Simulation und wissenschaftliches Rechnen (SiwiR)

Christoph Pflaum
Content of the Lecture

- Why performance optimization?
- Short introduction to computer architecture and performance problems
- Performance optimization.
- Libraries and expression templates.
- Finite differences, iterative solvers.
- Introduction to parallelization.
- Debugging.
- Finite difference discretizations, stability
- Ray tracing
Simulation in Fluid Dynamics

Assume that we want to compute the flow of water in a hydroelectric power plant.
Simulation in Fluid Dynamics

- It is impossible to compute the flow exactly.
- We have to compute an approximate solution on a discretization grid.

Example of a 2D discretization grid:

$k_y$ grid points

$|h|$ $k_x$ grid points
In 3D, $O(k_x * k_y * k_z)$ data and $O(k_x * k_y * k_z * k_t)$ floating-point operations are needed.

Example: $k_x = k_y = k_z = 200$ and $k_t = 10000$.
This leads to: $k_x * k_y * k_z = 8 * 10^6$ data and $k_x * k_y * k_z * k_t = 8 * 10^{10}$ operations.
Due to technical reasons the clock rate cannot be arbitrary high.

In the last years the CPU performance (clock rate, ...) of processors increased more than the performance of memory (bandwidth, ...).
Latency and Bandwidth

Definition 1 (Latency and bandwidth, access time).

- The latency $L$ is the time needed until the execution of an instruction can start.
- The execution of every instruction needs a certain computational time.
- The bandwidth $B$ is the maximum speed of message transfer in Mbyte/sec (or Gbps) for an infinitely large message.

Thus, the time $T$ for sending a message of size $M$ is:

$$T = L + \frac{M}{B}.$$

The time for reading a certain amount of data from memory is often called access time.
The performance depends on:

- latency and bandwidth or access time of the memories and
- the latency and execution time of instructions of the CPU.
- pipeline concepts and parallelization on instruction level of the processor.
Pipeline Concept of a Processor

FETCH instructions

DECODE instructions

COMPUTATION and/or LOAD / STORE of data

WRITE data

1 cycle 1 cycle 2 cycle 1 cycle

→ latency of a single instruction: 2 cycles
The latency of several instructions can be reduced by a pipeline concept.

Example 1.

*AMD Opteron:* 15 pipeline stages

*Intel Nehalem:* 16 pipeline stages
Pipeline Concept of a Processor

→ reduction of the latency by parallel computations in a pipeline concept.
Bypassing

Example: Computation of $x * (a + b)$. By a “bypassing concept”, the result of $a + b$ can be used directly after computing it for a multiplication with $x$. 
Fusion of Multiply and Add

Example: Computation of $x \ast a + b$.
Several processors are able to compute one multiplication and one addition by one instruction.
Parallel Computations in a Processor

Modern processors are able to perform several instructions in parallel. This can be obtained by

- superscalar processors and
- VLIW processors (very long instruction word)
  e.g. EPIC-concept
  (Explicitly Parallel Instruction Computing)

Example 2.

- superscalar processors: usually:
  2 floating point operations and 2 integer operations and
  1 read or write of data.
- Itanium 2: EPIC
- Radeon R600: GPU
If a pipeline cannot accept a new instruction at a certain stage, then this is called “stalled”. There exist several reasons for this. One is that certain data are needed which are not contained in registers. Another may be that a previous computation has to end until the new computation can be performed.

→ This increases the latency time.
Access Time of Data

main memory
  e.g. maximal 4GByte

  access time: 32 cycles

  cache
  e.g. 32kByte or 3MByte

  access time: 3 cycles

  register
  e.g. 32

  access time: 0 cycles

  CPU
Cache Sizes

The cache consists of several parts of different sizes. A large cache implies a higher access time.

- **L1**: 16 KB, access time 5+
- **L2**: 256 KB, access time 5+
- **L3**: 3 MB, access time 12+
Blocks in Memories

- Fully associative: Block 11 can go anywhere.
- Direct mapped: Block 11 goes to $3 = 11 \mod 8$.
- Set associative (4 sets): Block 11 can go anywhere in set 3.

