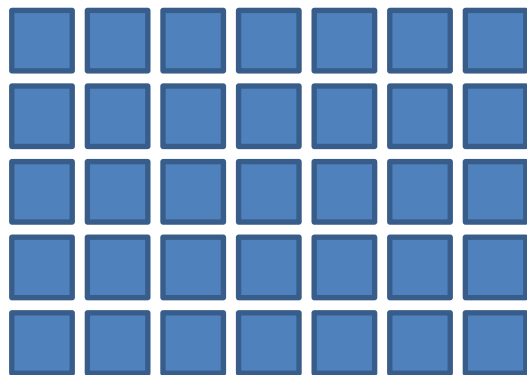
- **Řešení cvičení LIII.3**  
zadání, násobení matic
- **Vysvětlení získaných výsledků**  
architektura počítače a její úzká hrdla
- **Použití optimalizovaných knihoven**  
BLAS, LAPACK, LINPACK, porovnání výsledků

# Řešení cvičení LIII.3

# Cvičení LIII.3

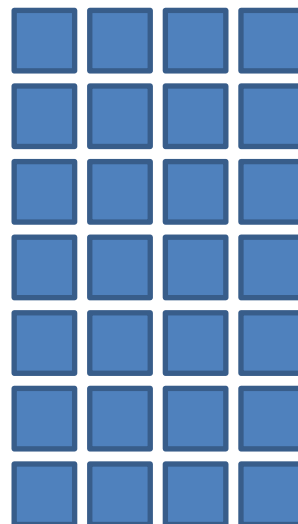
1. Vytvořte program, který dynamicky alokuje dvojrozměrné pole **A** o velikosti **n x n**. Prvky pole inicializujte náhodnými čísly z intervalu  $\langle -10 ; 20 \rangle$ . Pole vytiskněte na obrazovku.
2. Vytvořte dvě dvourozměrné pole (matice) **A** a **B** o rozměrech **n x n**. Pole inicializujte náhodnými čísly stejným způsobem jako v předchozím cvičení. Napište program pro násobení matic **A** a **B**. Výsledek uložte do matice **C**.
3. Kolik operací v plovoucí čárce program vykoná při násobení matic? Změřte dobu, za kterou program uskuteční násobení matic (do času nezapočítávejte vytvoření a inicializaci matic A a B). Z počtu operací a doby potřebné k násobení stanovte přibližný výkon procesoru v MFLOPS.
4. Změřte výkon procesoru pro různé velikosti matic **A** a **B**. Naměřené hodnoty pro **n** v intervalu 10 až 1000 vynesete do grafu a vyhodnoťte.

