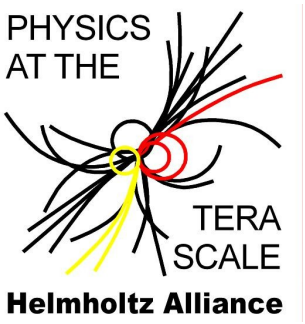
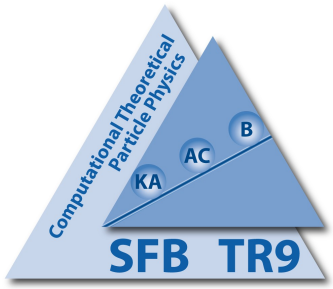
Efficient Computing in Particle Physics

Peter Uwer





What do we mean by efficiency ?

Getting the result as fast as possible !

Different kinds of efficiency

- Different algorithms may be more or less efficient
- Different implementations of the same algorithm may be more or less efficient
- Different tools may be more or less efficient for specific tasks

Example: Algorithms for sorting

Bubble sort:

1. Step through the list, compare neighboring elements,
2. Swap them if they are in the wrong order,
3. Repeat until no swapping occurs

} Algorithm

First Pass:

(**5 1** 4 2 8) → (**1 5** 4 2 8), Swap since 5 > 1
(1 **5 4** 2 8) → (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) → (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) → (1 4 2 **5 8**),

Second Pass:

(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Third Pass:

(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

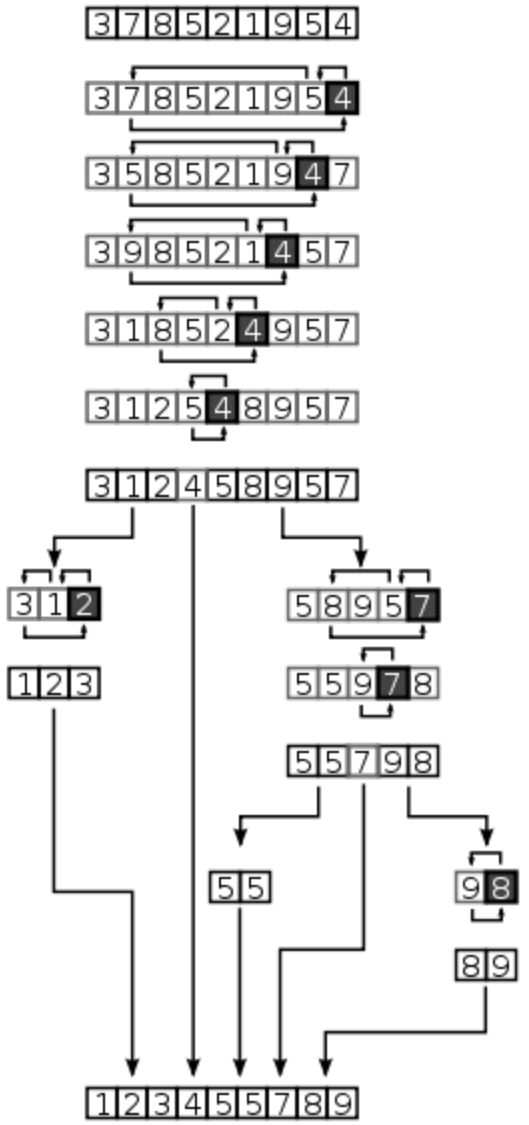
What is needed by the sorting algorithm ?

→ Compare items (is_greater(a,b))
→ Swap items

Example: Algorithms for sorting

Quick sort:

1. Pick an element x from the list.
2. Reorder the list so that all elements which are less than x come before x and all elements greater than x come after it
3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.



Example: Algorithms for sorting

Many other algorithms available:

- Insertion sort
- Shell sort
- Merge sort
- Heap sort
- Bucket sort
- Radix sort
- Distribution sort
- Shuffle sort
- …

[see for example D. Knuth's book for details]

What is the difference ?

Their efficiency !

Example: Algorithms for sorting

Bubble sort:

1. Step through the list, compare neighboring elements,
2. Swap them if they are in the wrong order,
3. Repeat until no swapping occurs

First Pass:
(**5 1** 4 2 8) → (**1 5** 4 2 8), Swap since 5 > 1
(1 **5 4** 2 8) → (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) → (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) → (1 4 2 **5 8**),

Second Pass:
(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Third Pass:
(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

If list has n objects we need n passes with every pass having n comparisons,

→ n^2 operations to sort a list of n elements

→ "complexity is of order n^2" $(O(n^2))$

Big O notation

Asymptotic behaviour for large integer n

$$O(f(n))$$

How to read?

$$O(f(n)) = x_n \leq M|f(n)| \text{ for } n > n_0$$

Example: Harmonic sums for large n

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

$$H_n = \ln(n) + \gamma + O\left(\frac{1}{n}\right) \quad \Rightarrow \quad |H_n - \ln(n) - \gamma| \leq \frac{M}{n} \text{ for } n > n_0$$

Computational cost

$O(1)$	cheap
$O(\log(n))$	fairly cheap
$O(n)$	expensive
$O(n\log(n))$	expensive
$O(n^2)$	very expensive
$O(e^n)$	unaffordable
$O(n!)$	unaffordable

Complexity of different sorting algorithms

Name	Average	Worst	Memory
Bubble sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Cocktail sort	—	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Comb sort	—	—	$\mathcal{O}(1)$
Gnome sort	—	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Selection sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Insertion sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Shell sort	—	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(1)$
Binary tree sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
Library sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
In-place merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$
Heapsort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$
Smoothsort	—	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$
Quicksort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\log n)$
Introsort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$
Patience sorting	—	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Strand sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$

