Revision 1

C2115
Practical introduction to supercomputing

Lesson 12

Petr Kulhanek

kulhanek@chemi.muni.cz

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

Contents

- **Representation of numbers in digital technology**

  integers, real numbers

- **From problem to result**

  algorithm, source codes, translation, program execution, programming languages
  - **numerical integration**
  - **matrix multiplication**

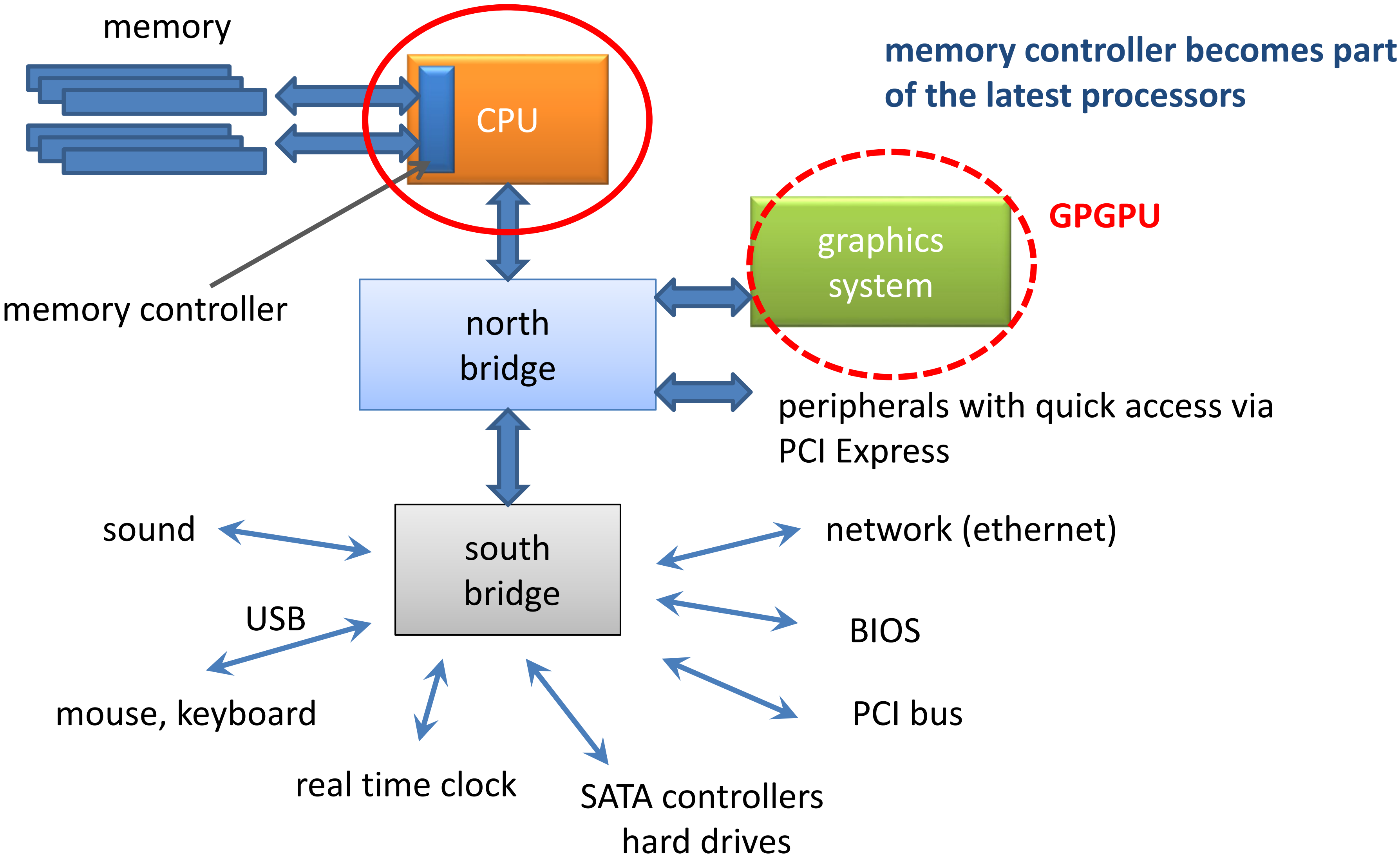
Conclusion

To solve problems, it is always advisable to use **existing software libraries** or **programs**, that are **greatly optimized** for the given problem and **hardware.***

* May not always be appropriate in case of design verification (proof of concept), as the use of optimized approaches may not be trivial at first.

Representation of numbers

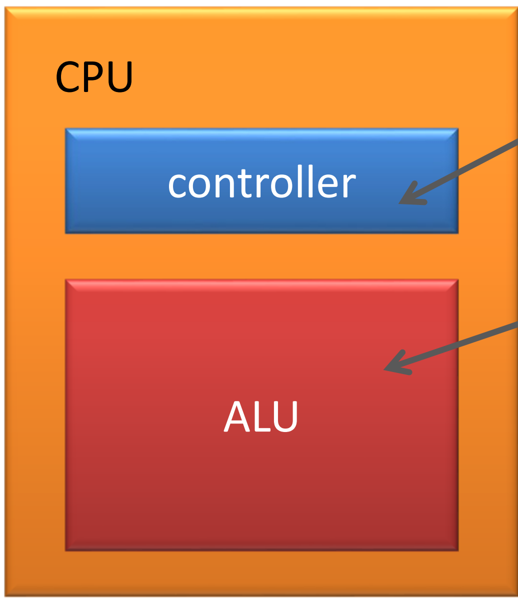
Typical computer scheme



CPU

Processor also **CPU** (**Central Processing Unit**) is an essential part of the computer; it is a very complex sequential circuit that **executes machine code** stored in the computer's operating memory. The machine code consists of individual machine instructions of a computer programs loaded into the operating memory.

www.wikipedia.org



reads machine instructions and data and prepares their processing in ALU

ALU (arithmetic and logic unit), performs arithmetic operations, evaluates conditions

sequential processing of machine instructions is controlled by an internal clock cycle

How does the CPU (ALU) work with numeric values?

Integer numbers

The smallest unit of information in digital technology is one **bit**. Bits are formed into words. The smallest word is **byte** which contains 8 bits.

One byte can describe integers ranging from 0 to 255.

128	64	32	16	8	4	2	1		
0	1	0	1	0	1	1	1	=	87

Signed integers can also be expressed. In this case, one bit is reserved for the sign, the remaining bits for the number. There are several implementation options. Intel architecture uses **two's complement**, which leads to range from -128 to 127.

128	64	32	16	8	4	2	1		
0	1	1	1	1	1	1	1	=	127
0	1	0	1	0	1	1	1	=	87
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	-1
1	0	1	0	1	0	0	1	=	-87
1	0	0	0	0	0	0	0	=	-128

bit reserved for sign

Integer numbers, II

Integers with greater dynamic range can be expressed using larger words typically composed of four bytes (32 bit word) or eight bytes (64 bit word).

32-bit unsigned integer:	0 to 4.294.967.295
32-bit signed integer:	−2.147.483.648 to 2.147.483.647
64-bit unsigned integer:	0 to 18.446.744.073.709.551.615
64-bit signed integer:	−9.223.372.036.854.775.808 to 9.223.372.036.854.775.807

When working with integers it is necessary to take into account that **you cannot express an arbitrarily large number** and the option of **underflow** or **overflow** of values must be consistently avoided.

Real numbers

Real numbers are expressed in the following format (**floating point** format):

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

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 are the bits of the m

In digital technology, real numbers are most often expressed in a format defined by a standard **IEEE 754**.

type	width	mantisa	exponent
single precision	32	23	8
double precision	64	52	11

Real numbers, II

Type	Range	Accuracy
single precision	$\pm 1.18\times10^{-38}$ to $\pm 3.4 \times 10^{38}$	approximately 7 decimal places
double precision	$\pm 2.23\times10^{-308}$ to $\pm 1.80\times10^{308}$	approximately 15 decimal places

By a special combination of value of the mantissa and the exponent, following **special values** can be expressed:

0 positive zero
-0 negative zero
NaN not a number, e.g., the result of division by zero
+Inf positive infinity (number is too large to express)
-Inf negative infinity (number is too large to express)

When working with real numbers, it is necessary to pay attention to propagation of **rounding errors**, in logical comparisons, it is **not appropriate** to use operators **equals** and **does not equal**, except for the zero comparison situation.

Exercise 1

1. Variable of type **signed char** (signed byte) contains the number 127. What value will the variable have if we increase it by one?
2. Variable of type **unsigned char** (unsigned byte) contains the number 88. How does the numeric value change if the bit representation of the number is shifted one position to the right or left? What is the mathematical meaning of the operation.
3. What will be the result of the sum of real numbers represented in double precision and having values:

   $0,1346978.10^{-12}$

   $1,2312657.10^{6}$

4. What is big-endian and little-endian? Indicate architecture that use given type of endianity. What effect does the endianity have on transfer binary data?