Memory block number:

- Set 0: 0 1 2 3 4 5 6 7
- Set 1: 8 9 10 11 12 13 14 15
- Set 2: 16 17 18 19 20 21 22 23
- Set 3: 24 25 26 27 28 29 30 31

Memory:
## Example: Intel ’Nehalem’ Architektur

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>32KB</td>
<td>256 KB</td>
<td>2MB</td>
</tr>
<tr>
<td>line size</td>
<td>64</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>number of lines</td>
<td>512</td>
<td>2048</td>
<td>16,384</td>
</tr>
<tr>
<td>associative sets</td>
<td>64</td>
<td>256</td>
<td>1024</td>
</tr>
<tr>
<td>associative</td>
<td>8-way</td>
<td>8-way</td>
<td>16-way</td>
</tr>
</tbody>
</table>
Example: Itanium 2

<table>
<thead>
<tr>
<th>Itanium 2</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>16KB</td>
<td>256 KB</td>
<td>3MB</td>
</tr>
<tr>
<td>line size</td>
<td>64</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>number of lines</td>
<td>256</td>
<td>2048</td>
<td>24,576</td>
</tr>
<tr>
<td>associative sets</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>associative</td>
<td>64-way</td>
<td>256-way</td>
<td>2048-way</td>
</tr>
<tr>
<td>update policy</td>
<td>write</td>
<td>write</td>
<td>write</td>
</tr>
<tr>
<td></td>
<td>through</td>
<td>back</td>
<td>back</td>
</tr>
</tbody>
</table>
## Example: IBM Power 3

<table>
<thead>
<tr>
<th>IBM Power 3</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>65KB</td>
<td>4MB</td>
</tr>
<tr>
<td>line size</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>number of lines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>associative sets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>associative</td>
<td>128-way direct mapped</td>
<td></td>
</tr>
<tr>
<td>update policy</td>
<td>write through</td>
<td>write back</td>
</tr>
</tbody>
</table>
Cache Misses

- Compulsory cache misses: Every data have to be fetched a first time to the cache. These cash misses cannot be avoided.

- Capacity cache misses: Every cache has a maximal size. Therefore it might happen, that a cash line was overwritten by another cash line.

- Conflict cache misses: If the cache is a directly mapped or set associative cache, then it may happen, that the cache cannot completely be used. Thus, cache lines will be overwritten, however there are free cache lines.
Common Subexpression Elimination

Instead of

\[
\begin{align*}
q &= a+b+c; \\
p &= a+b+d;
\end{align*}
\]

the compiler evaluates

\[
\begin{align*}
t &= a+b; \\
q &= t+c; \\
p &= t+d;
\end{align*}
\]
Loop-Invariant Code Motion

The compiler optimizes

\[
\text{for}(i=0; i<n; ++i) \\
\quad a[i] = r \times s + b[i];
\]
by

\[
t = r \times s; \\
\text{for}(i=0; i<n; ++i) \\
\quad a[i] = t + b[i];
\]
Evaluation of Constants

The compiler optimizes
\[ x = 3 \times 4.0 + y; \]
by
\[ x = 12.0 + y; \]

\[ \rightarrow \text{Optimization by meta-programming in C++!} \]
Strength Reduction

For an integer $i$ the compiler replaces

$$2\times i$$

by

$$i+i$$

In FORTRAN, the compiler replaces

$$x^{\times 2}$$

by

$$x\times x$$
Instead of

\[
\begin{align*}
a &= b + c; \\
d &= 2.0 \times a + e; \\
g &= 2.0 \times c; \\
q &= g + b \times 2.0; \\
\end{align*}
\]

the compiler could evaluate

\[
\begin{align*}
a &= b + c; \\
g &= 2.0 \times c; \\
d &= 2.0 \times a + e; \\
q &= g + b \times 2.0; \\
\end{align*}
\]

and try to optimize the use of the registers. This is a very complex optimization problem.
Performance Optimization: Mult-Add

Several processors perform $a + b \times c$ as fast as one multiplication. Thus,

$$a = b + c \times d + f \times g;$$

often is faster than

$$a = f \times g + c \times d + b;$$
Eliminating Overheads

There exist a lot of ways to avoid overheads. A simple example is the following. Replace

```c
if(sqrt(tt) < eps) { ... }
```

by

```c
if(tt < eps*eps) { ... }
```
Loop Unrolling

Loop unrolling is the general concept to improve performance!

Instead

```c
for(int i=0; i<n*m; ++i) Comp(i);
```

perform

```c
for(int i=0; i<m*n; i=i+n)
    for(int j=0; j<n; j=j+1) Comp(i+j);
```

or

```c
for(int i=0; i<m*n; i=i+n) {
    Comp(i+0);
    Comp(i+1);
    ...
    Comp(i+n-1);
}
```

Perform additional changes of the computations in the interior loop!
Loop Unrolling

Loop unrolling can optimize the performance of a code by
- software pipelining
- instruction parallelization
- improvement of the memory access.
Instruction-Parallelization
Pentium 4 - Vectorization