The complexity of the n-gluon amplitude

"Number of diagrams" [Fabio Maltoni]

n	full Amp	partial Amp	BG
4	4	3	3
5	25	10	10
6	220	36	35
7	2485	133	70
8	34300	501	126
9	559405	1991	210
10	10525900	7335	330
11	224449225	28199	495
12	5348843500	108280	715

$$(2n)! \qquad \underbrace{3.8^n \qquad n^4}_{\text{color-ordered}}$$

Importance of Big O

Note:

For many application n is still rather small

→ Asymptotic behaviour may be irrelevant

For small n an algorithm which is formal more complex may still be faster

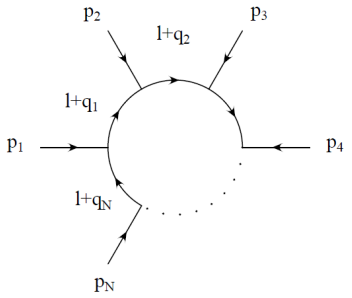
→ implementation, computing architecture

Implementation may switch between different algorithms

In many cases we are already happy if we have ONE algorithm

Example: One-loop tensor reduction

$$I_N^{\mu_1\mu_2\cdots\mu_m}(D;\{q_i\},\{1\}) = \int \frac{d^D\ell}{i\pi^{D/2}} \frac{\ell^{\mu_1}\ell^{\mu_2}\cdots\ell^{\mu_m}}{d_1 d_2 \cdots d_N}$$



$$d_i = (\ell + q_i)^2 + i0.$$

q_i external momenta

The algorithm

$$I_N^{\mu_1\mu_2\cdots\mu_m}(D;\{q_i\},\{1\}) = \sum_{\lambda,x_1,x_2,\ldots,x_N} \delta_{(2\lambda+\sum_i x_i - m)} \left(-\frac{1}{2}\right)^{\lambda} x_1! x_2! \cdots x_N!$$

$$\times \left\{ g^{\lambda} q_1^{x_1} q_2^{x_2} \cdots q_N^{x_N} \right\}^{\mu_1\mu_2\cdots\mu_m}$$

Lorentz structures

$$\times\ I_N(D + 2(m-\lambda);\{q_i\},\{1+x_i\}).$$

Scalar integrals

[Davydychev]

$$I_N(D;\{q_i\},\{\nu_i\}) = \int \frac{d^D\ell}{i\pi^{D/2}} \frac{1}{d_1^{\nu_1} d_2^{\nu_2} \cdots d_N^{\nu_N}}$$

Tensor reduction using integration-by-parts

[Chetyrkin, Kataev, Tkachov]

$$0 = \int \frac{d^d\ell}{(2\pi)^d} \frac{\partial}{\partial \ell^\mu} \left(\frac{\sum_i^n y_i (\ell + q_i)}{((\ell+q_1)^2 - m_1^2)^{\nu_1} \ldots ((\ell+q_n)^2 - m_n^2)^{\nu_n}} \right)$$

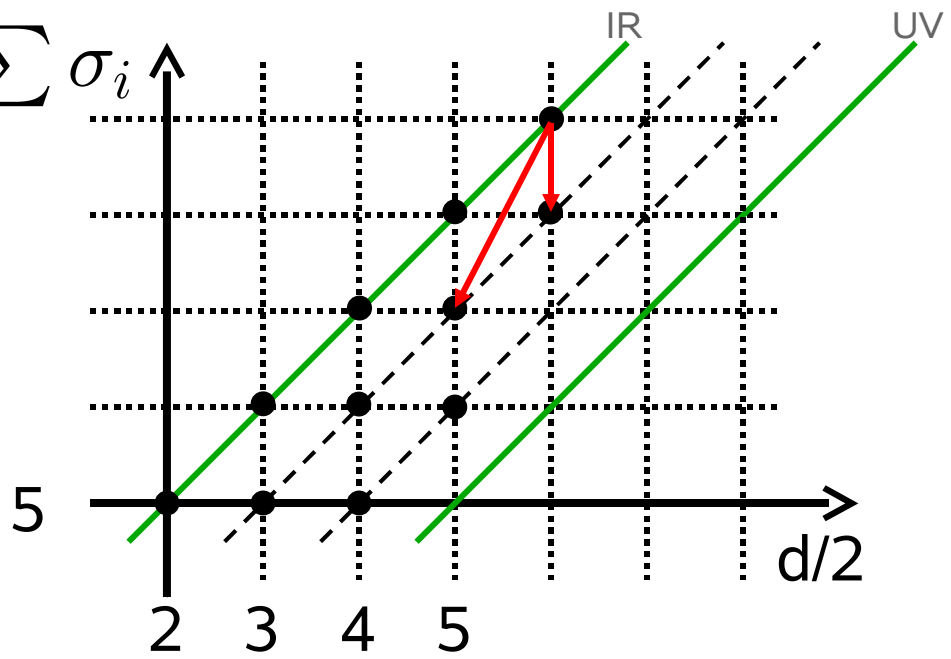
[Duplancic, Nizic 03, Giele, Glover 04]

$$(\nu_k - 1) I(d; \{\nu_l\}) =$$

$$\sum_{i=1}^{n} S^{-1}_{ki} I(d-2; \{\nu_l - \delta_{li} - \delta_{lk}\}) - b_k (d - \sigma) I(d, \{\nu_l - \delta_{lk}\})$$

$$\sigma = \sum \sigma_i$$

$$S_{ki} = (q_i - q_j)^2$$



5-point

Tensor reduction using integration-by-parts

Integrals with raised powers of propagators are reduced recursively to integrals with lower powers:

```
IntType Int(int d,int n1,int n2,int n3,int n4,int n5,
                    int n6,...){
     .
     .
     .
      return( + b[1] * Int(d,n1-1,n2,n3,n4,n5,n6,...)
              + b[2] * Int(d,n1,n2-1,n3,n4,n5,n6,...)
              + b[3] * Int(d,n1,n2,n3-1,n4,n5,n6,...)
              + b[4] * Int(d,n1,n2,n3,n4-1,n5,n6,...)
              + b[5] * Int(d,n1,n2,n3,n4,n5-1,n6,...)
              + b[6] * Int(d,n1,n2,n3,n4,n5,n6-1,...)
              );
}
```