Joint exercise: conversion of numerical values from decimal to binary and hexadecimal

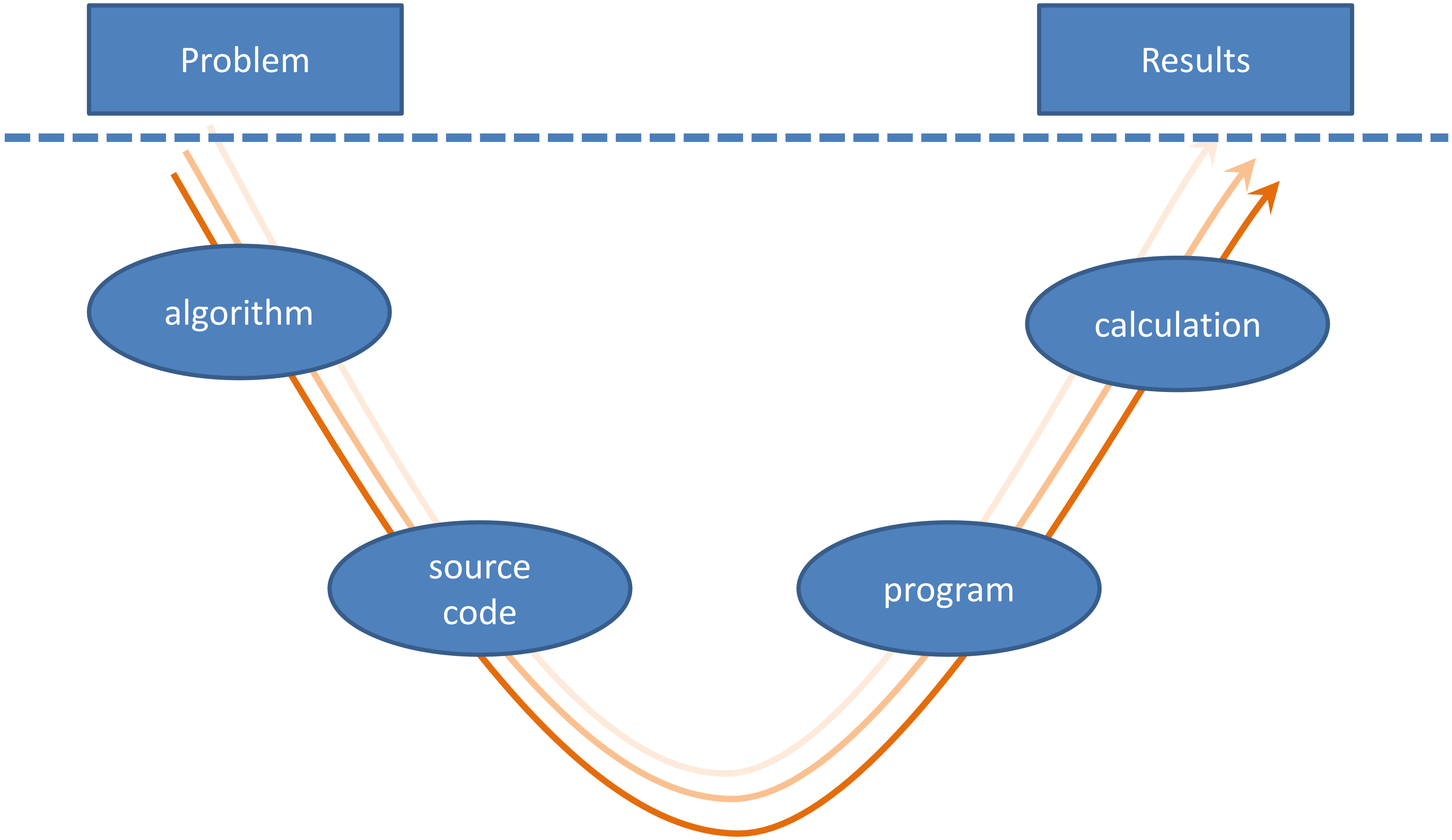
Conclusions

- CPUs (or other computing units, e.g., GPGPUs) operate with some numerical precision.
- Errors (numerical errors) can occur in numerical calculations, which can lead to incorrect results (predictions).
- When designing computer programs, it is therefore necessary to use such algorithms that are either not sensitive to rounding errors or significantly reduce their effect.

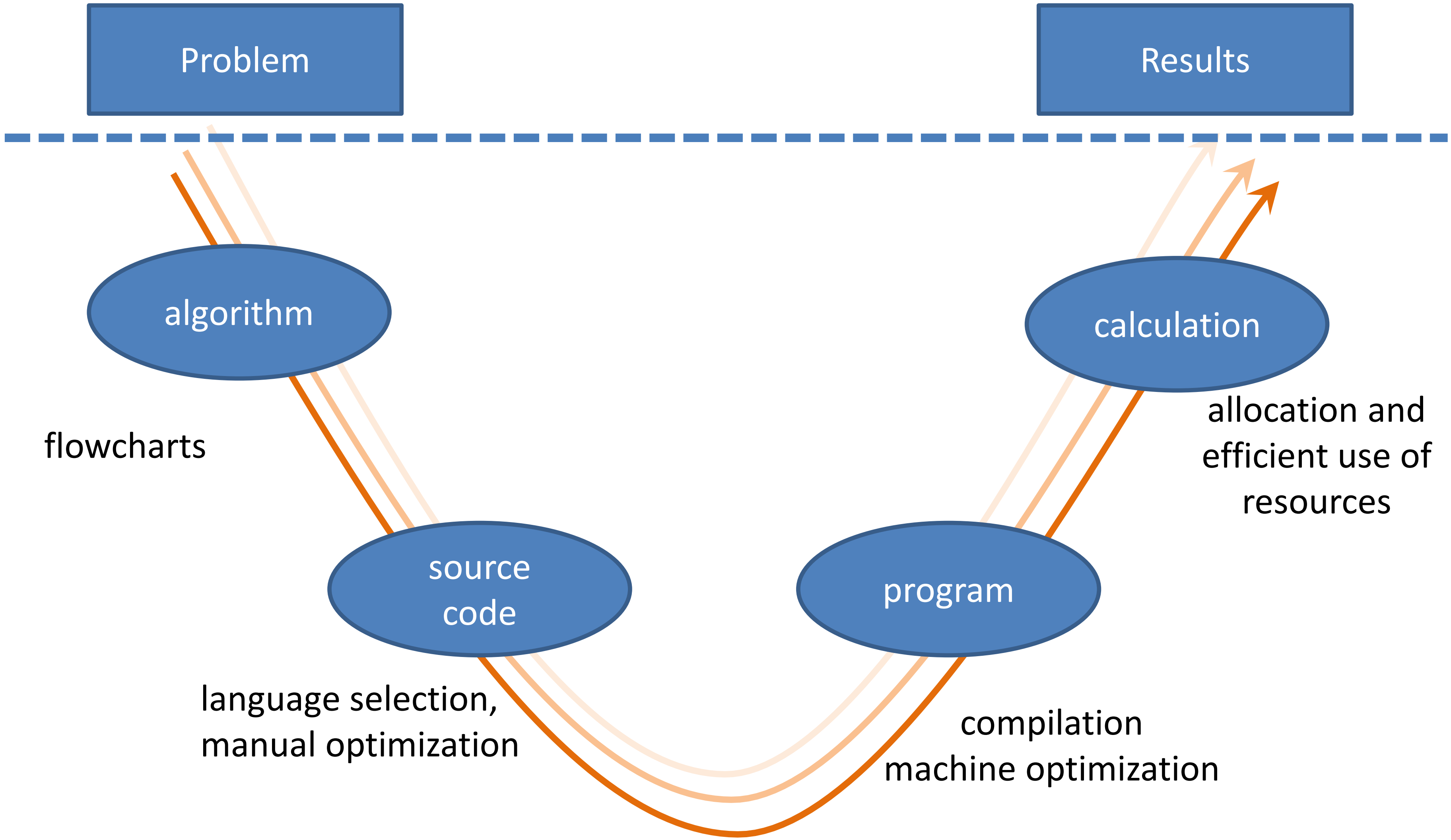
From problem to results

From problem to results ...

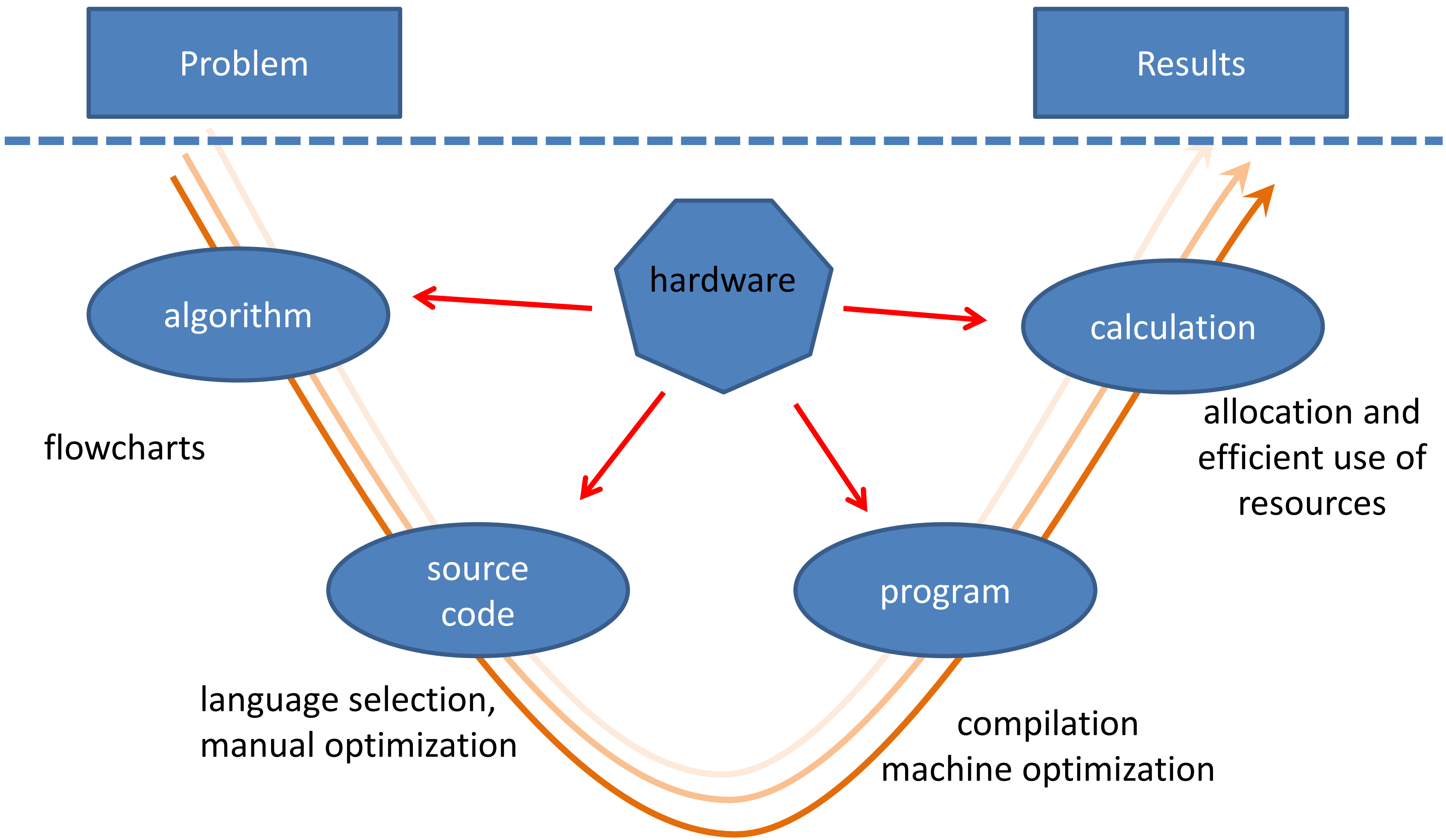
From problem to results ...