# Násobení matic



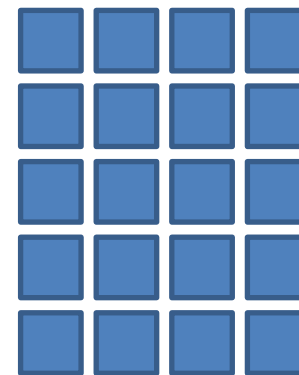
**A(n,m)**

**x**



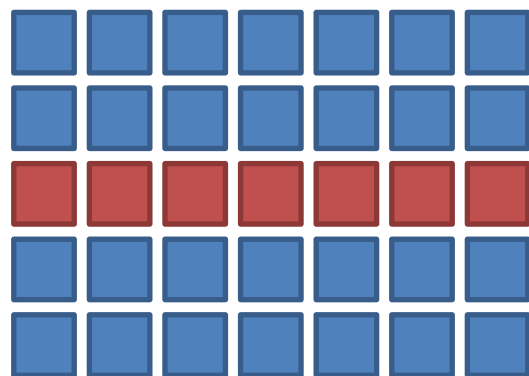
**B(m,k)**

**=**



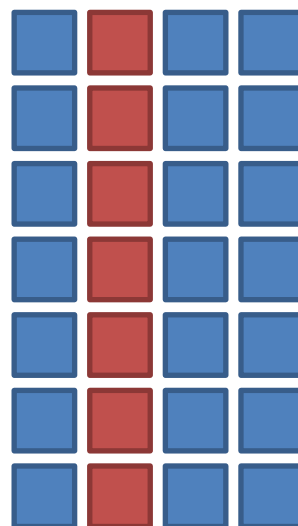
**C(n,k)**

# Násobení matic



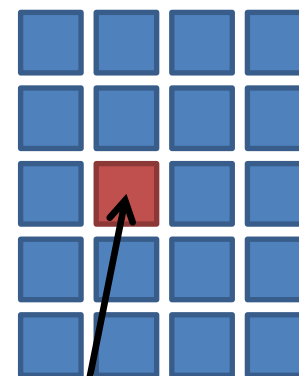
$A(n,m)$

$\times$



$B(m,k)$

$=$



$C(n,k)$

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

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

# Násobení matic, program

```
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

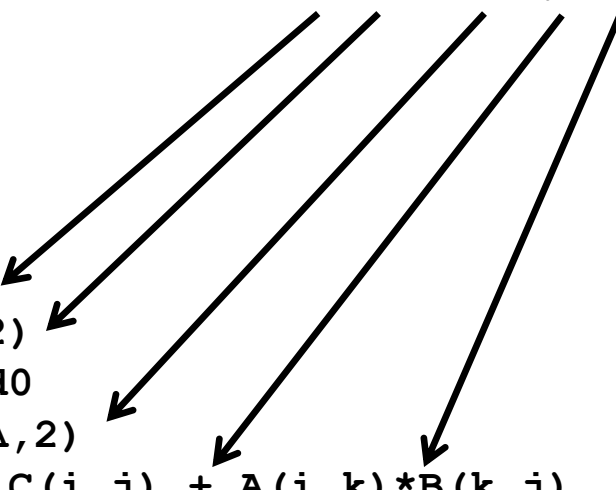
end subroutine mult_matrices
```

# Počet operací

Za předpokladu, že matice **A** a **B** jsou čtvercové o rozměrech  $N \times N$ :

$$N * N * N * (1 + 1) = 2 * N^3$$

```
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
```