The implementation

Easy to implement, but some integrals are evaluated several times → poor performance

Tensor reduction using integration-by-parts

Need to store integrals already known in a data base

→ **use cache**

For complicated topologies large improvement in performance, when cache is used

→ difference in runtime by large factor (~1000)

[numerical implementation used in tt+1-Jet production @ NLO]

Note:

- For testing the basic implementation, cache is not required
- Impressive speed up only due to poor behaviour of initial version

Efficiency of tools

Intrinsic limitations:

- Performance of CAS like Maple and Mathematica often degrades when large expressions are encountered
- Form delivers performance even when operating on large expressions

On the other hand:

- If for example factorization is important to keep your expressions small, Maple or Mathematica are probably better adopted to your problem

→ No general rule, the decisions depend on the problems and the algorithms

Choosing the right tools

- Know about strength and weakness of tools
- If possible extend capabilities to your own needs
  - Extension of Maple/Mathematica by C/C++/Fortran code
  - Extension of Form via pipe mechanism
- Be flexible in switching between tools [see Kouhei Hasegawa's talk]
  - Store result, use different tool, continue with original tool
- Do not blindly believe the common blurb

Comments on tuning

- First consider of buying a faster computer
- Get things work first, then consider tuning it
- Don’t expect too much from tuning the implementation
- Improving the algorithm is usually much more important
  - Improve on the complexity
  - Use algorithm more adopted to your tools
- Use tools more adopted to your algorithms

Note:

- If computing effort is distributed “equally” over several steps → difficult to tune
- First identify the “hot-spots” → profiling

What this lecture won't tell you

- How to develop efficient algorithms for your problem
- A full proofed method to tune the implementations of your algorithms
- What tools you should use

What I will try to tell you

- Some ideas to organize your work in an efficient way
- Some guidance in finding the right tools
- Encourage you to learn new techniques
- A short introduction into C++

In many cases you may not save time in the beginning, but:

- What do you like more? Do the same thing over an over again, or spent the time on learning something new to do it automatically and saving time in the future?
- Knowledge always pays off sooner or later…

…let's consider a real life problem…

A generic project…

- Genious idea
- Some calculations on paper
- Computer Algebra
- Numerical part
- Publication

→ Different steps may involve different tools
→ Some steps may relay on others
→ Need to redo certain things if we discover mistakes
→ Don’t redo things which were not affected

Organization of a project

Abstract formulation:

Rules and dependencies

Rules tell us how to create results from other results

Dependencies tell us which result depends on which input

Standard tool to treat this type of problem:

make

[see also Thomas Hahn's lecture]

Example

The master equation:

$$f(n,k,x) = \binom{n}{k} \frac{(x+2)(x+3)}{(x+5)^3}, \quad (1)$$

$$b = \cos(x), \quad (2)$$

$$w = b \times f(n,k,x). \quad (3)$$

Using

$$n = 5, \quad (4)$$

$$k = 2. \quad (5)$$

A very sopisticated evaluation on a $2,000,000 computer yields Figure 1.

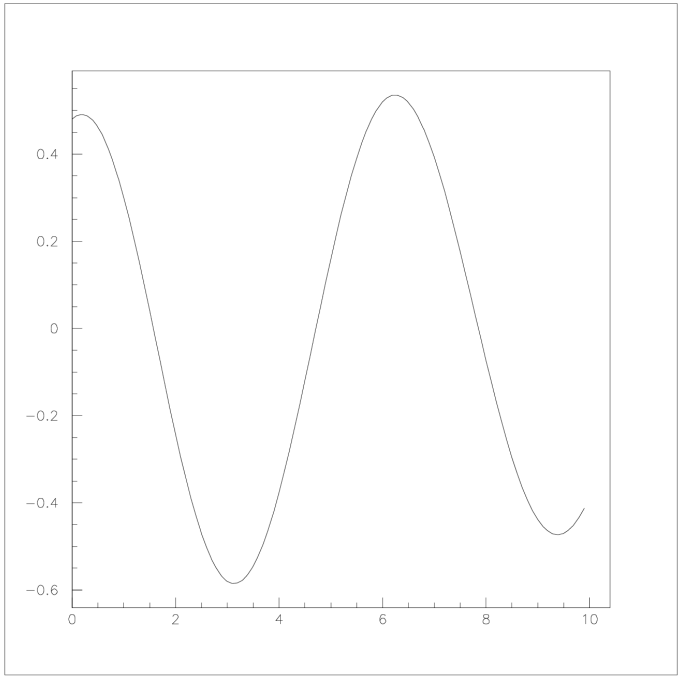


Figure 1: The result.