The Pentium 4 architecture allows two floating point instructions per cycle by SSE2 floating point instructions. Using the option \(-xW\) for the Intel-Compiler this leads to a so called “vectorization”. For example the compiler shows the output:

```
cpc -O3 -xW -c main.cc
main.cc(32) : (col. 3) remark: LOOP WAS VECTORIZED
main.cc(84) : (col. 5) remark: LOOP WAS VECTORIZED
icpc -O3 -xW -o run main.o -lm
```
Pentium 4 - Vectorization

for(i=0; i<n; ++i)
for(i=0; i<n; ++i)
Improvement of Memory Access
Automatic Loop Unrolling

Loop unrolling is often automatically performed by the compiler. But in some cases it is impossible for the compiler to unroll a loop. An example is:

```c
sum = 0.0;
for(int i=0; i<n; i=i+1)
    for(int j=0; j<n; j=j+1) {
        sum = sum + x[i][j] * (i*i + j*j);
    }
```

By hand it is possible to do a loop unrolling with respect to $i$ and $j$. 
Limit of Loop Unrolling

There is a limit for the size of loop unrolling. This limit is caused by

- a limited number of registers and
- overhead caused by too small loops and a loop length, which is not a multiple of the size of the interior loop.
Optimization of Memory Access

Optimization of memory access is very important in HPC!

The general rule is:

Optimize data locality!

This means compute with data from cache!
Example in FORTRAN

```fortran
dimension a(n,n),b(n,n)
LOOP A
  do 10,i=1,n
    do 10,j=1,n
    10   a(i,j)=b(i,j) * b(i,j)+1.
```

and

```fortran
dimension a(n,n),b(n,n)
LOOP B
  do 10,j=1,n
    do 10,i=1,n
    10   a(i,j)=b(i,j) * b(i,j)+1.
```

Which version is faster?
Example in C

double a[n][n], b[n][n];
// LOOP A
for (i=0; i<n; ++i)
    for (j=0; j<n; ++j)
        a[i][j] = b[i][j] * b[i][j] + 1.0;

and

double a[n][n], b[n][n];
// LOOP B
for (j=0; j<n; ++j)
    for (i=0; i<n; ++i)
        a[i][j] = b[i][j] * b[i][j] + 1.0;

Now, loop A is faster!
double **a, **b;
a = new double* [n];
b = new double* [n];
for(i=0;i<n;++i) {
    a[i] = new double[n];
    b[i] = new double[n];
}

// LOOP C
for(i=0;i<n;++i) {
    for(j=0;j<n;++j)
        a[i][j]=b[i][j]*b[i][j]+1.0;
}

Now, the data of a are cut in several pieces. This leads to less data locality and optimizations as vectorization cannot be performed in an optimal way.
Efficient Dynamic Memory Allocation

The following code leads to a better data allocation:

```c
double *a, *b;
a = new double[n*n];
b = new double[n*n];

// LOOP C
for (i=0; i<n; ++i)
    for (j=0; j<n; ++j)
        a[i*n+j] = b[i*n+j] * b[i*n+j] + 1.0;
```

Using such a data structure, an optimal performance can be obtained on vector machines.
Loop Fusion

Consider the following code. Instead of

\[
\text{for}(i=0; i<n; ++i) \\
\quad u[i] = u[i] + \tau \times g[i]; \\
\text{for}(i=0; i<n; ++i) \\
\quad r[i] = b[i] + \alpha \times g[i];
\]

implement

\[
\text{for}(i=0; i<n; ++i) \\
\quad \{ \\
\quad \quad u[i] = u[i] + \tau \times g[i]; \\
\quad \quad r[i] = b[i] + \alpha \times g[i]; \\
\quad \}
\]
Data Layout

Construct a data layout such that the computations can be done locally.
As an example consider the coordinates of particles. In FORTRAN write

\begin{verbatim}
    dimension r(3,n)
\end{verbatim}

instead of

\begin{verbatim}
    dimension rx(n), ry(n), rz(n)
\end{verbatim}
Blocking

Blocking is similar to loop unrolling. Consider the matrix transposition

\[ \text{dimension } a(n,n), b(n,n) \]

\[
\text{LOOP A} \\
do 10, i=1,n \\
do 10, j=1,n \\
10 \quad b(i,j)=a(j,i)
\]

Subdivide the index set in small blocks of size \( s \times s \):

\[
(1, 1) \quad \ldots \quad (1, n) \quad \ldots \quad (k_1, k_2) \quad \ldots \quad (k_1, k_2 + s) \\
\vdots \quad \ldots \quad \vdots \quad \quad \quad \quad \vdots \quad \ldots \quad \vdots \\
(n, 1) \quad \ldots \quad (n, n) \quad \ldots \quad (k_1 + s, k_2) \quad \ldots \quad (k_1 + s, k_2 + s)
\]

Then, perform the matrix transposition on each of these blocks.

