C2115 Practical Introduction to Supercomputing

5th Lesson

Petr Kulhánek, Jakub Štěpán

kulhanek@chemi.muni.cz

National Centre for Biomolecular Research, Faculty of Science
Masaryk University, Kotlářská 2, CZ-61137 Brno

european social fund in the czech republic
EUROPEAN UNION
MINISTRY OF EDUCATION, YOUTH AND SPORTS
OP Education for Competitivness

INVESTMENTS IN EDUCATION DEVELOPMENT

CZ.1.07/2.2.00/15.0233

Contents

- **Exercise LIII.3 solution**

  input, matrix multiplication

- **Result explanation**

  computer architecture and its bottlenecks

- **Optimized libraries usage**

  BLAS, LAPACK, LINPACK, result comparison

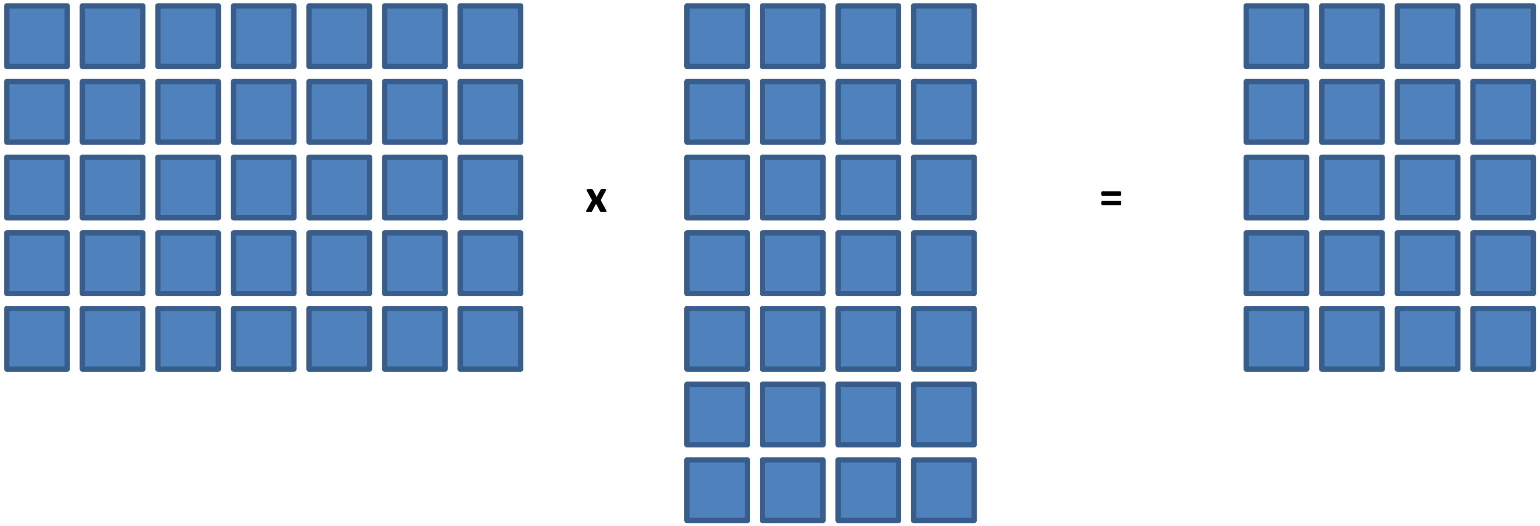
Exercise LIII.3 solution

- **Input, matrix multiplication**

Exercise LIII.3

1. Write program, that dynamically allocates two dimensional array **A** of size **n** x **n**. Items will be initialized by random numbers from range <-10 ;20>. Print array to terminal.
2. Create two separate arrays (matrices) **A** and **B** of size **n** x **n**. Initialize arrays in same way as in previous exercise. Write code for matrix **A** and **B** multiplication, save result to matrix **C**.
3. How many floating point operations will be done during matrix multiplication? Measure time necessary for matrix multiplication (do not include matrix initiation and creation). Calculate approximate processor power in MFLOPS from operation number.
4. Calculate processor performance for different matrix **A** and **B** sizes. Create graph for values of **n** in range 10 to 1000.