From problem to results ...

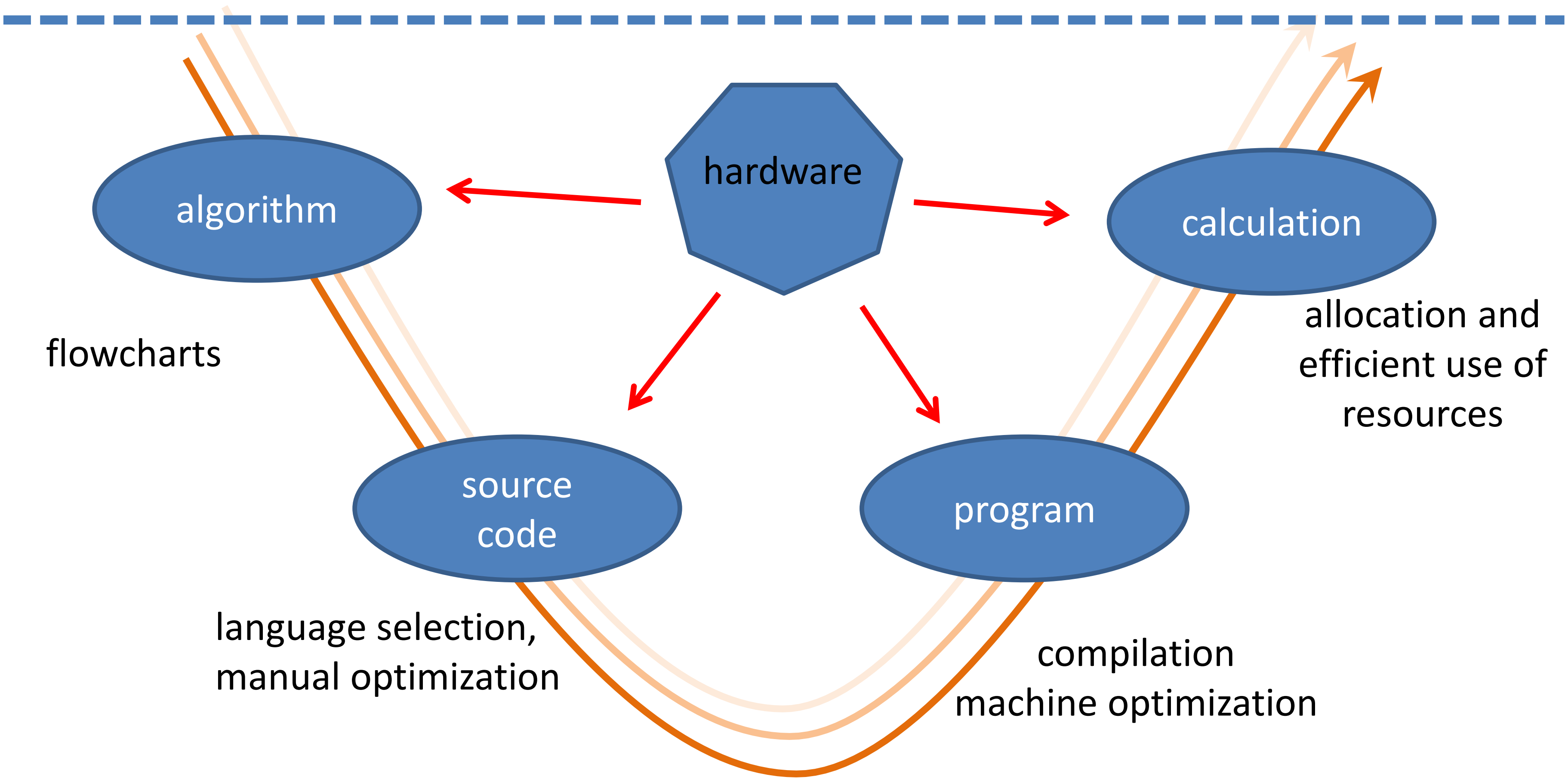


From problem to result ...



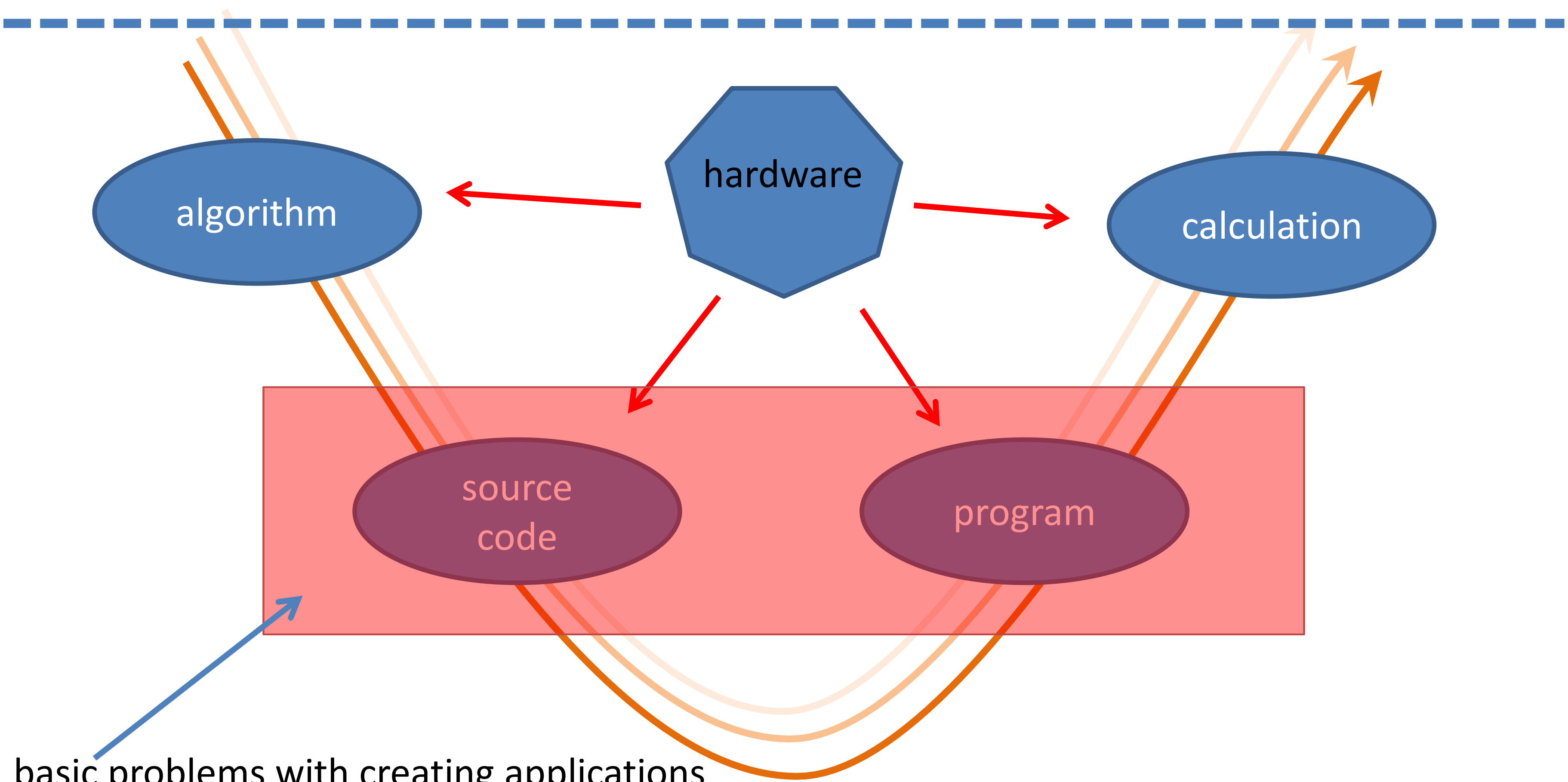
From problem to result ...

When solving problems using computer technology (supercomputers), it is necessary to **comprehensively evaluate** several aspects, including used hardware and its architecture.



Covered topics ...