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

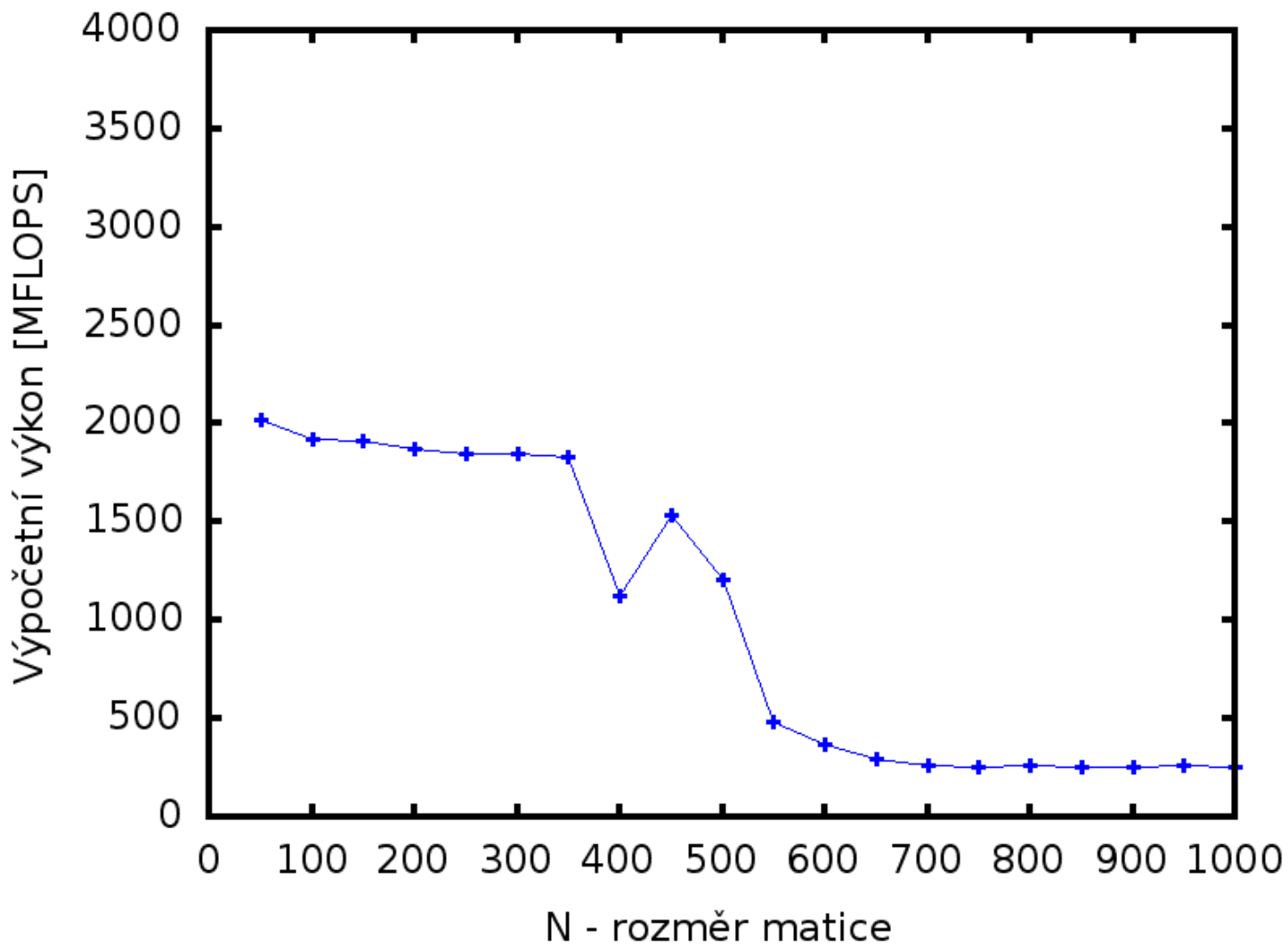
NR – počet opakování

NOPs – počet operací v FP

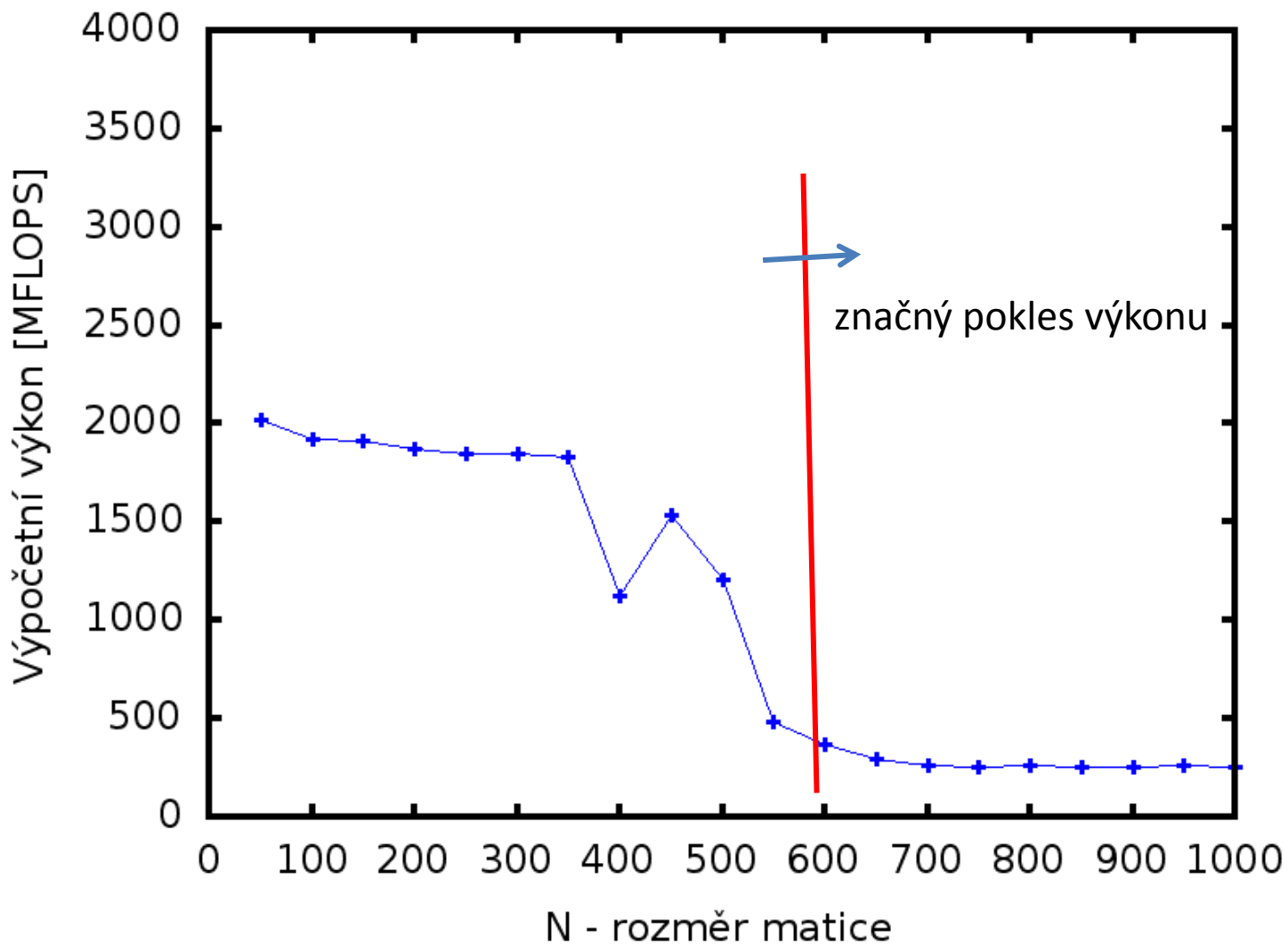
Time – doba vykonávání v s

MFLOPS – výpočetní výkon