Matrix multiplication



x

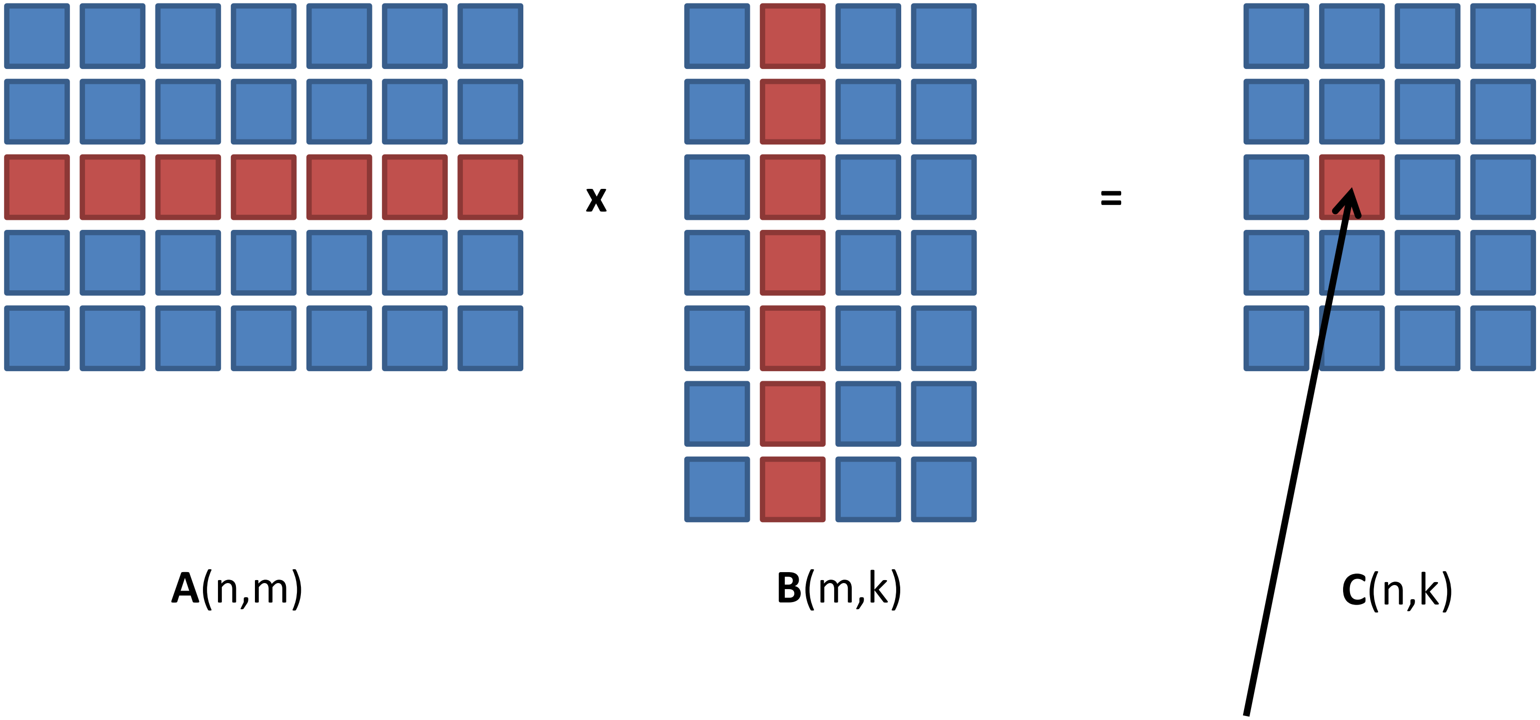
=

A(n,m)

B(m,k)

C(n,k)