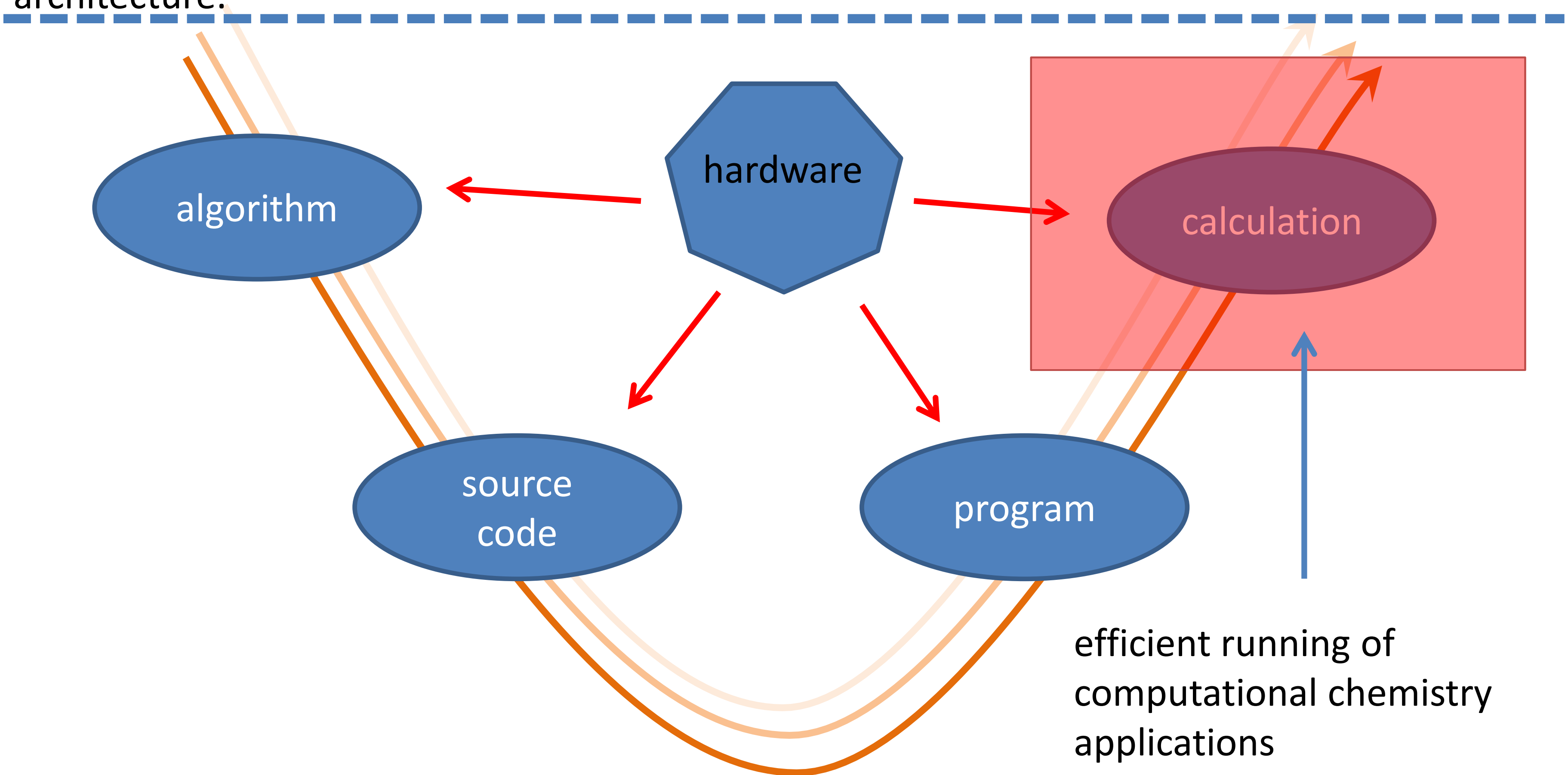
When solving problems using computer technology (supercomputers), it is necessary to **comprehensively evaluate** several aspects, including used hardware and its architecture.



basic problems with creating applications
for demanding calculations, parallelization

Topics covered ...

When solving problems using computer technology (supercomputers) it is necessary **comprehensively evaluate** a number of aspects, including the hardware used and its architecture.



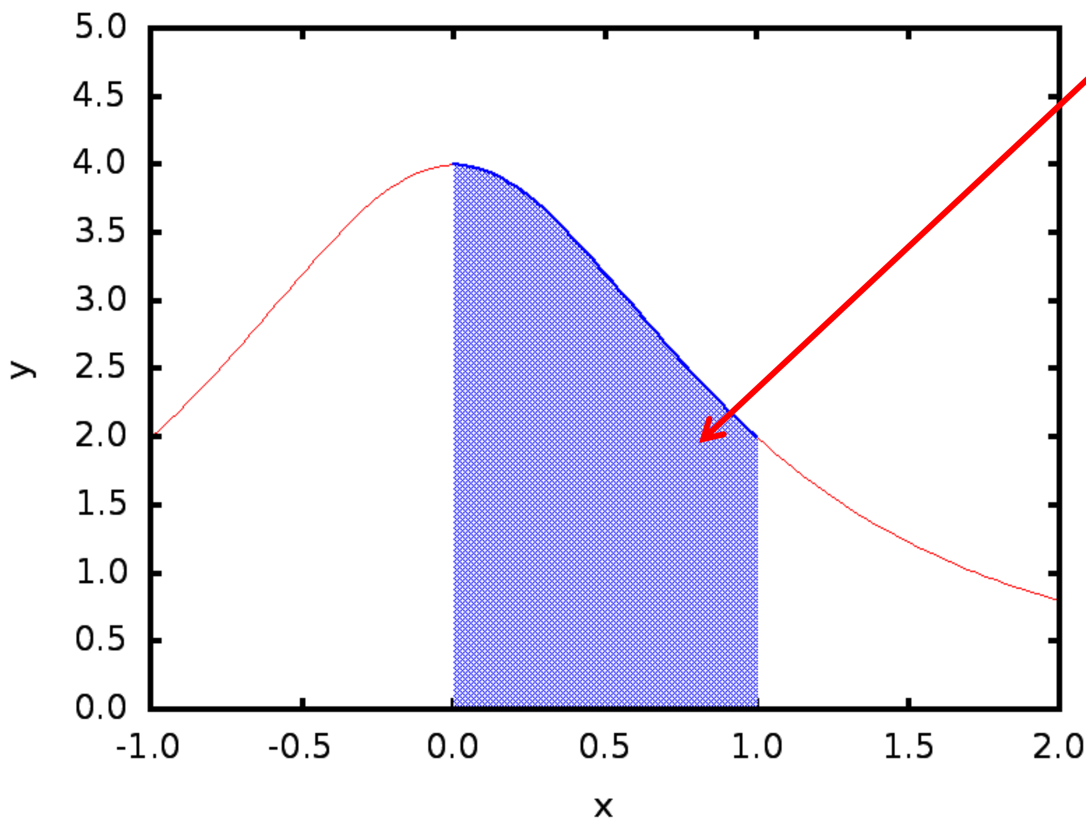
efficient running of computational chemistry applications (MetaCentrum, small clusters)

Numeric integration

Exercise LIII.3

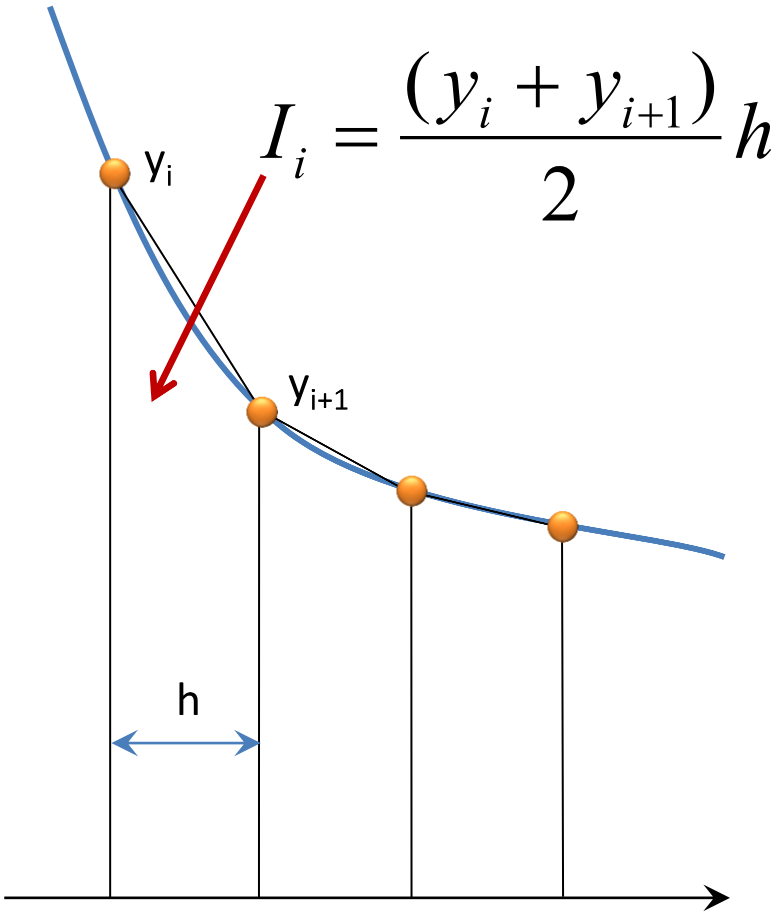
1. Write a program that calculates a certain integral below. Use the trapezoidal method for integration.