1

"The genious idea"

#dependencies:

```
paper.ps: paper.tex plot.eps

plot.eps: data.dat plotit.kumac

data.dat: a.out

a.out: numerics.c fun.c

fun.c: doit.mpl result

result: paper.tex
```

general part

formulae from paper, manipulated by maple

The rules specify how the dependencies are updated

Rules to update the dependencies

```
paper.ps: paper.tex plot.eps
        latex paper.tex; latex paper.tex; dvips paper.dvi -o paper.ps

plot.eps: data.dat plotit.kumac
        pawX11 -b plotit.kumac

data.dat: a.out
        rm -f data.dat; ./a.out > data.dat

a.out: numerics.c fun.c
        gcc -lm numerics.c

fun.c: doit.mpl result
        rm -f fun.c; maple < doit.mpl

result: paper.tex
        ExtractCode.csh paper.tex

clean:
        rm -f result fun.c a.out paper.aux paper.log paper.dvi paw.metafile

distclean: clean
        rm -f data.dat plot.eps paper.ps
```

do some mathematics produce part of the numeric program

Extract formulae form tex-file

How to write your LaTeX formula

$$f(n,k,x) = \binom{n}{k} \frac{(x+2)(x+3)}{(x+5)^3}$$

```
\begin{equation}
  %%STARTCODE
  \res = \binomial(n,k) \* {(x+2)\*(x+3) \over (x+5)^3}
  %;
  %%ENDCODE
  ,
\end{equation}
```

tag to extract formula

Macro taken from Jos:

```
\def\binomial(#1,#2){ \left(\!\!\begin{array}{c} #1 \\ #2 \end{array}\!\! \right) }
```

→ can be converted with a few `editor` commands to maple/math

Use `sed` to do this automatic

sed = stream editor, see example for details

How to extract and move information

Use shell-tools like

```
cat, sed, ed, awk, cut, paste,…
```

to extract and modify information

For more complicated situations use

Regular Expressions

Regular Expressions are similar to pattern matching with wildcards in Form

Example: Translate a(1),a(2),a(3) into b[1], b[2], b[3]

Regular Expressions can save you a lot of time during development or in porting information

Extract formulae from LaTeX

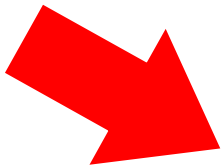
```
\begin{equation}
  %%STARTCODE
  \res = \binomial(n,k) \* {(x+2)\*(x+3) \over (x+5)^3}
  %;
  %%ENDCODE
  ,
\end{equation}
\begin{equation}
  %%STARTCODE
  b = \cos(x)
  %;
  %%ENDCODE
  ,
\end{equation}
\begin{equation}
  %%STARTCODE
  w = b \times \res
  %;
  %%ENDCODE
  .
\end{equation}
Using
\begin{equation}
  %%STARTCODE
  n = 5
  %;
  %%ENDCODE
  ,
\end{equation}
\begin{equation}
  %%STARTCODE
  k = 2
  %;
  %%ENDCODE
  .
\end{equation}
```

File: paper.tex

+

```
#!/bin/csh

rm -f result
sed -n '/%%STARTCODE/,/%%ENDCODE/ p' paper.tex | \
sed -f sed.cmd > result
```

File: ExtractCode.csh

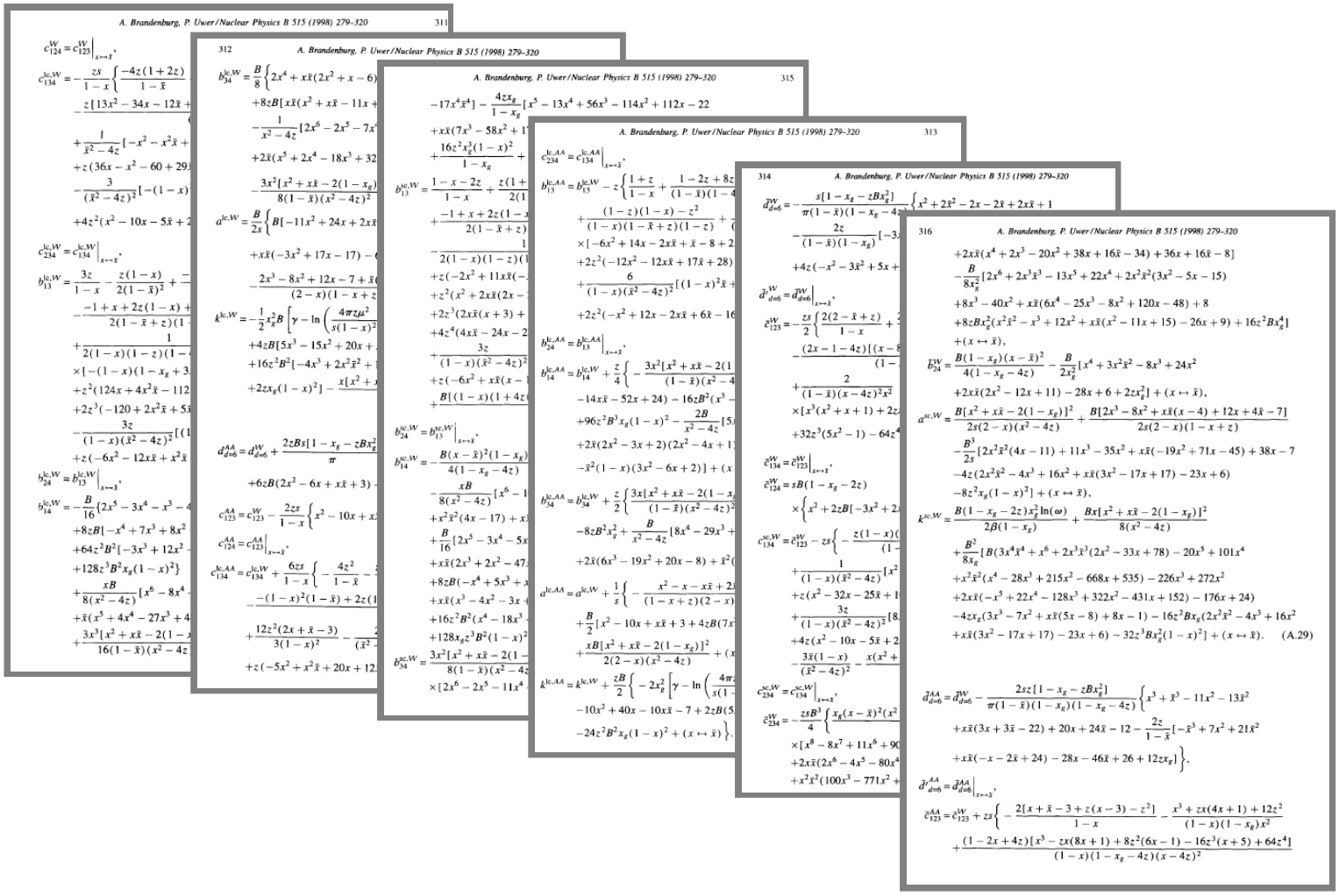


```
res := binomial(n,k) * ((x+2)*(x+3) )/( (x+5)^3)
  ;


  b := cos(x)
  ;


  w := b * res
  ;


  n := 5
  ;


  k := 2
  ;
```