The size of the cache must be larger than \( 2 \times s \times s \).
Compilers try to perform an automatic optimization. In particular, FORTRAN compilers are able to optimize a code by loop unrolling and automatic instruction parallelization. Using C or C++, there is a problem with aliasing. Let us consider the program:

```c
void f(double *a, double *b, double *c, double *d) {
    for(int i=0; i<n; ++i)
        a[i] = b[i] + c[i] * d[i];
}
```

Then, the C compiler does not know whether `b[i]` and `a[i-1]` point to the same value or not. Therefore, some compilers cannot perform an automatic optimization in this case.
To avoid the problem with *aliasing* some compilers support the keyword *restrict* or __restrict__ for pointers as follows:

```c
double * restrict a;
double * restrict b;
double * restrict c;
double * restrict d;

for(int i=0; i<n; ++i)
    a[i] = b[i] + c[i] * d[i];
```

Shared Memory Computer Architecture

proc 1

proc 2

proc 3

proc 4

Crossbar

memory
Parallelization with OpenMP

The parallelization with OpenMP is based on

- threads
- the usage of pragmas like `# pragma omp parallel for`
A simple parallelization of for loops in OpenMP can be obtained as follows:

```c
#include <omp.h>

int main() {

    double * __restrict a;
    double * __restrict b;
    double * __restrict c;

    #pragma omp parallel for
    for(int i=0;i<n;++i) {
        c[i] = a[i]*a[i] + b[i] * b[i] * b[i];
    }

```
Usage of restrict with OpenMP

Parallelization with and without __restrict
Construction in OpenMP

For more complicated constructions the simple pragma

```
#pragma omp parallel for
```

is not sufficient to obtain an efficient parallelization. One reason for poor performance of an OpenMP parallelization might be that the threads often need the same data from main memory. One way to avoid this is tell the compiler, that a variable is only used `private` by every thread. This can be done by the private construction as follows:
... double * __restrict a;
double * __restrict c;
double sum;
int i,j;
...

// good version
#pragma omp parallel for private(j,sum)
for(i=0; i<n; ++i) {
    sum = 0.0;
    for(j=0; j<n; ++j) {
        sum = sum + a[i*n+j];
    }
    c[i] = sum;
}
}
Computational time for OpenMP parallelization with 2 threads:

<table>
<thead>
<tr>
<th>n</th>
<th>12</th>
<th>120</th>
<th>1200</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec</td>
<td>3.9e-7</td>
<td>4.6e-5</td>
<td>4.7e-3</td>
<td>4.9e-1</td>
</tr>
<tr>
<td>sec parallel (good version)</td>
<td>1.5e-6</td>
<td>2.4e-5</td>
<td>2.3e-3</td>
<td>2.4e-1</td>
</tr>
</tbody>
</table>
Bad Parallelization with OpenMP

... double * __restrict a; double * __restrict b; double * __restrict c; double sum; int i, j; ...

// bad version
#pragma omp parallel for private(j,sum)
for(i=0;i<n;++i) {
    sum = 0.0;
    for(j=0;j<n;++j) {
        sum = sum + a[j*n+i];
    }
    c[i] = sum;
}
**Bad Parallelization with OpenMP**

This parallelization increases the computational time:

<table>
<thead>
<tr>
<th>n</th>
<th>12</th>
<th>120</th>
<th>1200</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec</td>
<td>3.9e-7</td>
<td>4.6e-5</td>
<td>4.7e-3</td>
<td>4.9e-1</td>
</tr>
<tr>
<td>sec parallel (bad version)</td>
<td>1.5e-6</td>
<td>1.1e-4</td>
<td>1.9e-2</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Euclidian Norm

Let us assume we want to calculate the euclidian norm of a vector

$$\|v\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Then, the following code leads to the wrong result:

```c
... double norm;
    norm = 0.0;
#pragma omp parallel for
    for(i=0;i<n;++i) {
        norm = norm + a[i]*a[i];
    }
    norm = sqrt(norm);
```
A correct code can be obtained by the reduction construction in OpenMP as follows:

```c
... double norm;
norm = 0.0;
#pragma omp parallel for reduction(+ : norm)
for(i=0;i<n;++i) {
    norm = norm + a[i]*a[i];
}
norm = sqrt(norm);
```
reduction can be applied to the operators +, *, -, &, |, &&, ^, ||. Here, || reduces a maximum calculation of a variable.
Consider the loop

... 
for (i=1; i<n; ++i)
    a[i] = a[i-1] + b[i]; 
... 
}

OpenMP will not parallelize this loop in a correct way.
Not Parallelizable Relaxation Loop

OpenMP cannot parallelize the following loop in a correct way:

\[
\ldots \\
\text{for}(i=1; i<\text{n}-1; ++i) \\
\quad a[i] = 0.5 \times (a[i-1]+a[i+1]); \\
\ldots 
\]
Parallelizable Relaxation Loop

The following loop can be parallelized in a correct way by OpenMP:

```c
... 
#pragma omp parallel for
for(i=1;i<n-1;i=i+2)
    a[i] = 0.5*(a[i-1]+a[i+1]);

#pragma omp parallel for
for(i=2;i<n-1;i=i+2)
    a[i] = 0.5*(a[i-1]+a[i+1]);
... 
```
The call of a function requires computational times. To avoid this problem a function can be defined to be inlined.