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



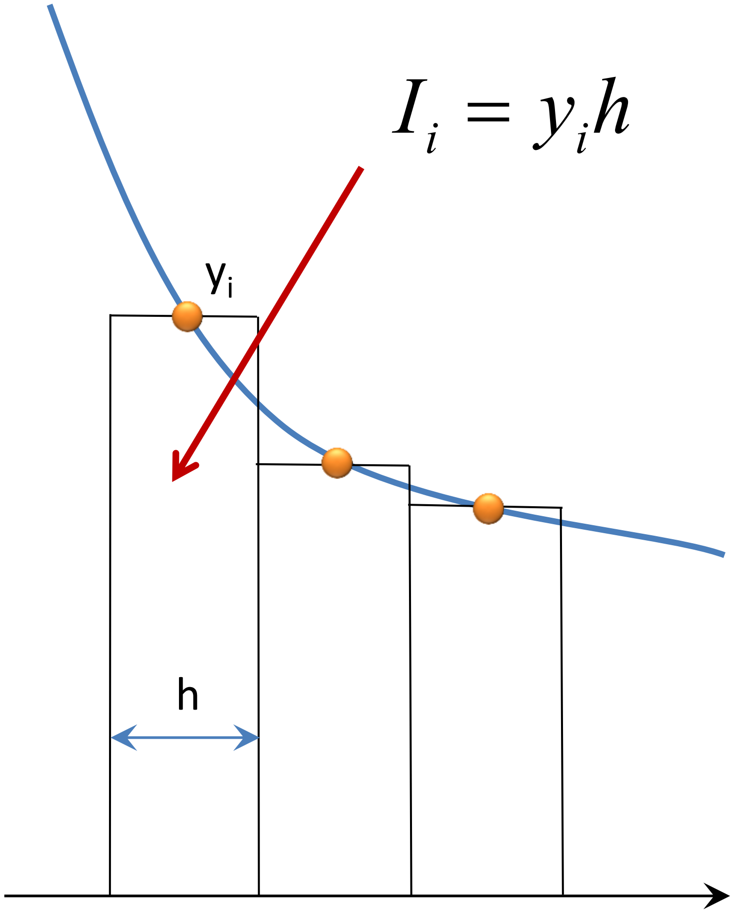
a certain integral is the area under the curve in the range of integration limits

Trapezoidal vs rectangular method



trapezoidal method

numerically more accurate method



rectangular method

numerically less accurate method
easier implementation and parallelization

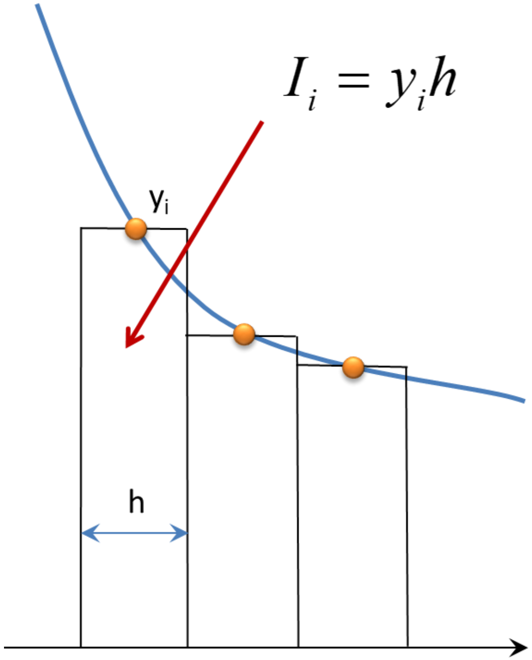
Sequential implementation

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

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

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

end program integral
```

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

rectangular method

$$I_i = y_i h$$



Exercise 2

Source codes:
/home/kulhanek/Documents/C2115/code/integral/single

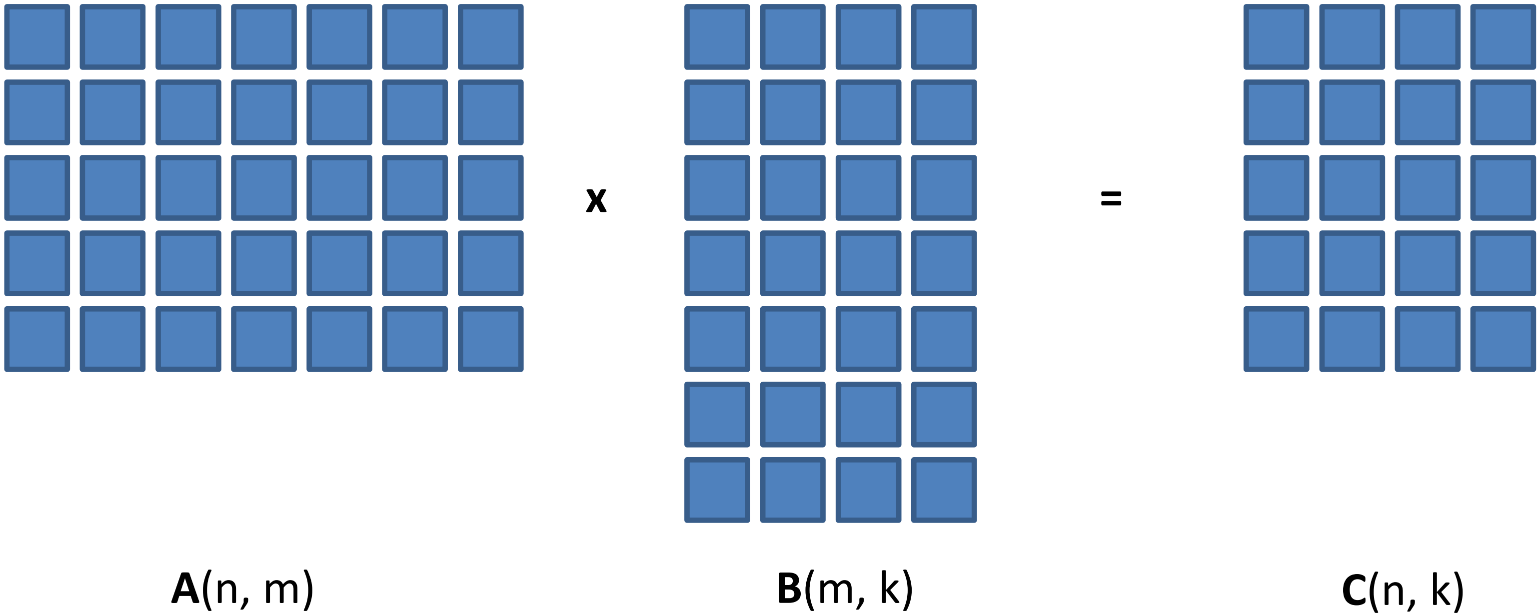
1. Compile the program **integral.f90** with optimization **-O3**
2. Measure application run time required to integrate the function. Use the program **/usr/bi /time** to measure the time.
3. What is the value of the integral equal to?
4. What effect does the value of the variable "n" (i.e., size h) have on the accuracy of the calculation? Use programs **integral-errors_sp.f90** and **integral-errors_dp.f90** to assess. Briefly discuss obtained results.

Matrix multiplication

Content

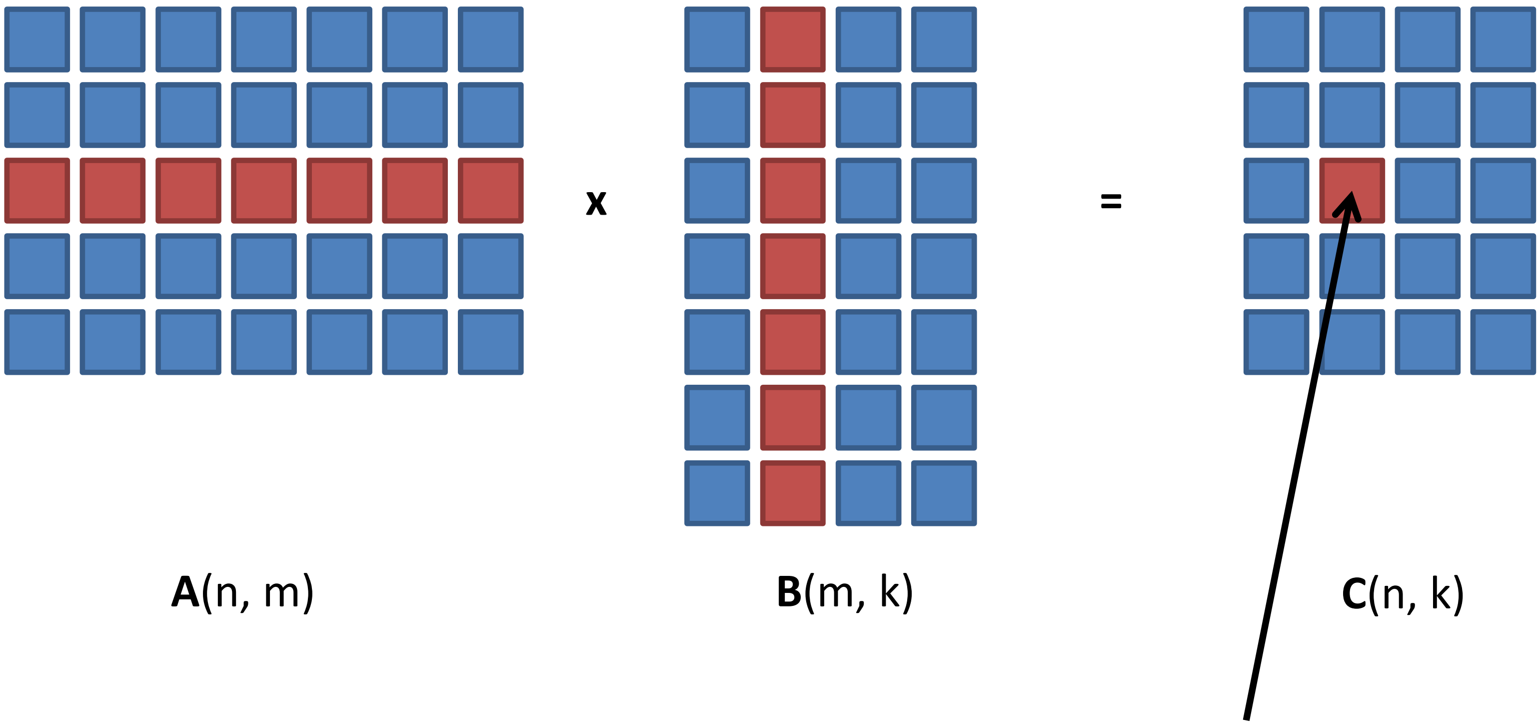
- **Matrix multiplication**
  implementation, complexity, computing power, exercises
- **Explanation of the obtained results**
  computer architecture and its bottlenecks
- **Use of optimized libraries**
  BLAS, LAPACK, LINPACK, comparison of results, exercises

Matrix multiplication



Use:

- finding eigenvalues and vectors of square matrices (quantum chemistry)
- solution of a system of linear equations (QSAR, QSPR)
- transformations (displacement, rotation, scaling - display and graphics)



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

Element of the resulting matrix **C** is the scalar product of the vectors formed by the line *i* of **A** matrix and column *j* of **B** matrix

Matrix mult., implementation

```
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

Assuming that matrices **A** and **B** are square with dimensions NxN:

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

```
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
```

In computer technology, computing power is assed via **FLOPS (FLoating-point Operations Per Second)** value, which expresses how many floating point operations a given device performs per second.