File: result

easy to implement
automatic checking

How it was done in the last millenium…



→To assure the correctness of the published formulae we typed them in again…

no fun and clearly not very efficient

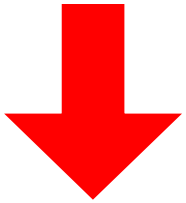
Produce C code from expressions using Maple

→ Similar features in Mathematica

```
read "./result";

with(CodeGeneration);

C(w,declare=[x::numeric],resultname="res",output="fun.c");
```

File: doit.mpl

Maple



```
res = 0.10e2 * cos(x) * (x + 0.2e1) * (x + 0.3e1) * pow(x + 0.5e1, -0.3e1);
```

File: fun.c

Useful option: `optimize`

Remark

Using the command line interface to maple/mathematica is important here!

[see also Thomas Hahn's lecture]

Everything can be run from a shell / batch queue !

Additional remark on worksheets:

- Not unusual that after an update worksheet becomes unreadable
- Porting a text file (ascii) from one CAS to another easier than porting a worksheet

Example

The master equation:

$$f(n,k,x) = \binom{n}{k} \frac{(x+2)(x+3)}{(x+5)^3}, \quad (1)$$

$$b = \cos(x), \quad (2)$$

$$w = b \times f(n,k,x). \quad (3)$$

Using

$$n = 5, \quad (4)$$

$$k = 2. \quad (5)$$

A very sopisticated evaluation on a $2,000,000 computer yields Figure 1.

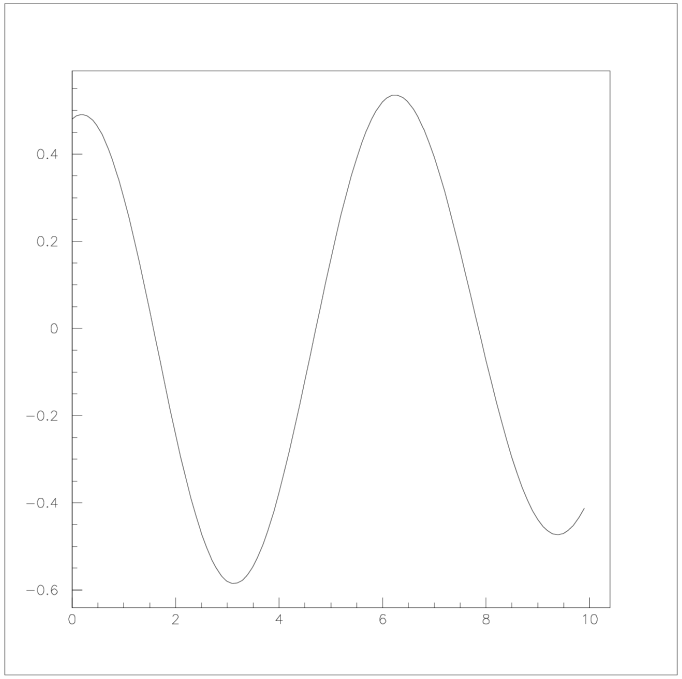


Figure 1: The result.

/afs/ifh.de/user/p/puwer/public/capp/exercises/make

1

```
paper.ps: paper.tex plot.eps
        latex paper.tex; latex paper.te

plot.eps: data.dat plotit.kumac
        pawX11 -b plotit.kumac

data.dat: a.out
        rm -f data.dat; ./a.out > data.

a.out: numerics.c fun.c
        gcc -lm numerics.c

fun.c: doit.mpl result
        rm -f fun.c; maple < doit.mpl

result: paper.tex
        ExtractCode.csh paper.tex

clean:
        rm -f result fun.c a.out paper.aux

distclean: clean
        rm -f data.dat plot.eps paper.p
```

make updates the entire project automatically

Automate testing

A long the same lines automate the testing

Possible checks:

- Evaluate numerically
- Check known cases
- Check symmetries
- …

be efficient in finding bugs

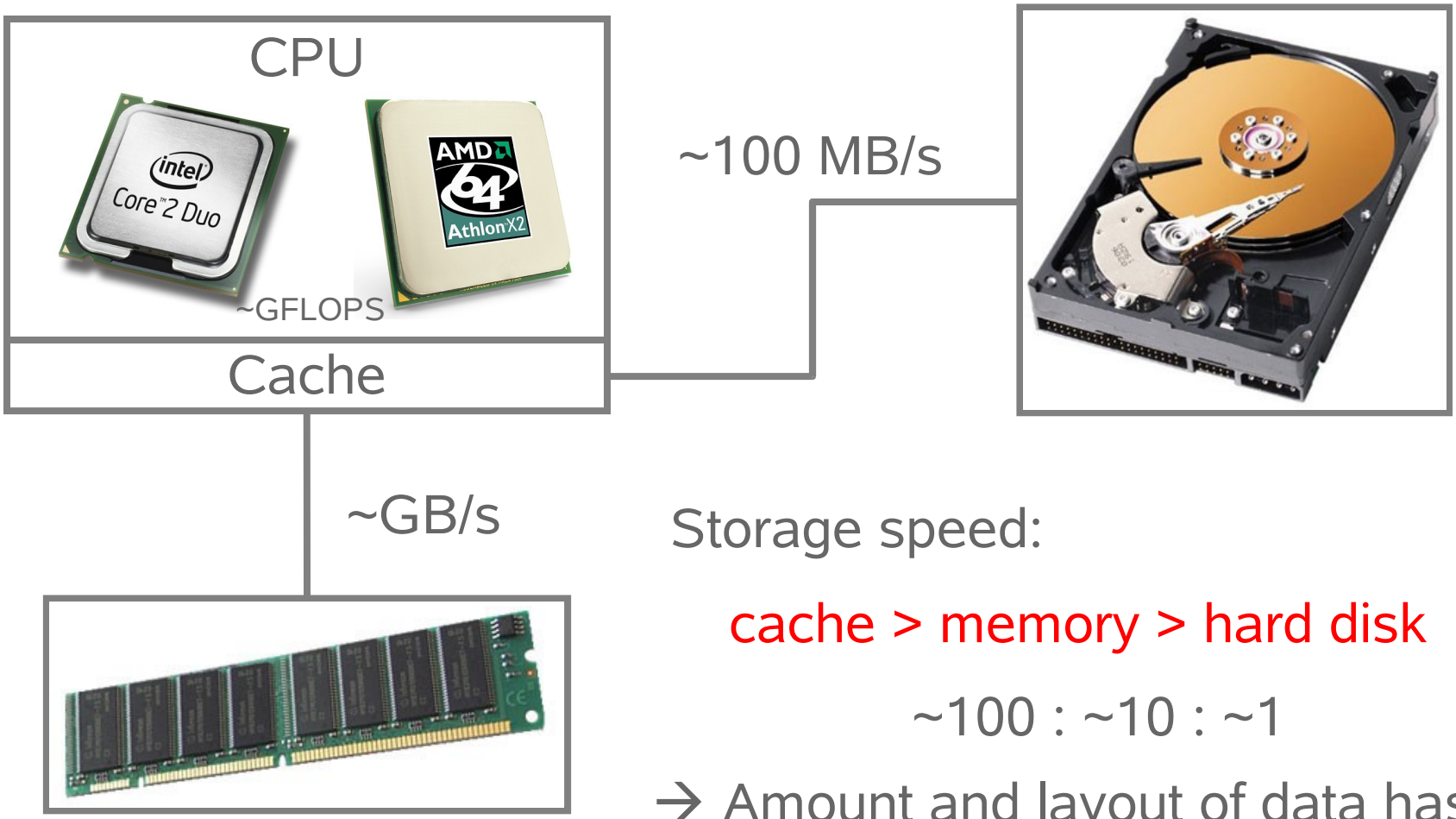
Automating these checks saves you a lot of time

→ debugging during development
→ comparison with your colleagues
→ assure correctness after modifications

Extreme case: First write the checks and than solve the problems

Nice side effect: Helps you in splitting/organizing your project in individual modules

Basic layout of a computer



Storage speed:

cache > memory > hard disk

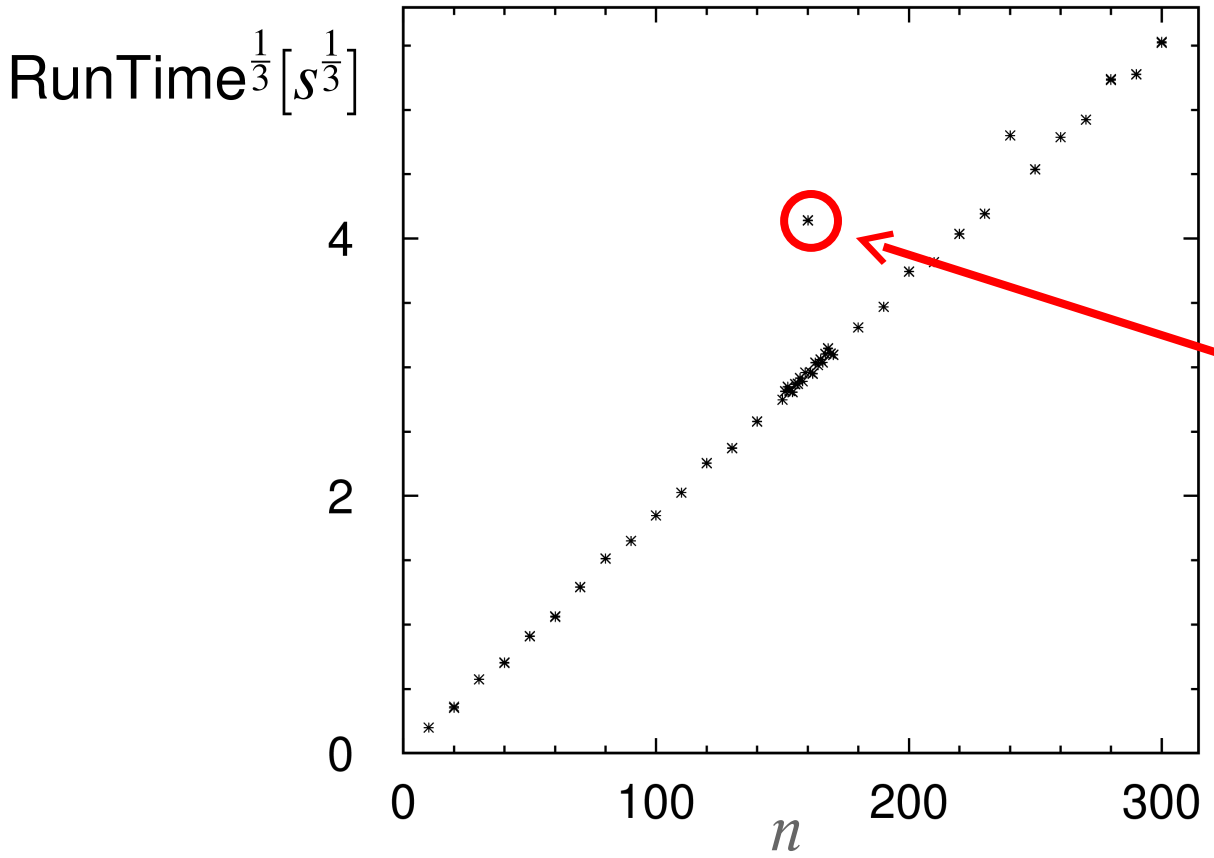
~100 : ~10 : ~1

→ Amount and layout of data has important impact on speed

The more memory you need the more you are limited by the bandwidth

Memory is accessed via cache

If your application conflicts with the way the cache works, this may have dramatic impact on performance:



for n = 160:

71s ← → ~26s

Matrix multiplication of n x n matrix → $O(n^3)$

Some details about the cache

Cache is organized in cachelines with 64byte each

For specific address region only a limited number of these lines are available

If you need more than the available number the cache will stop working → severe performance problem

→ align data to specific addresses,
→ keep data which is used together close to each other in memory
→ avoid data sizes conflicting with the cache mechanism

Important consequence

Obtain best performance if expressions stay in memory

If hard disk is used, speed of your application is limited by bandwidth of hard disk, don't use network file system in such cases

…your home directory is usually located on a network file system (nfs,afs)…

→ Try to keep your expressions (=data) small

No general rule how to do this, try to reduce expressions to a basis with a minimal number of individual structures

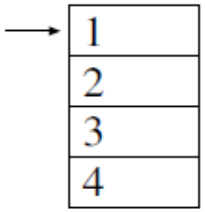
Example: $\frac{1}{1-x} - \frac{x}{1-x} - 1$ partial fractioning

Data storage: How

Some examples:

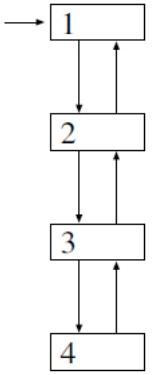
Arrays, Vectors:

$$\text{pos}_i = \text{base} + i \times \text{size}$$



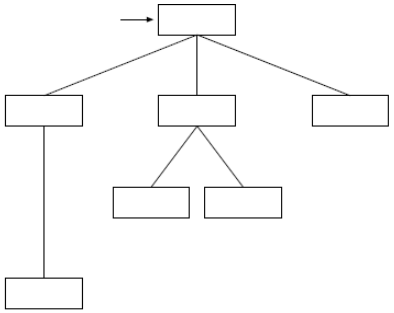
Access: $O(1)$
Insertion: $O(n)$

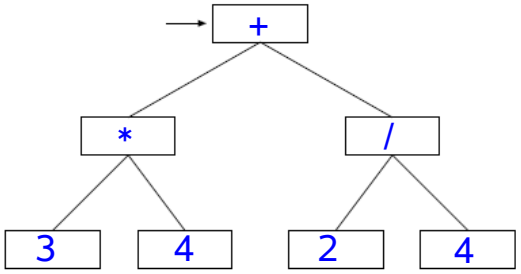
Lists:



Access: $O(n)$
Insertion: $O(1)$

Trees:





3*4+2/4

→ Adopt the way you store you data to the problem

Data storage: Where

static

at fixed distance with respect to the code,
available during the entire runtime

automatic

on the stack, disappear when stack is cleaned up,
i.e. after return from function

heap

Allocated dynamically during runtime, needs to be freed

Important for:

- Performance
- To understand error messages (stack overflow, segmentation fault)
- Parallelization using threads ("reentrant")

Note: Essentially no control over storage in computer algebra

Computing in Particle Physics

Analytic/Algebraic Methods

Numerical Methods

Numerics: Why ?

In the end we are interested in a number to compare with nature

→ there is always some numerics

Some algorithms are more useful when applied numerically

[see Malgorzata Worek's lecture, Costa Papadopoulos lecture]

Often we don't care about the intermediate (analytic) results

Numerics

What language shall we use ?

Supervisor: Fortran77, I did this 20 years ago and it was fine

The IT Expert: C++, only object-oriented programming makes sense

The performance expert: Use a compiled language otherwise it will be too slow

Some recommendations

- Use the language you know best
- Learning a new language in particular object oriented ones, is a huge investment
- Start with small projects
- Don’t translate old code into a new language

  → see Thomas Hahn’s lecture how to interface different languages

A short introduction into C++

Why C++?