**Example:**

```cpp
inlining double f(double x) { ... };
```

**Advantage:**
- optimization of the code in the area where the function is called (such as common subexpression elimination and vectorization)
- no overhead by calling the function

**Disadvantage:**
- longer compilation time
- longer executable code
Parameters of functions which will not be changed should be defined to be `const`.

**Example:**

```cpp
inline double f(const double x) { ... };
```

Member functions of a class which do not modify member values of the class should be defined to be `const` member functions:

**Example:**

```cpp
class A {
...
    inline double f(const double x) const { ... };
};
```

`const` can help a compiler to optimize a code.
The compiler optimizes
\[ x = 3 \times 4.0 + y; \]
by
\[ x = 12.0 + y; \]
Can we obtain such an optimization for
\[ x = \text{Factorial}(4) + y; \]
where \text{Factorial}(4) \text{ mathematically means}
\[ 4! = 1 \times 2 \times 3 \times 4 = 24 \]
Factorial by Meta-Programming

Consider the C++ construction

```cpp
template<int N>
class Factorial {
public:
    enum { value = N * Factorial<N-1>::value };
};
class Factorial<1> {
public:
    enum { value = 1 };
};

Then, the compiler replaces

```cpp
x = Factorial<4>::value + y;
```

by

```cpp
x = 24 + y;
```
Meta-Programming

Meta-Programming means to write a program, which is evaluated during compile-time and not during runtime.
Insulation Property of a Wall

outside: cold

\[ \lambda_{\text{insulation}} = 0.04 \frac{W}{mK} \]

inside: warm

\[ \lambda_{KS} = 0.56 \frac{W}{mK} \]

Figure 1: Construction of a wall
Simple Mathematical Model

\[ -\text{div} \lambda \ \text{grad} T = 0 \quad \text{on} \ \Omega \]
\[ T|_{\Gamma_{\text{out}}} = -10 \quad \text{on} \ \Gamma_{\text{out}} \]
\[ T|_{\Gamma_{\text{in}}} = 20 \quad \text{on} \ \Gamma_{\text{in}} \]
\[ \frac{\partial T}{\partial n}|_{\Gamma_{\text{N}}} = 0 \quad \text{on} \ \Gamma_{\text{N}} \]

\[ \Gamma_{\text{out}} \]

\[ \lambda_{\text{insulation}} \]

\[ \lambda_{\text{KS}} \]

\[ \Gamma_{\text{in}} \]

\[ \Gamma_{\text{N}} \]

Figure 2: Model of a wall
Model Problem

\[-\triangle T + cT = f \quad \text{on } \Omega\]

\[T|_{\Gamma} = g \quad \text{on } \Gamma\]

\[\Omega = (0, L)^2,\]

where \(c > 0\) is a constant and \(L > 0\) is the size of the domain. Observe that

\[-\text{div grad } T = -\triangle T = -\frac{\partial^2 T}{\partial x^2} - \frac{\partial^2 T}{\partial y^2}.\]
The first step in a finite difference discretization is the construction of a discretization grid:

\[ \Gamma_h, \Omega_h \]

Figure 3: Finite difference discretization grid
Finite Difference Discretization

Let $\Omega = (0, 1)^2$. Number the points of the discretization grid $\Omega_h$ by:

$$h(1, 1), ..., h(1, m - 1), h(2, 1), ...$$

Then, the FD discretization leads to an equation $L_h U_h = F_h$, where

$$L_h = \frac{1}{h^2} \begin{pmatrix} D_h & -E \\ -E & D_h & \ddots \\ & \ddots & \ddots & -E \\ -E & D_h \end{pmatrix},$$

and where

$$D_h = \begin{pmatrix} 4 & -1 \\ -1 & 4 & \ddots \\ & \ddots & \ddots & -1 \\ -1 & 4 \end{pmatrix}.$$
Finite Element (FE) Norms

The finite element method leads to approximations $u_h \in C(\Omega)$ of an exact solution $u \in C(\Omega)$ of a PDE. Suitable norms for calculating the discretization error are

$$
\| u - u_h \|_{L_\infty} := \max_{x \in \Omega} |(u - u_h)(x)|
$$

$$
\| u - u_h \|_{L_2} := \sqrt{\int_{\Omega} |(u - u_h)(x)|^2 \, dx}
$$

- These norms have the *normalization* property