Results

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

Key:

N - dimension of the matrix

NR - number of repetitions

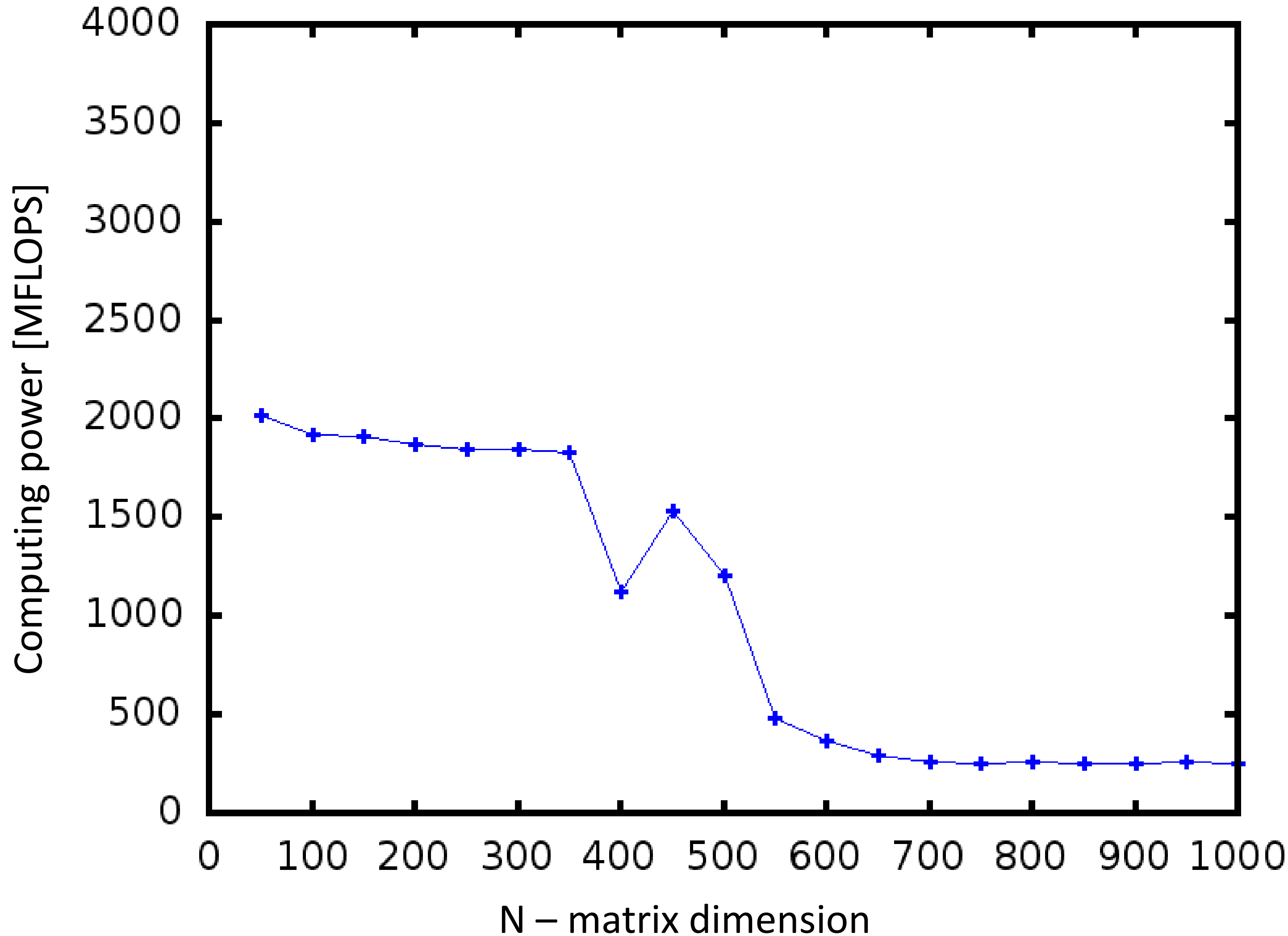
NOPs - number of operations in FP

Time - execution time in s

MFLOPS - computing power

Results

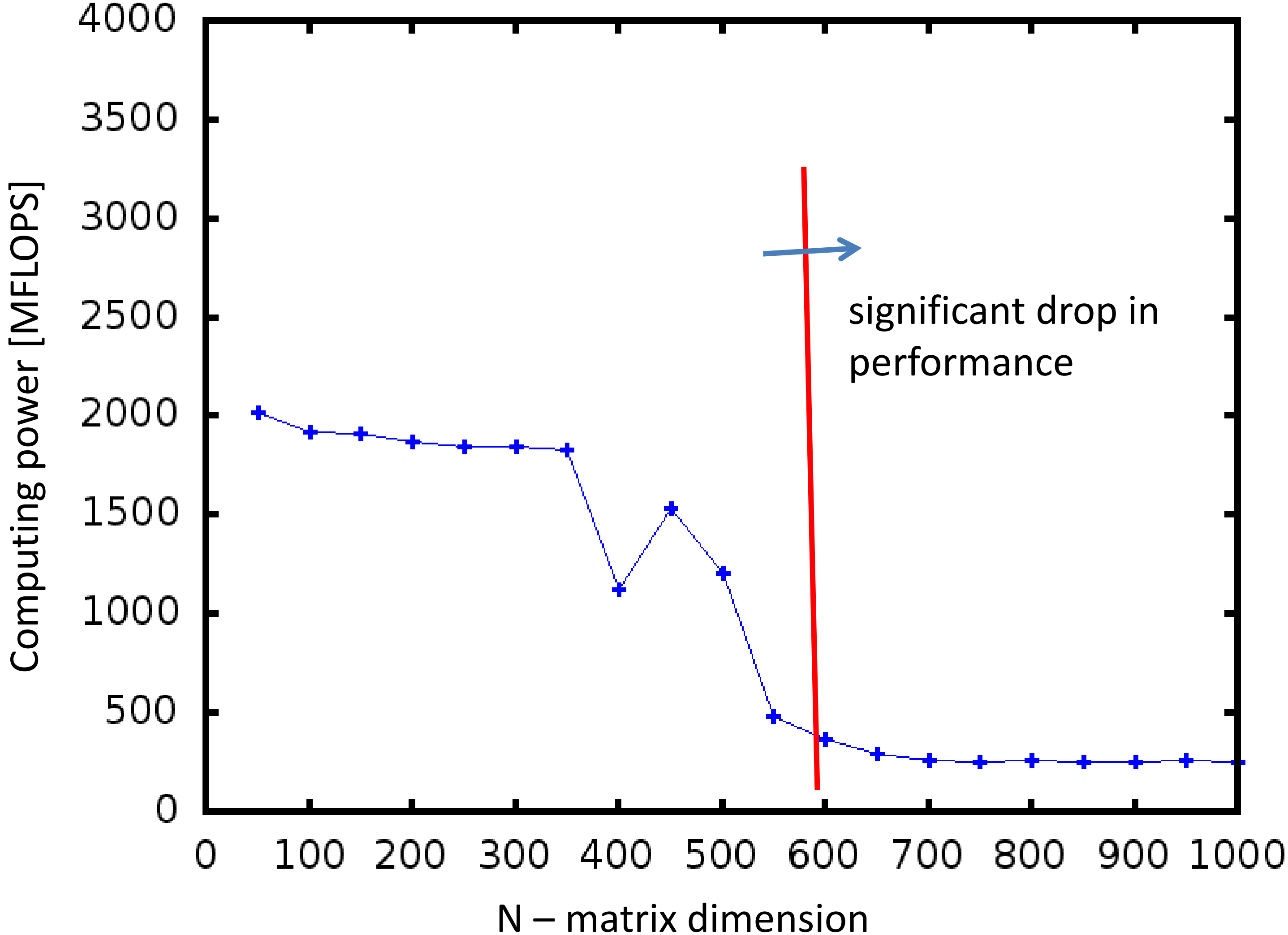
wolf21



Results

wolf21

wolf21



Exercise 3

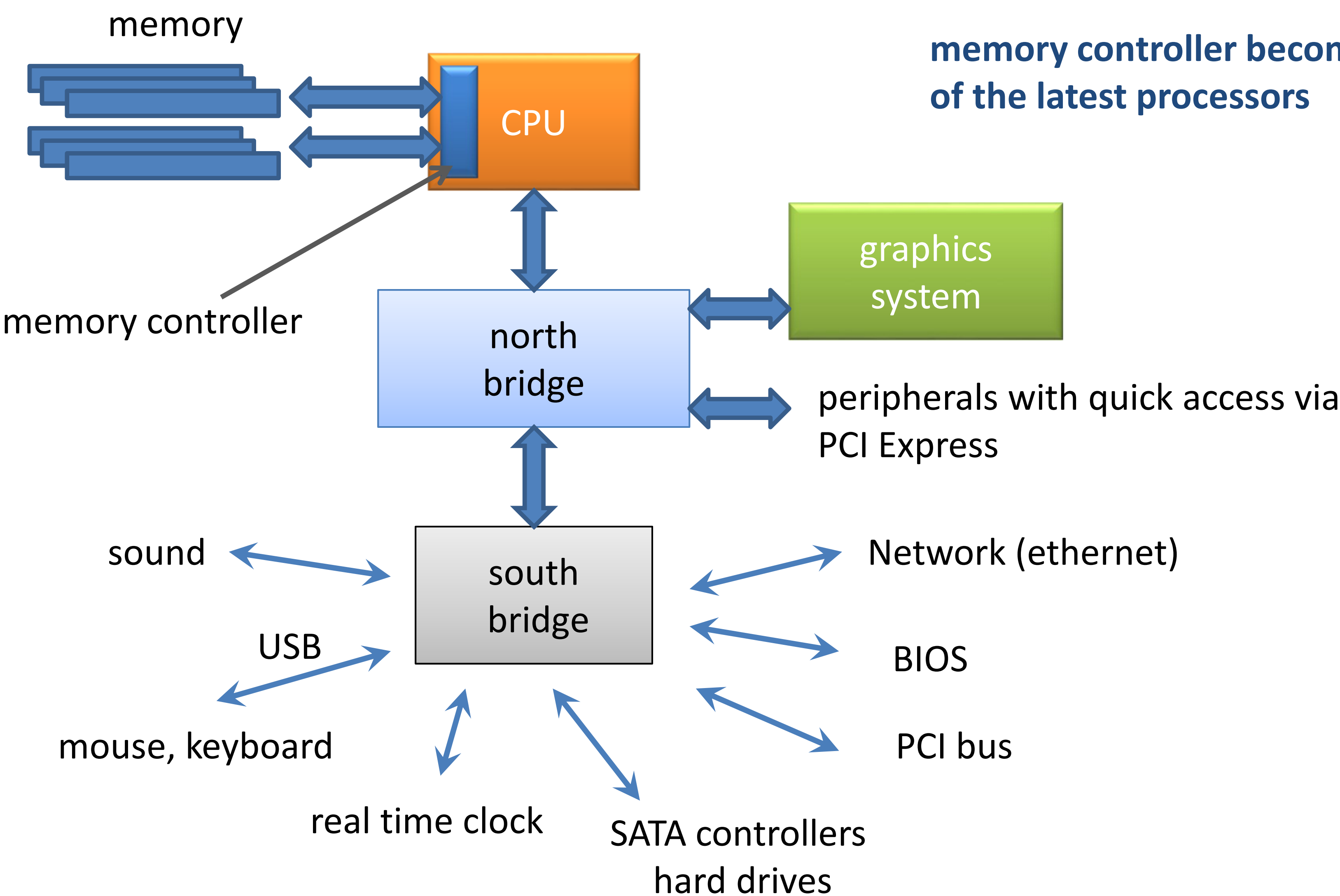
Source codes:
/home/kulhanek/Documents/C2115/code/matrix

1. Compile the program **mult_mat_naive_dp.f90** with **gfortran** compiler, use **-O3** optimization.
2. Run the program and display the obtained dependence of the computational power depending on the size of the matrix in the form of a graph (interactive mode gnuplot).
3. Compare the results for the optimization levels **-O3 and -O0**. Display the obtained dependencies in one graph. Insert the graph into the protocol. Be sure to specify the CPU type (command lscpu).
4. Discuss obtained results.

Matrix multiplication