Matrix multiplication



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

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

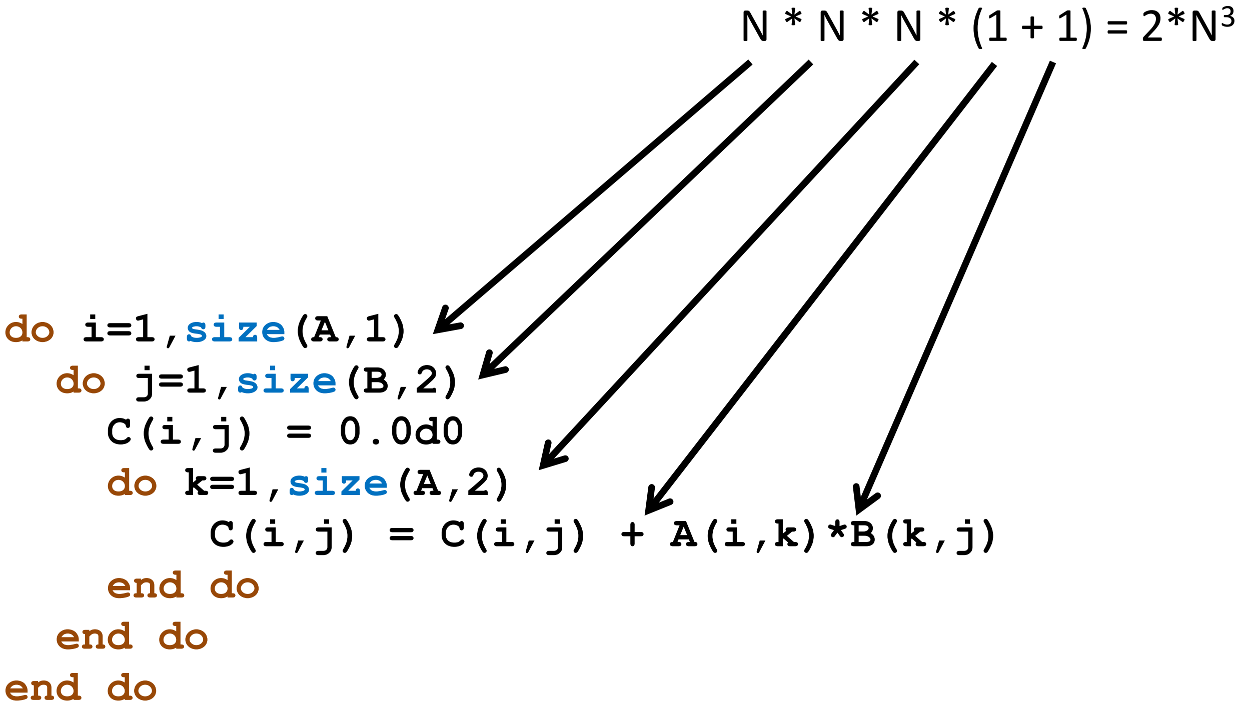
Resulting **C** matrix item is scalar product of vectors given by i-th row of matrix **A** and j-th column of matrix **B**

Matrix multiplication, 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
```

Number of operations

Expect that matrices **A** and **B** are square matrices of NxN size:



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

```
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
```

Computing measures computational performance as number of **FLOPS (FLoating-point Operations Per Second)**, that is how many floating point operations are done in second.

Results

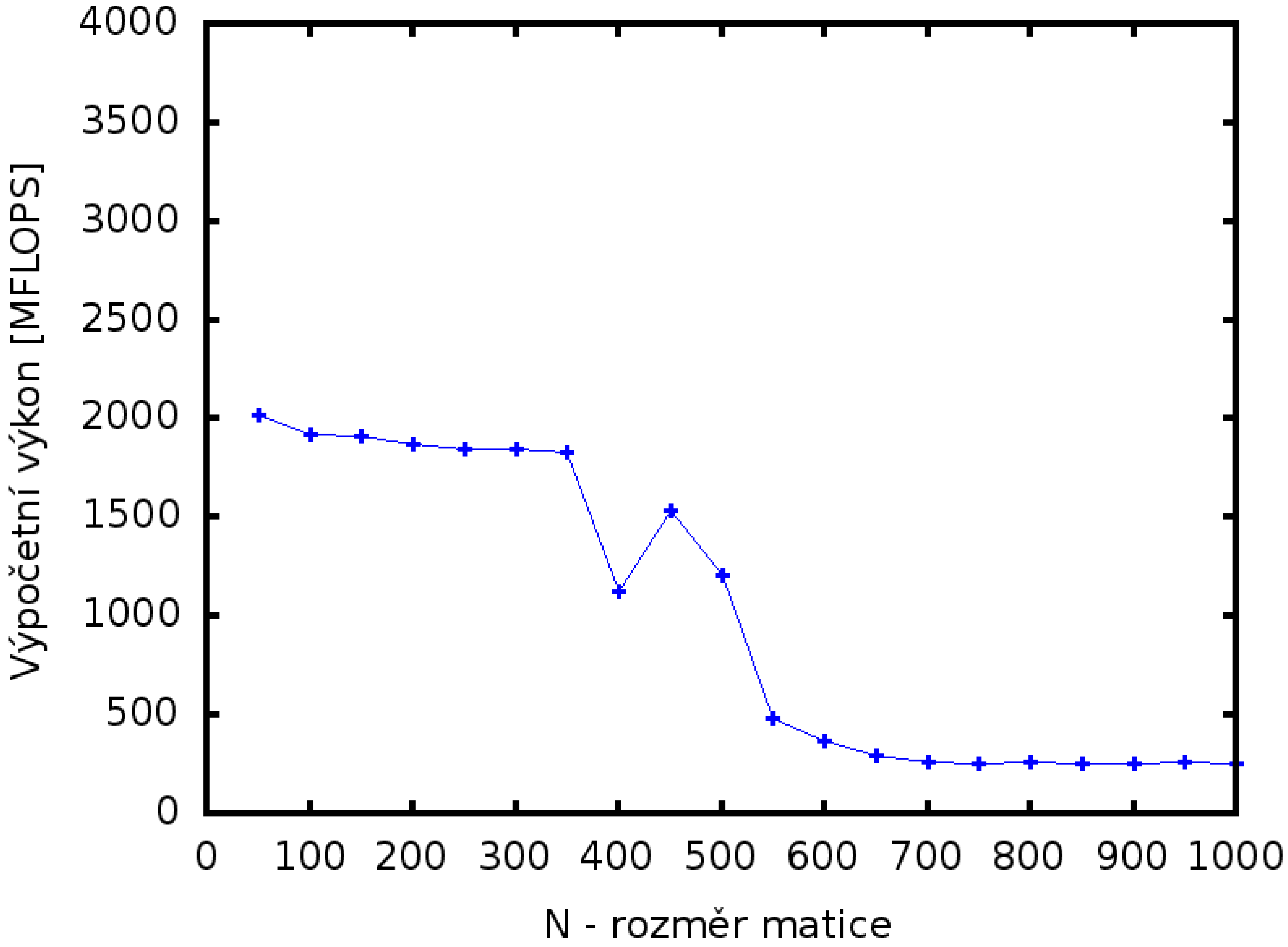
wolf21: gfortran 4.6.3, optimization O3, Intel(R) Core(TM) i5 CPU 750 @ 2.67GHz