# Výsledky

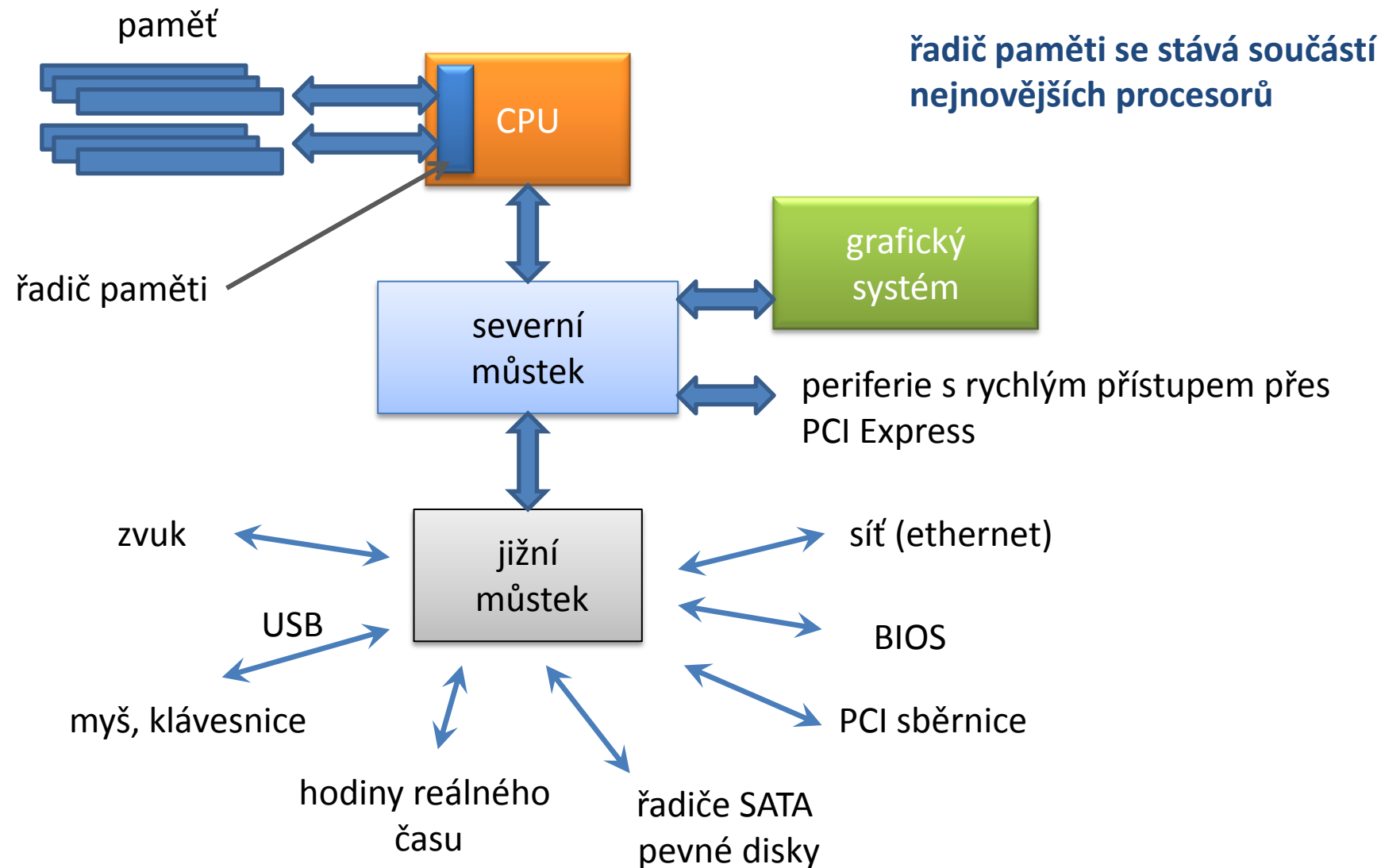


# Výsledky

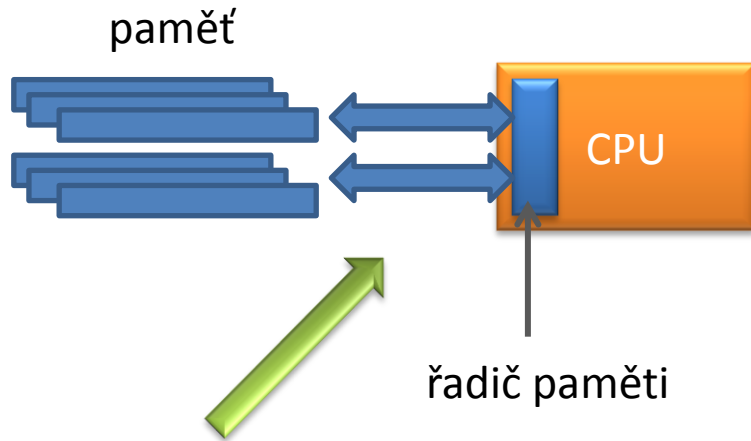


# Architektura počítače

# Architektura, celkový pohled

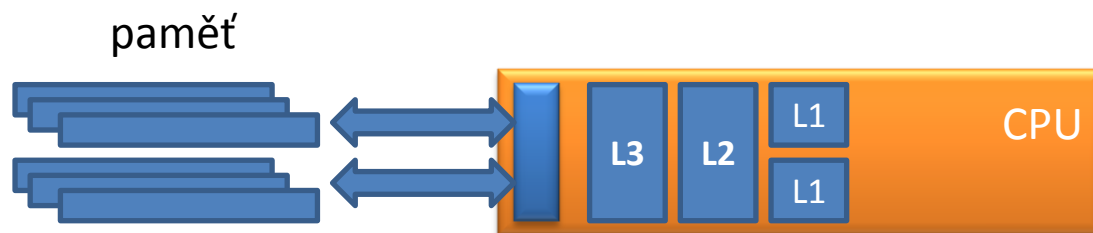


# Architektura, úzké hrdlo



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

# Hierarchický model paměti

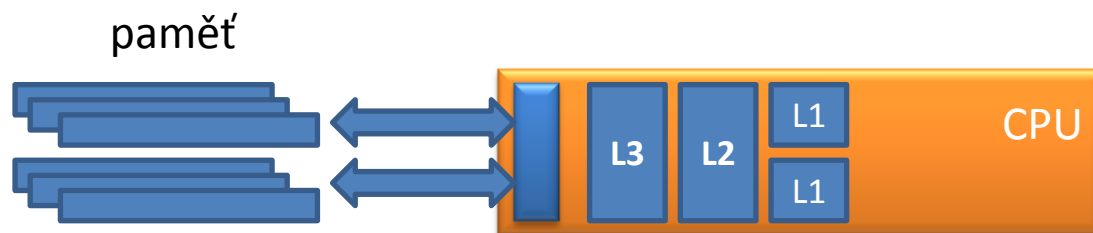


**rychlá mezipaměť** (cache), různé úrovně s různými přístupovými rychlostmi

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.