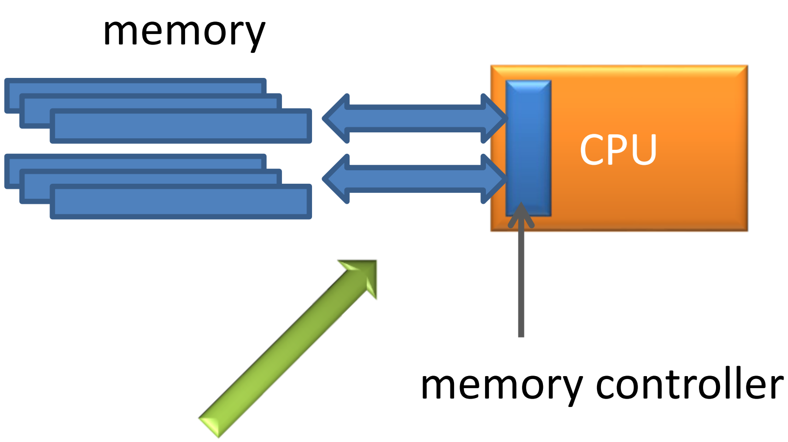
vs
Architecture of computers

Architecture, overal view



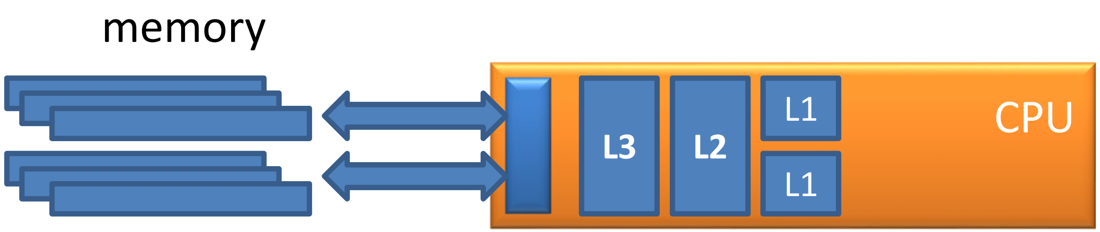
memory controller becomes part of the latest processors

Architecture, bottleneck



Bottleneck: data transfer rate between memory and CPU is slower than the speed at which the CPU is able to process data

Hierarchical model of memory

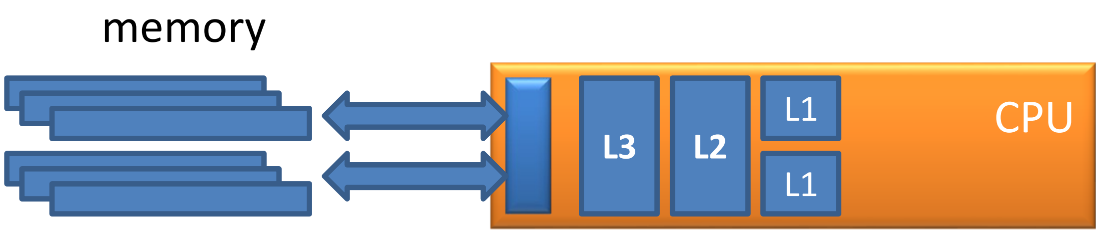


fast cache, different levels with different access speeds

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

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

Hierarchical model of memory



fast cache, different levels with different access speeds

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

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

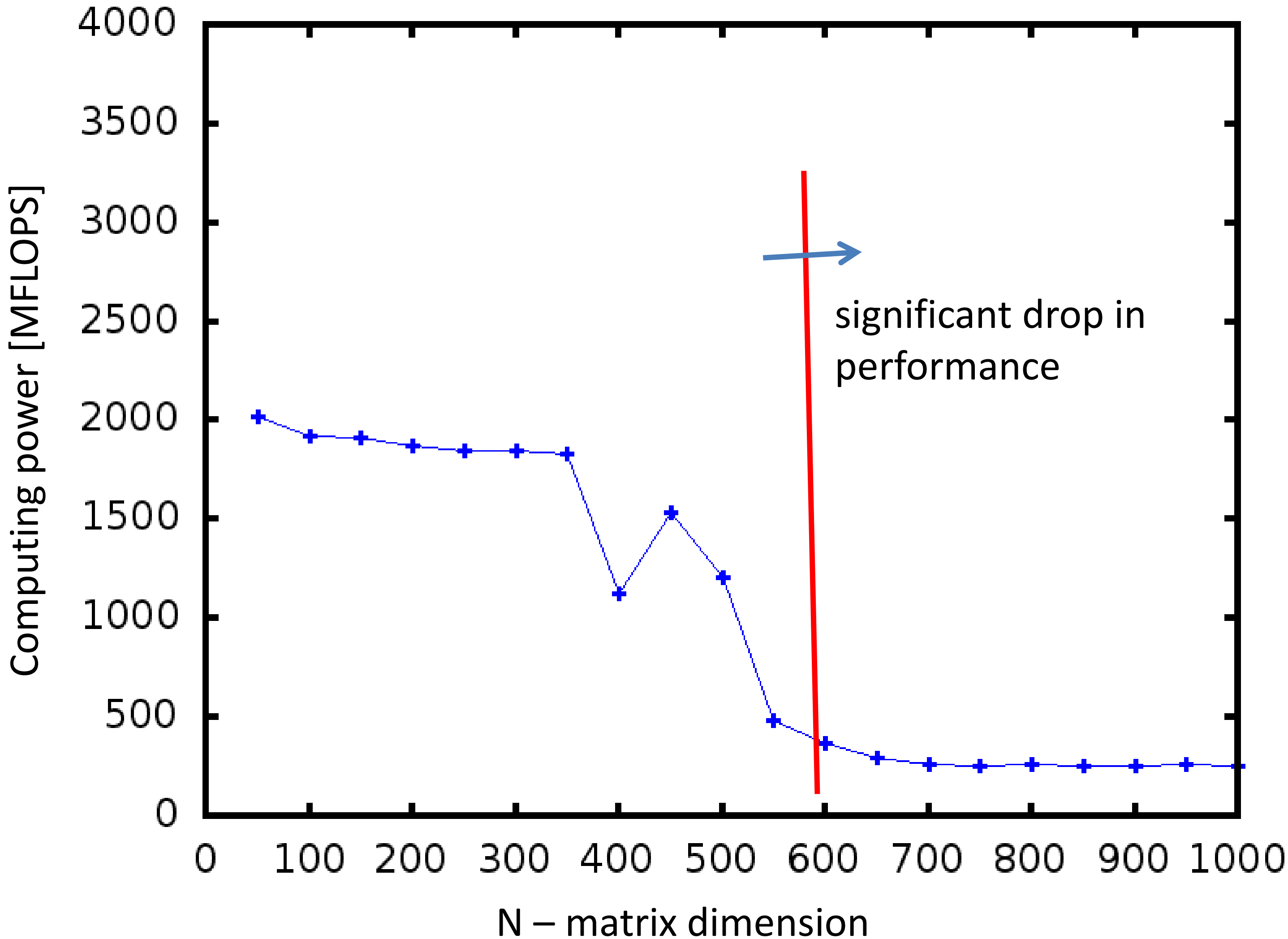
When the size of problem exceeds size of CPU cache, data transfer rate between physical memory and CPU becomes the **speed limiting step**.