```
  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
```

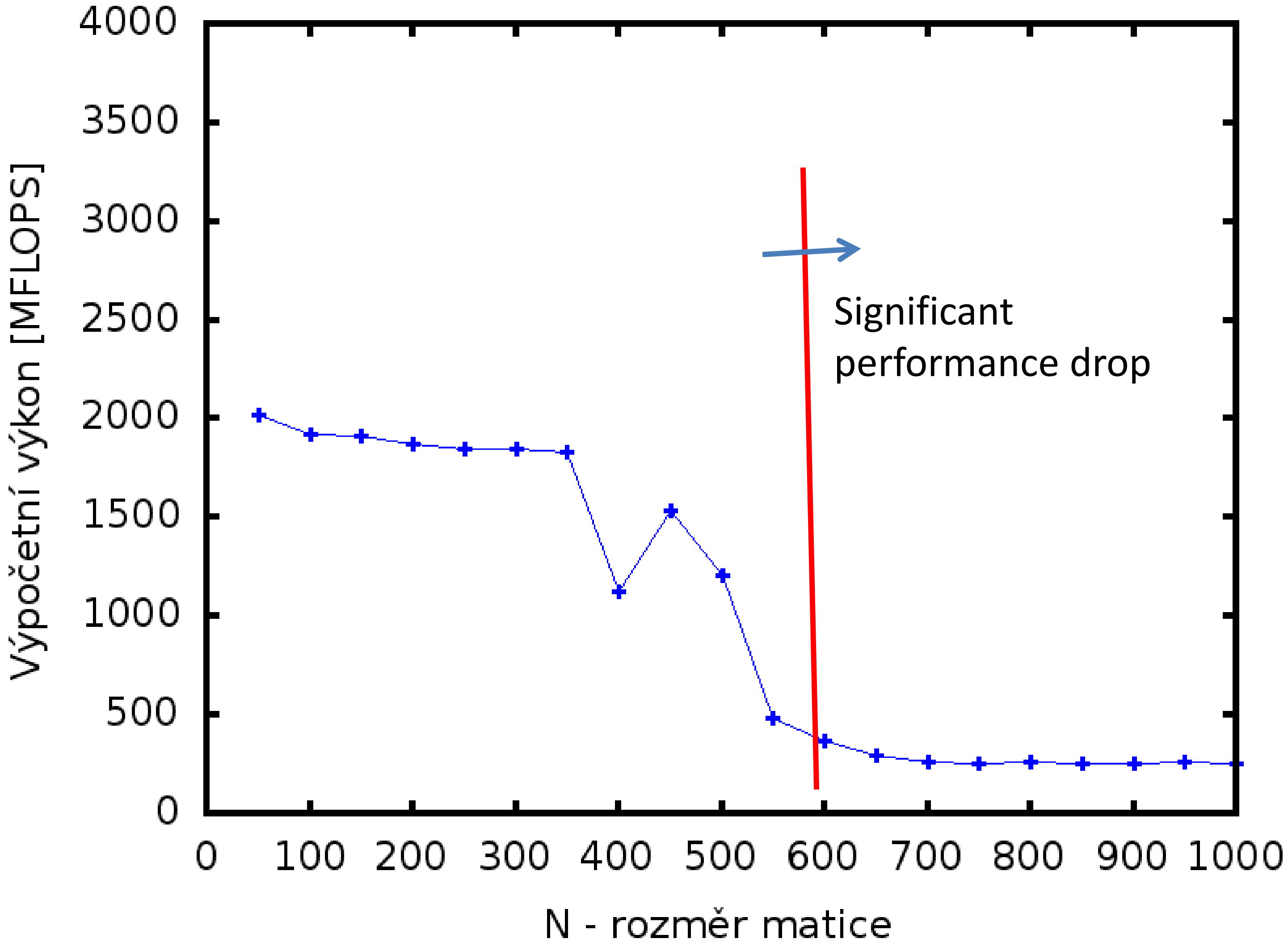
Legend:

N – matrix size
NR – number of iterations
NOPs – Floating Point operations
Time – runtime in seconds
MFLOPS – performance

Results


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

Results

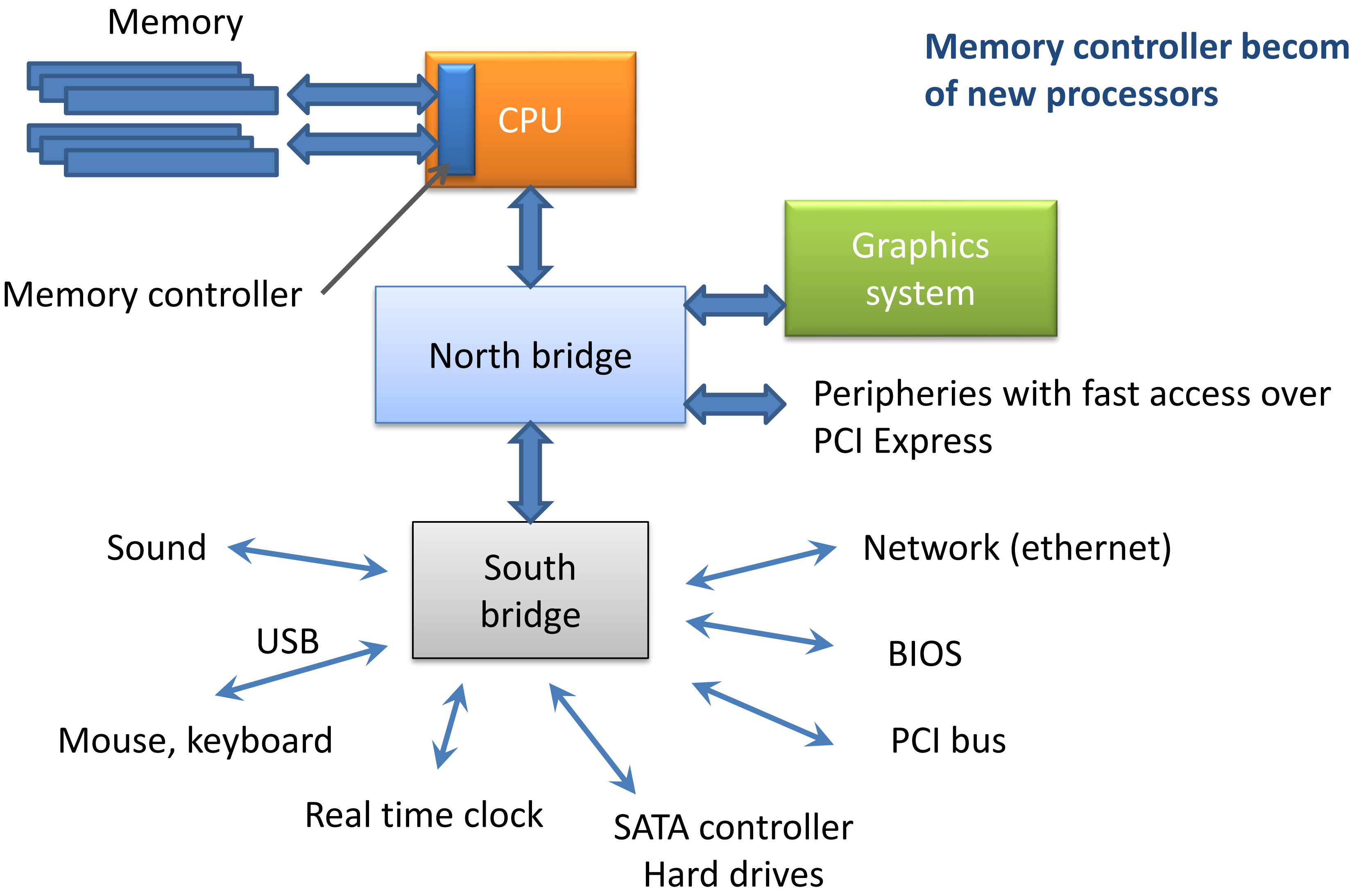


Significant performance drop

Results explanation

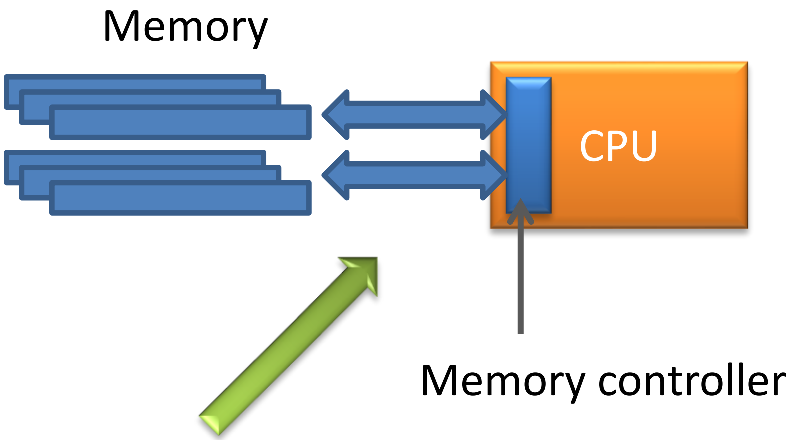
- Computer architecture
- Bottlenecks