- Very powerful language
- Many existing libraries
- Considered as language allowing efficient code reuse
- Can do both in C++: analytic as well as numerical calculations
- Well suited to deal with increasing complexity in computational physics
- Conceptually very different from non-object oriented languages

Object oriented programming — basic ideas

- Introduces new data types
- Put data and methods (≈functions) together (→ “class”)
- Hide the storage of the data and the implementation of the methods from the user

Example:

```
class TwoDimVec {
  public:
   double norm2();
  private:
   double x,y;
}
```

← No direct access to x,y!

Usage:

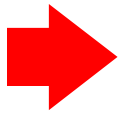
```
TwoDimVec a;
a.norm2();
```

Object oriented programming — basic ideas

If we change later

```
class TwoDimVec {
  public:
   double norm2();
  private:
   double x,y;
}
```

```
double norm2(){return(x*x+y*y);};
```



```
class TwoDimVec {
  public:
   double norm2();
  private:
   double x[2];
}
```

```
double norm2(){return(x[0]*x[0]
                      +x[1]*x[1]);};
```

the user would use the code as before

The class can ensure:

- that objects are correctly initialized
- that data always remains consistent
- that data which belongs together is stored together
- that objects are cleaned up when no longer used
- many more sophisticated things…

Object oriented programming — basic ideas

Function overloading — use the same name for functions with different arguments

```
int max(int a, int b);
float max(float a, float b);
double max(double a, double b);
```

compiler figures out which function to use,
programs become more readable

How does it work?

"name mangling"

The compiler appends the type of the arguments to the function name

→ annoying when interfacing with C and Fortran code

to switch off use: `extern "C" { … }`

Object oriented programming — basic ideas

Operator overloading — define basic operators for new class:

```
TwoDimVec a,b,c;
c = a + b;
```

(not possible in Java)

Useful applications:

- Code becomes more readable
  - Examples: Complex numbers, 4-vectors, histograms
- Can be used for algebraic manipulations (→ Ginac)

Note that internally the advanced concepts are still realized by function calls

Ignoring what happens internally may lead to severe performance problems

Looking under the hood

Assume a,b,c,… some sort of vector with + overloaded in the canonical way

How is

tmp = a + b + c + d + f

translated by the compiler?

Might be translated as:

```
tmp1 = add(a,b);
tmp2 = add(tmp1,c)
tmp3 = add(tmp2,d)
…
```

Instead of:

```
for(i=0,…){
   tmp[i] = a[i] + b[i] + …
}
```

Be careful when using C++ for time critical applications !
Very difficult without detailed knowledge about C++

Object oriented programming — Advanced concepts

- Generic programming with templates

Example: Sorting of objects relies only on a comparison function (“is_greater(a,b)”)

→ can be formulated for arbitrary data types, provide code as template, compiler generates the specialization

Generic formulation of algorithms

Many useful examples in the Standard Template Library (STL)

→ have a look before developing something from scratch

Useful application:

Function templates allow easily to switch to higher accuracy using QD for example

[see also D. Bailey’s lecture]

Object oriented programming - Advanced concepts ++

- Expressions templates and template meta programming

Basic idea:

Let the compiler do the work for you,
avoid draw back of some nice C++ features,
i.e. overhead in operator overloading

→ very sophisticated technology, highly non-trivial

```
template<int N>
class Factorial {
public:
  static const double value =  N * Factorial<N-1>::value;
};
template<>
class Factorial<0>{
public:
  static const double value = 1.;
};
```

Usage:

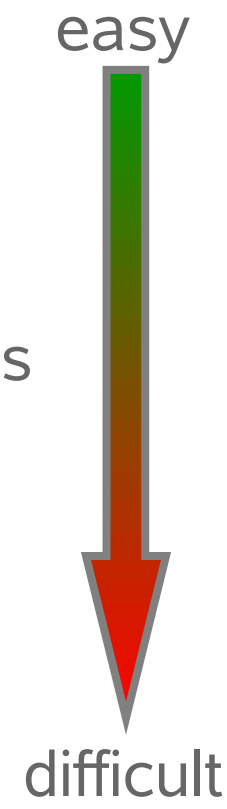
```
Factorial<69>::value;
```

The value is substituted at compile time,
no time needed during execution!

Object oriented programming — Advanced concepts

Compose new classes from old ones via

- Composition → built new classes from old ones
- Inheritance → extend existing classes by new methods
- Polymorphism → make old code calling new code



Recommendation

- Start with C using the C++ compiler
- Start with the easy concepts
- A good object oriented design needs a lot of time

Conclusion

Being efficient is very difficult !

Depends on

- Algorithms
- Implementation
- Tools / Hardware
- Organizing the work in an efficient way

→ No general rule, try out various methods to see what works best for you

The more you know what is available on the market the more likely you may find the right tools

I hope, I gave you some ideas how to address the problem

Exercises:

1) Go to the directory `make`
   - Run the project, understand how it works
   - Extend it by “checks”

2) Go to to the directory `sed-awk`
   - Add the data in `data.dat` using `awk`
   - Create C assignments `a_i = #` from data.dat using `awk`
   - Translate the result `a_i =...` → `a[i] =...` using `sed`

3) Define a function template in C++
   - Evaluate the function in single and double precision
   - Evaluate the function in double double using QD

4) Calculate Fibonacci numbers using template techniques

see /afs/ifh.de/user/p/puwer/public/capp/exercises