  $$(u - u_h)(x) = 1 \quad \forall x, h \quad \Rightarrow \quad \| u - u_h \| = \text{const} \quad \forall h$$

- In case of solutions with singularities one can expect a better convergence for the $\| \cdot \|_{L_2}$ norm.
Suitable Norms for FD

Let $\Omega_h$ be a sequence of discretization grids. We are looking for a sequence of norms on $\mathbb{R}^{|\Omega_h|}$ with similar properties for the FD method.

**Example:** A not suitable norm is

$$\|w\| := \sqrt{\sum_{z \in \Omega_{h,i}} |w(z)|^2}.$$
Suitable Norms for FD

**Definition 2.** *We call the sequence of norms* $\| \cdot \|_{h_i}$ *on* $\mathbb{R}^{\Omega_i}$ *normalized, if*

$$\| 1 \|_{h_i} = 1,$$

*where* $1$ *is the constant function* $x \mapsto 1$.

**Example 3.**

$$\| x \|_2 := \sqrt{\frac{1}{|\Omega_i|} \sum_{z \in \Omega_i} x_z^2}$$

$$\| x \|_\infty := \max_{z \in \Omega_i} |x_z|$$
Convergence of the FD Method

**Theorem 1.** Consider the finite difference discretization of Poisson’s equation on $\Omega = (0, L)^2$ with meshsize $h$. Then, there is a constant $C > 0$ such that

$$\|u - u_h\|_\infty \leq Ch^2 \left( \left\| \frac{\partial^4 u}{\partial x^4} \right\|_\infty + \left\| \frac{\partial^4 u}{\partial y^4} \right\|_\infty \right).$$

**Example:**

- If $u = x^2 \ast y^3$, then $u = u_h$,
- If $u = x^4$, then $\|u - u_h\|_\infty \leq Ch^2$. 


Example: Poisson’s equation on \((0, 1)^2\).
Let \(f(x, y) = -12.0 \times x^2 - \exp(y)\). Then, the exact solution of

\[
-\triangle u = f = -12.0 \times x^2 - \exp(y) \quad \text{on } \Omega
\]
\[
u|_{\partial\Omega} = x^4 + \exp(y) \quad \text{on } \partial\Omega
\]
is

\[
u(x, y) = x^4 + \exp(y).
\]

The following table depicts the error \(e_{h,\text{max}} := \|u - u_h\|_{\infty}\):

<table>
<thead>
<tr>
<th>(h = )</th>
<th>0.5</th>
<th>0.25</th>
<th>0.125</th>
<th>0.0625</th>
<th>0.03125</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N = )</td>
<td>1</td>
<td>9</td>
<td>49</td>
<td>225</td>
<td>961</td>
</tr>
<tr>
<td>(e_{h,\text{max}} \approx)</td>
<td>0.033</td>
<td>0.0094</td>
<td>0.0024</td>
<td>0.00061</td>
<td>0.00015</td>
</tr>
<tr>
<td>(e_{h/2,\text{max}}/e_{h,\text{max}} \approx)</td>
<td>0.28</td>
<td>0.26</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>
How to Choose the Meshsize $h$

Assume that the discretization error converges according

$$\| u - u_h \| \leq C h^p.$$  

How should we choose $h$ to obtain a discretization error $\| u - u_h \| \leq \eta$?

Assume that we can calculate $\| u_{h/2} - u_h \|$. Then, the assumption $\| u - u_h \| \approx C h^p$ leads to

$$\| u - u_h \| \approx \frac{1}{1 - 2^{-p}} \| u_{h/2} - u_h \|. \quad (1)$$

Thus, we have to choose $h$ such that

$$\| u_{h/2} - u_h \| \leq \eta(1 - 2^{-p}).$$
Eigenvectors and Eigenvalues of $L_h$

Consider the FD discretization of Poisson’s equation on the unit square. Then, the matrix $L_h$ has the eigenvalues

$$\lambda_{\nu,\mu} = \frac{4}{h^2} \left( \sin^2 \left( \frac{\pi \nu h}{2} \right) + \sin^2 \left( \frac{\pi \mu h}{2} \right) \right)$$

with eigenvectors

$$e_{\nu,\mu} = \left( \sin(\nu \pi x_i) \sin(\mu \pi y_j) \right)_{(x_i,y_j) \in \Omega_h}$$

where $\nu, \mu = 1, \ldots, m-1$

and $h = \frac{1}{m}$.

- Smallest eigenvalue: $\frac{4}{h^2} 2 \sin^2 \left( \frac{\pi h}{2} \right) \approx 2\pi^2$.
- Largest eigenvalue: $\frac{4}{h^2} 2 \sin^2 \left( \frac{\pi (m-1)}{2m} \right) \approx \frac{8}{h^2}$. 
Eigenvectors and Eigenvalues of $L_h$

eigenvector $e_{1,1}$

eigenvector $e_{2,1}$

eigenvector $e_{3,3} = \left( \sin(3\pi x_i) \sin(3\pi y_j) \right)_{(x_i, y_j) \in \Omega_h}$
Direct and Iterative Solvers