Architecture, general overview



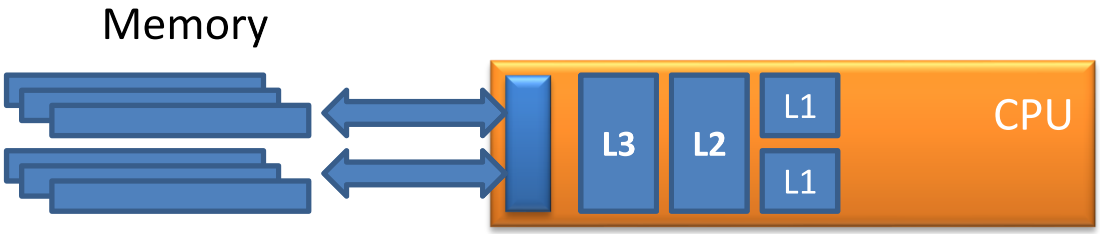
Memory controller becomes part of new processors

Architecture, bottleneck



Bottleneck: data transfer rate between memory and CPU is usually slower then speed that CPU processes data.

Hierarchy memory model

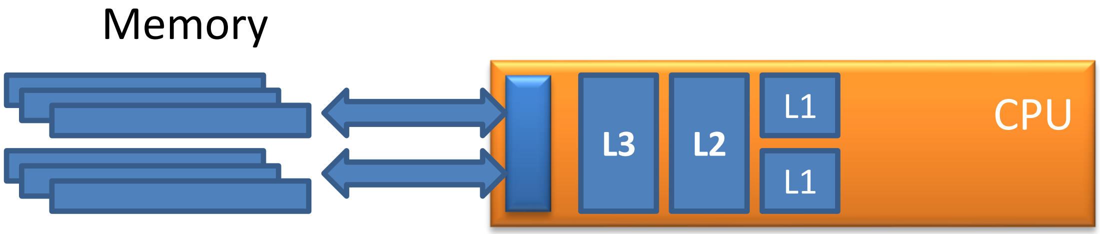


Fast cache memory (cache), various levels with different access rates.

wolf21 – transfer rates (memtest86+, http://www.memtest.org/)

Type	Size	Rate
L1	32kB	89 GB/s
L2	256 kB	35 GB/s