N=600
600x600x3x8 = 8437 kB

A, B, C double precision

Results

wolf21



Libraries for linear algebra

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, ownproblems, 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 Schurfactorizations 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 BLAS and LAPACK libraries

- optimized by the hardware vendor
- 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 FFT libraries (Fast Fourier Transform)

- optimized by the hardware vendor
- 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 via 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 interface of BLAS library does not contain information about argument type.
Programmer must enter all arguments in the correct order and type!!!!

Compilation:

```
$ gfortran -O3 mult_mat_blas_dp.f90 -o mult_mat_blas_dp -lblas
```

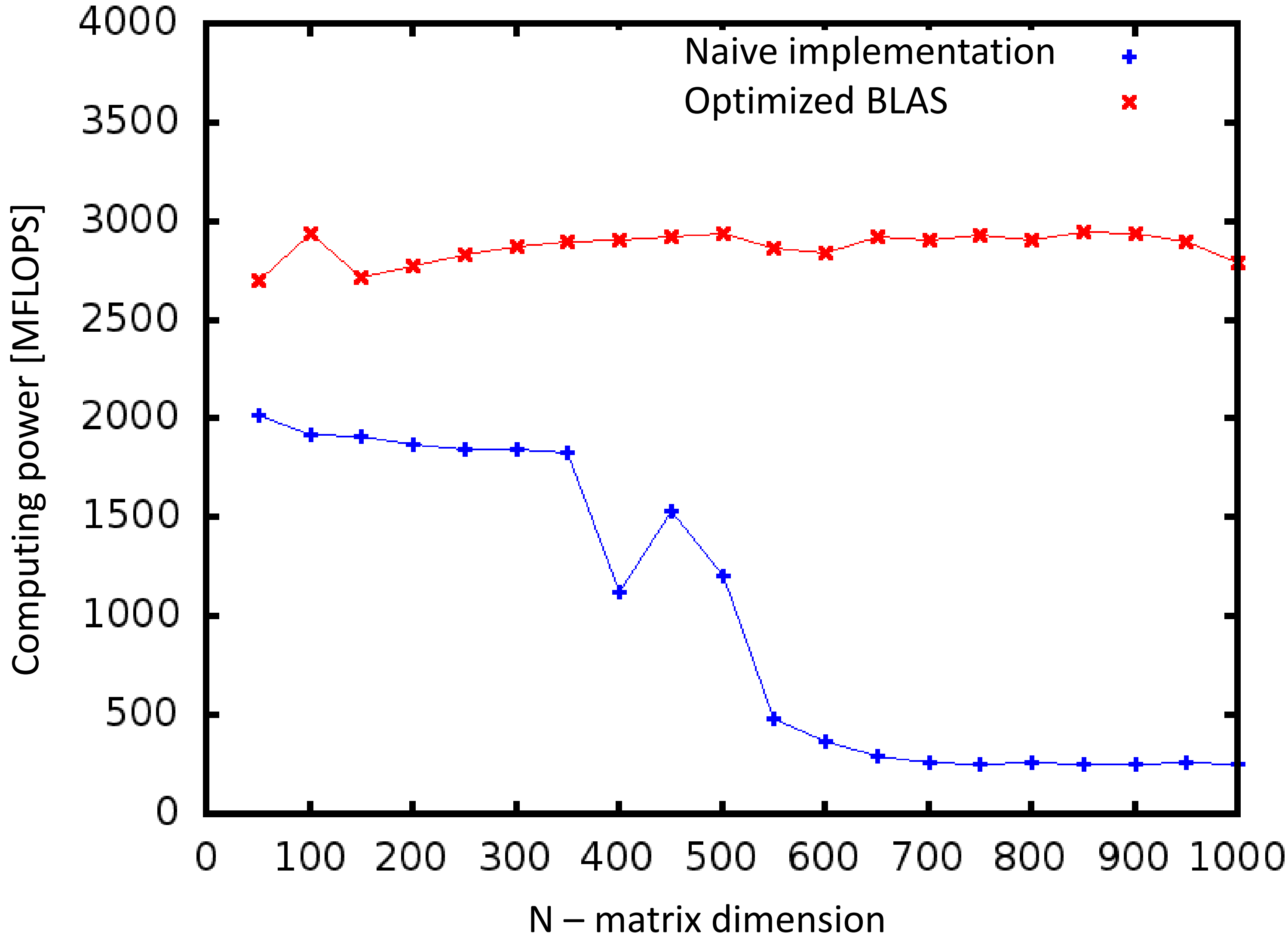
Matrix multiplication via BLAS - dp

```
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
```

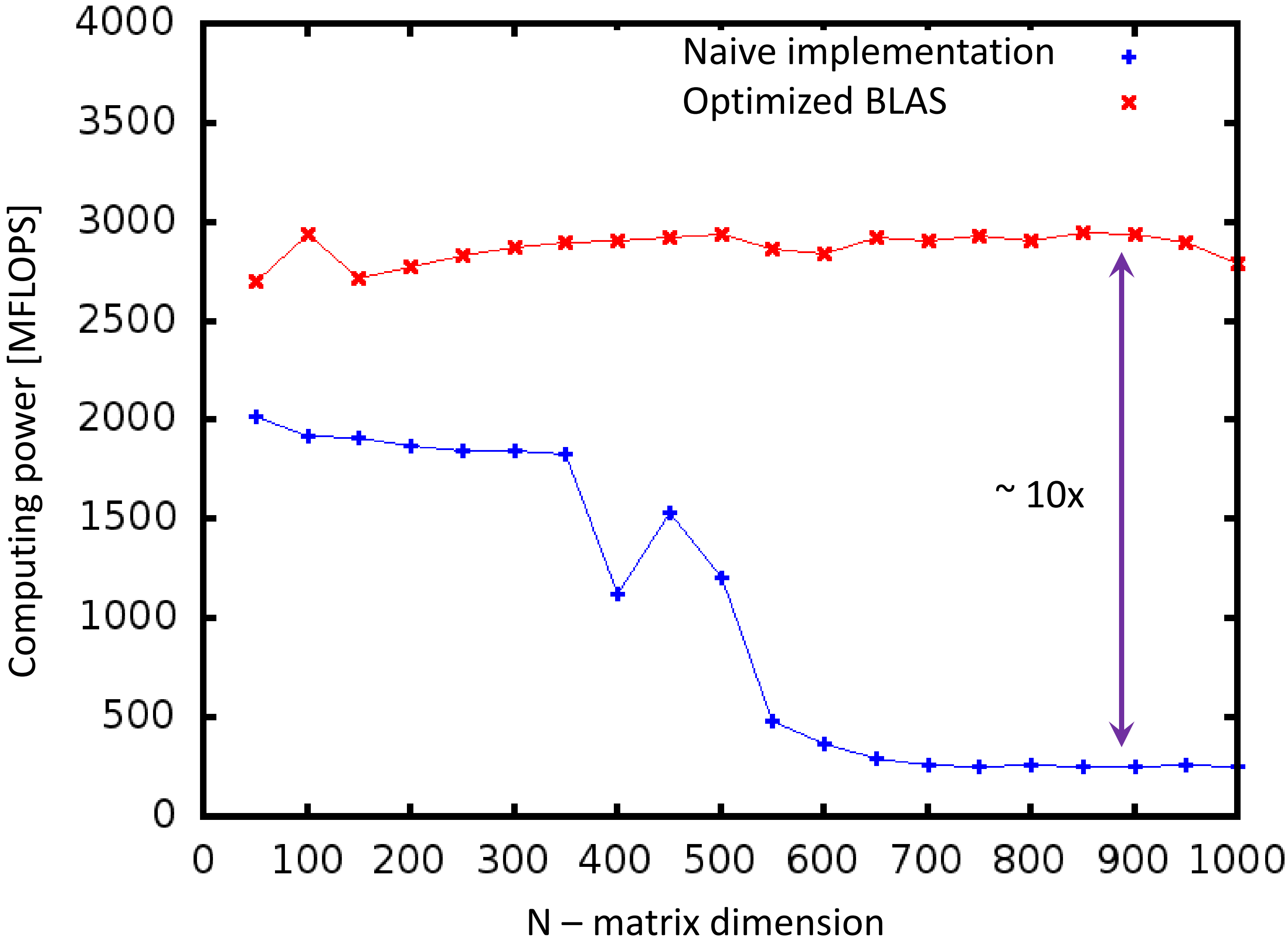
Compilation:

```
$ gfortran -O3 mult_mat_blas_sp.f90 -O mult_mat_blas_sp -lblas
```

Naive *vs* optimized solution



Naive *vs* optimized solution



Exercise 4

Source codes:
/home/kulhanek/Documents/C2115/code/matrix

1. Compile the program **mult_mat_blas_dp.f90** with **gfortran** compiler, use **-O3** optimization.
2. Run the program and display the obtained dependence of computing power depending on the size of the matrix in the form of a graph (interactive mode gnuplot).
3. Determine the computational power for the optimization levels **-O3** and **-O0**. Display the obtained dependencies in one graph. Insert the graph into the protocol. Be sure to specify the CPU type (command lscpu).
4. Compare computing power for **native** and **blas** approaches in the optimized version (option -O3). Display the obtained dependencies in one graph. Insert the graph into the protocol. Be sure to specify the CPU type (command lscpu).
5. Discuss obtained results.

Compilation:

```
$ gfortran -O3 mult_mat_blas_dp.f90 -o mult_mat_blas_dp -lblas
```