$N=600$

$$600 \times 600 \times 3 \times 8 = 8437 \text{ kB}$$

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

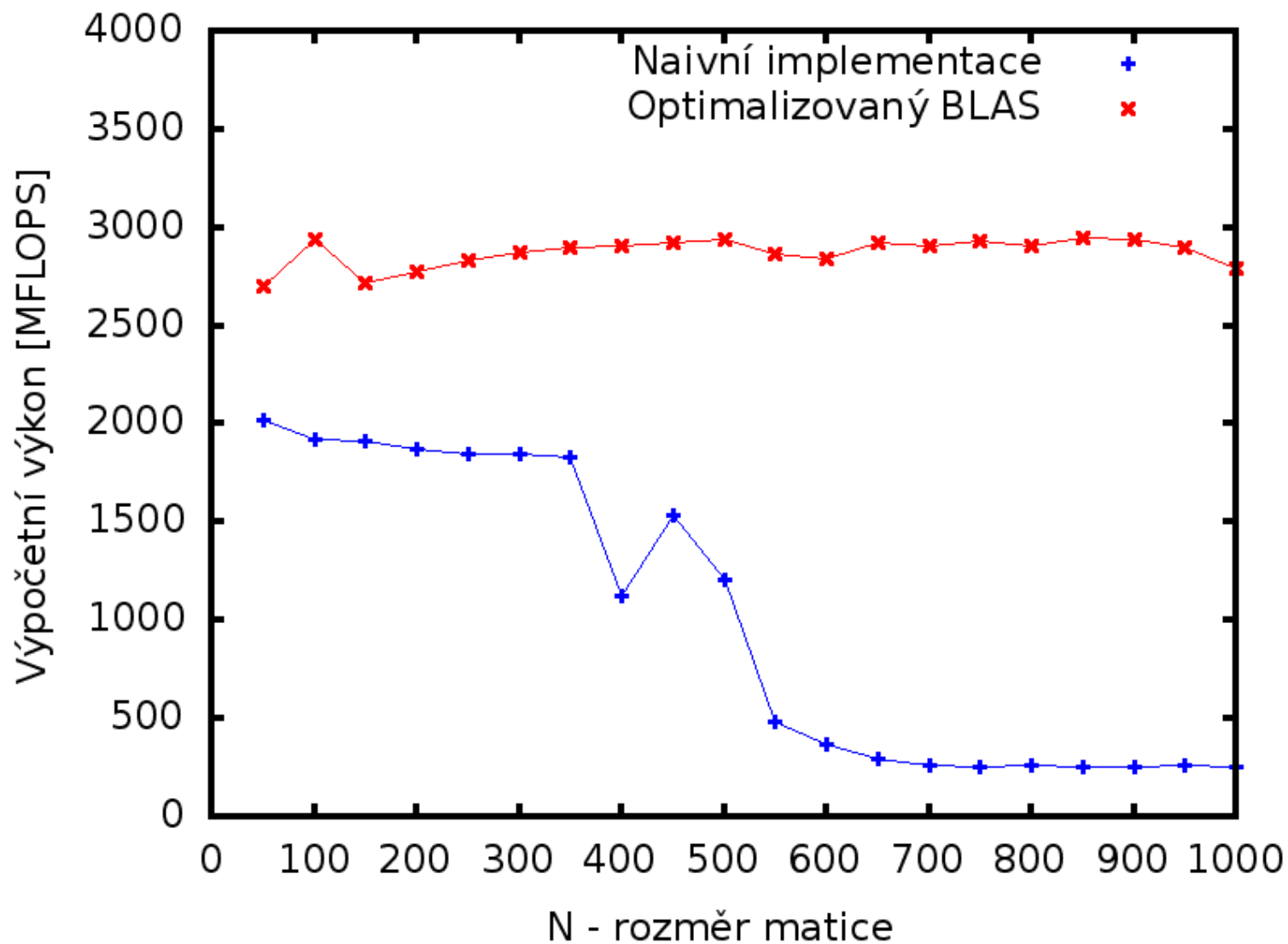
```
subroutine mult_matrices_blas(A,B,C)
    implicit none
    double precision      :: A(:, :)
    double precision      :: B(:, :)
    double precision      :: C(:, :)
!-----
    if( size(A,2) .ne. size(B,1) ) then
        stop 'Error: Illegal shape of A and B matrices!'
    end if

    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
```