The FD discretization leads to an equation system

\[ A_h u_h = b_h, \]

where \( A_h \) is an \( n \times n \) matrix and \( u_h, b_h \in \mathbb{R}^n \) are vectors. There are

- direct methods and
- iterative methods

for solving such an equation system.

Both methods lead to an approximation \( \tilde{u}_h \) of \( u_h \).

\[ \| u_h - \tilde{u}_h \| \] is called algebraic error.
Discretization Error and Algebraic Error

- $\|u - u_h\|$ is called discretization error.
- $\|u_h - \tilde{u}_h\|$ is called algebraic error.
- $\|u - \tilde{u}_h\|$ is called total error.

The algebraic error should satisfy the property

$$\|u_h - \tilde{u}_h\| \leq \|u - u_h\|\alpha,$$

where $\alpha \approx 0.1$.

Let the discretization satisfy $\|u - u_h\| \leq Ch^2$.
Then, this implies

$$\|u - \tilde{u}_h\| \leq C(1 + \alpha)h^2.$$
It is very difficult to calculate numerically the algebraic error $\|u_h - \tilde{u}_h\|$ and the discretization error $\|u - u_h\|$.

Therefore, one often calculates

- the residuum norm $\|A_h \tilde{u}_h - b_h\|$ or
- the norm $\|\tilde{u}_h - \tilde{u}_h/2\|$.

If the exact solution is known, then one can numerically calculate the total error

- total error $\|u - \tilde{u}_h\|$.
Direct Solvers for Linear Equation Systems

- The Gauss-elimination applied to a full matrix requires
  - $O(n^3)$ operations
  - $O(n^2)$ data

for solving a linear equation system with $n$ unknowns.

- The Gauss-elimination applied to a band matrix of bandwidth $2k - 1$ requires
  - $O(n \times k \times k)$ operations
  - $O(n \times k)$ data.

A band matrix of bandwidth $k$ has the form:

$$
\begin{pmatrix}
a_{11} & \cdots & a_{1k} \\
\vdots & a_{22} & \cdots & \cdots \\
a_{k1} & \cdots & \cdots & \cdots & a_{n-k+1,n} \\
\vdots & \vdots & \cdots & a_{n-1,n-1} & \vdots \\
a_{n,n-k+1} & \cdots & a_{nn}
\end{pmatrix}
$$
Direct Solvers for FD Discretization

Now consider the matrix of the FD discretization of Poisson’s equation on \( \Omega = (0, 1)^2 \). The discretization matrix is a band matrix of size \( n = (m - 1)^2 \) and bandwidth \( 2m - 1 \), since \( h = \frac{1}{m} \)

\[
L_h = \frac{1}{h^2} \begin{pmatrix} D_h & -E \\ -E & D_h & \ddots \\ & \ddots & \ddots & -E \\ -E & D_h \\ \end{pmatrix}, \quad \text{where} \quad D_h = \begin{pmatrix} 4 & -1 \\ -1 & 4 & \ddots \\ & \ddots & \ddots & -1 \\ & & \ddots & -1 & 4 \\ \end{pmatrix}
\]

Then, the Gauss-elimination applied to the band matrix \( L_h \) requires

- \( O(n^2) \) operations
- \( O(n^{1.5}) \) data.
Iterative Solvers

Consider the linear equation system $Ax = b$. An iterative solver for solving a linear equation system is a mapping

$$S : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

with start vector $x_0 \in \mathbb{R}$ such that the sequence $(x_i)_{i \in \mathbb{N}}$ defined by

$$x_{i+1} = S(x_i)$$

converges to $x$:

$$\lim_{i \rightarrow \infty} x_i = x.$$

Obviously, $x$ should satisfy the fix point property $x = S(x)$. 
Relaxation of the $i$-th unknown $x_i$:
Correct $x_i^{old}$ by $x_i^{new}$ such that the $i$-th equation of the equation system

$$A \cdot x = b$$

is correct.
Numbering of GS Relaxation

For Gauss-Seidel iteration one often applies lexicographical and red-black numbering of the grid points.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>24</td>
<td>12</td>
<td>25</td>
<td>13</td>
</tr>
<tr>
<td>21</td>
<td>9</td>
<td>22</td>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>19</td>
<td>7</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>17</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>2</td>