L3	8192 kB	24 GB/s
Memory	8192 MB	12 GB/s

Hierarchy memory model



Fast cache memory (cache), various levels with different access rates.

wolf21 – transfer rates (memtest86+, http://www.memtest.org/)

Type	Size	Rate
L1	32kB	89 GB/s
L2	256 kB	35 GB/s
L3	8192 kB	24 GB/s
Memory	8192 MB	12 GB/s

If problem size exceeds CPU cache memory size, then transfer rate between CPU and physical memory becomes **speed limiting factor**.

N=600

600x600x3x8 = 8437 MB

A,B,C double precision

Optimized libraries usage

- BLAS
- LAPACK
- LINPACK
- Result comparison

Linear algebra libraries

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

Optimized libraries

Optimized libraries BLAS and LAPACK

- optimized by hardware producer

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

Optimized libraries FFT (Fast Fourier Transform)

- optimized by hardware producer

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

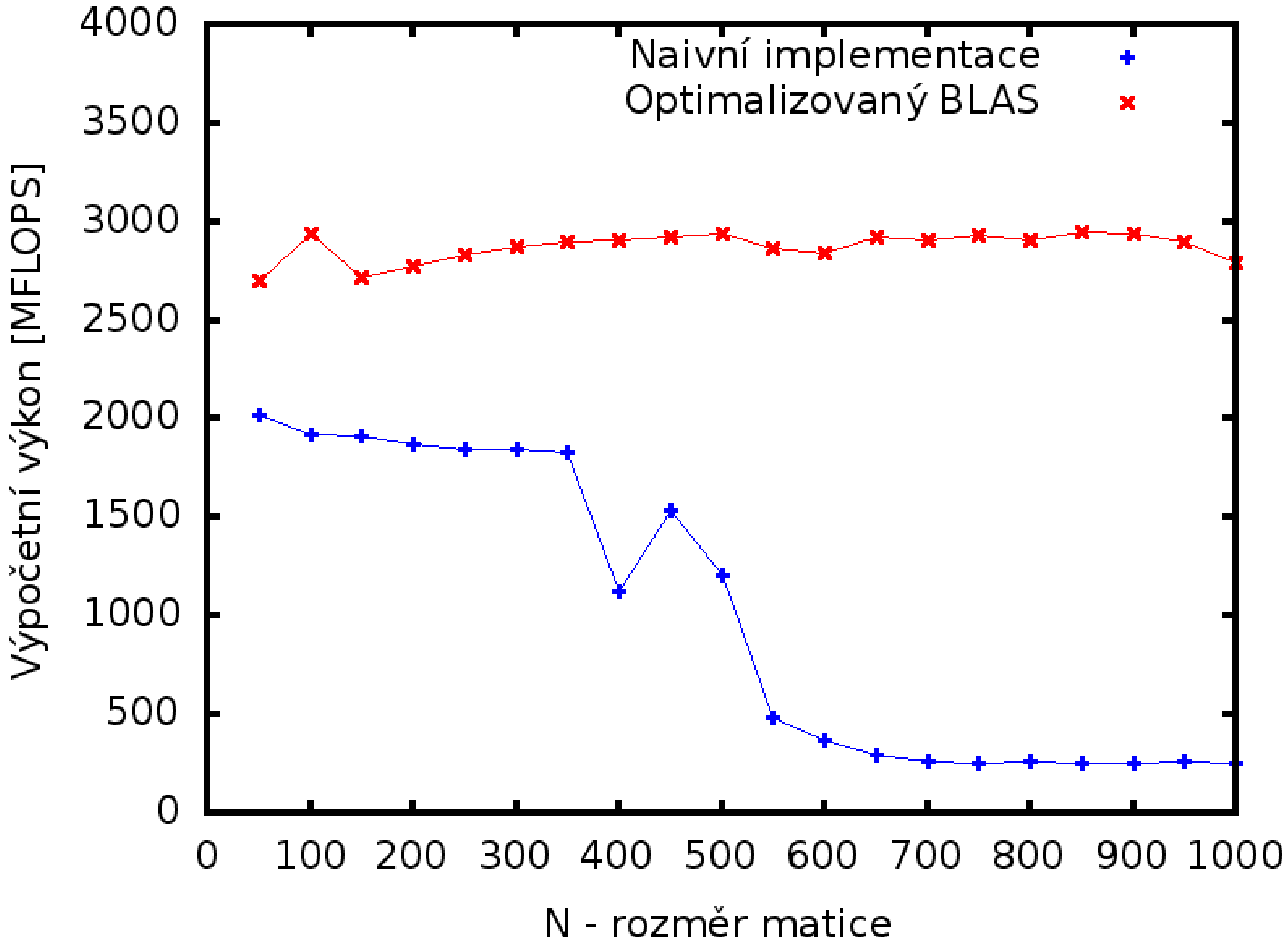
Matrix multiplication using BLAS