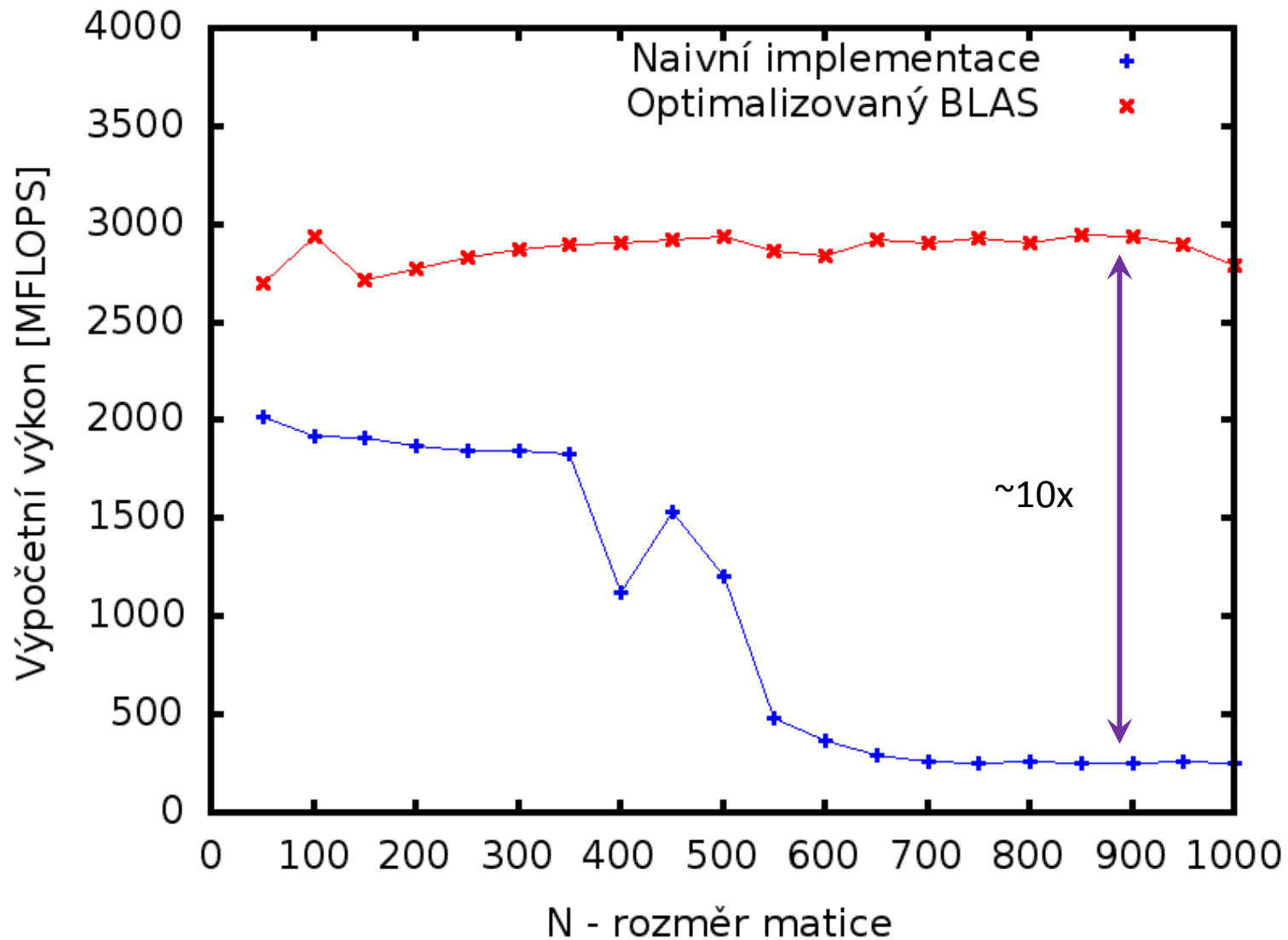
## Kompilace:

```
$ gfortran -O3 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**.