```fortran
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
```

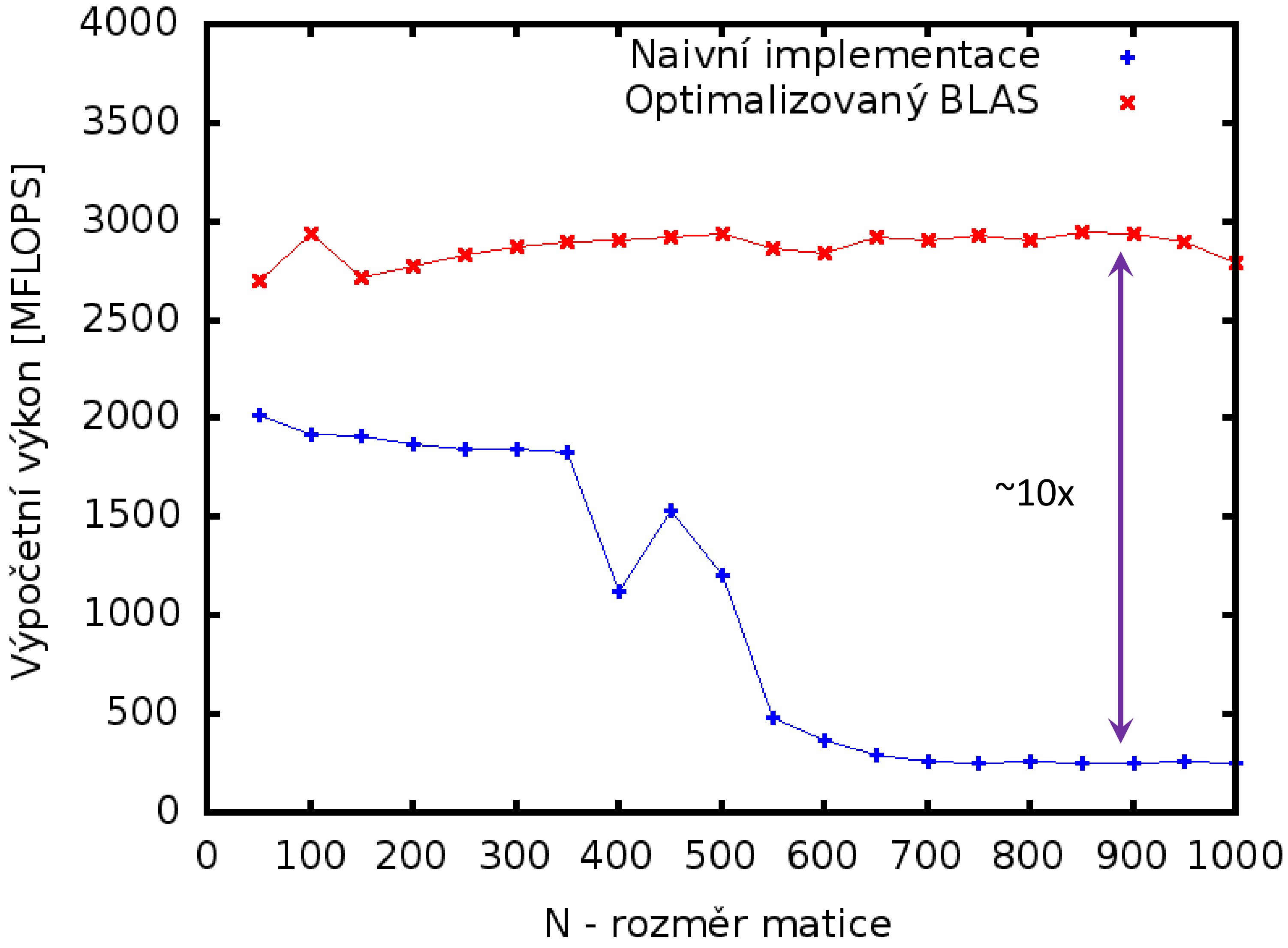
Compilation:

```
$ gfortran -O3 mutl_mat.f90 -o mult_mat -lblas
```

Naive *vs.* optimized solution



Naive *vs.* optimized solution



Summary

It is always appropriate to use **existing library** to solve problem, because these are usually **highly hardware optimized**.