<td>15</td>
<td>3</td>
</tr>
</tbody>
</table>
Gradient Method

Let $A$ be a symmetric positive definite $n \times n$ matrix and $b \in \mathbb{R}^n$.
The gradient method for solving

$$Ax = b$$

is

Start with $x_0$ and calculate the sequence $x_k$ by:

$$d_k = b - Ax_k$$
$$\alpha_k = \frac{d_k^T d_k}{d_k^T Ad_k}$$

$$x_{k+1} = x_k + \alpha_k d_k,$$

where $k = 0, 1, 2, \ldots$. 
Let the start solution be $x_0$. Then:

\[ g_0 = Ax_0 - b \]
\[ \delta_1 = g_0^T g_0 \quad \text{if } \delta_1 \leq \epsilon \text{ stop} \]
\[ d_1 = -g_0 \]

**recursion:*** $k = 1, \ldots:$

\[ h_k = Ad_k \]
\[ \alpha = \frac{\delta_k}{d_k^T h_k} \]
\[ x_k := x_{k-1} + \alpha d_k \]
\[ g_k := g_{k-1} + \alpha h_k \]
\[ \delta_{k+1} = g_k^T g_k \quad \text{if } \delta_{k+1} \leq \epsilon \text{ stop} \]
\[ \beta_k = \delta_{k+1}/\delta_k \]
\[ d_{k+1} = -g_k + \beta_k d_k \]
Estimation of the Algebraic Error

Assume we want to solve

\[ Ax = b \]

and we get the approximation \( \tilde{x} \). A practical problem is:

- How large is the algebraic error \( \| \tilde{x} - x \| \)?
- Assume, we apply an iterative solver. How many iterations do we have to perform to obtain a small algebraic error?
- How to choose \( \epsilon \) in the cg-iteration?
- Assume, you have implemented two iterative solvers. Which one is faster?

**But:** We do not know \( x \)!
Estimation of the Algebraic Error for Tests

For testing a code one does the following test:
Construct right hand sides $b$ such that the exact solution $x$ is well-known.

**Example:**
- Choose $b = 0$.
- FD on a unit square: Choose $u = x^2 y^3$.

Start with $x_0 = 1$.

Then, one can compute the algebraic error $\|\tilde{x} - x\|$ and one can compare two different codes.
A Hard Approach

If the exact solution is unknown, one applies the following difficult approach:
Calculate a very good approximation $x_e$ of $x$ by a time consuming solver. Then, consider

$$\|\tilde{x} - x_e\|$$

as the algebraic error $\|\tilde{x} - x\|$. 
The residuum is defined as

\[ r := A\tilde{x} - b \]

Then,

\[ \|\tilde{x} - x\| = \|A^{-1}r\| \leq \|A^{-1}\|\|r\|. \]

**Example:**

FD, Poisson on \( ]0, 1[^2 \):

\[ \|A^{-1}\|_2 = \lambda_1^{-1} \approx \frac{1}{2\pi^2}. \]

- Assume that \( \tilde{x} - x = e_{m-1,m-1} + h^2e_{1,1} \). Then,

\[ \|\tilde{x} - x\|_2 \approx 1 \quad \text{and} \quad \|A^{-1}\|_2\|r\|_2 \approx h^{-2}. \]

- Assume that \( \tilde{x} - x = h^2e_{m-1,m-1} + e_{1,1} \). Then,

\[ \|\tilde{x} - x\|_2 \approx 1 \quad \text{and} \quad \|A^{-1}\|_2\|r\|_2 \approx 1. \]

Thus, if \( \|r\| \) is small, then \( \|\tilde{x} - x\|_2 \) can be large or small!
The residuum is defined as

\[ r := A\tilde{x} - b \]

Then, if \( \|r\| \) is small, then \( \|\tilde{x} - x\|_2 \) can be large or small! Therefore, do not use the size of the residuum to compare two different iterative algorithms.

**Example:**

FD, Poisson on \( ]0, 1[^2 \): We want to obtain \( \|\tilde{x} - x\|_2 = O(h^2) \).

- **MG**: Iterate such that \( \|r\| = O(1) \).
- **SSOR**: Iterate such that \( \|r\| = O(h^2) \).
Let $x_0 \in \mathbb{R}^n$ and

$$S : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

be an iterative solver such that the sequence $(x_i)_{i \in \mathbb{N}}$ defined by

$$x_{i+1} = S(x_i)$$

converges to $x$. Most of the iterative solvers have the following property:

There exists a constant $0 < q < 1$ and $s, i_{\text{min}} \in \mathbb{N}$ such that

$$\|x_{i+s+1} - x_{i+s}\| \leq q^s \|x_{i+1} - x_i\|$$

for every $i > i_{\text{min}}$. $q$ is called convergence rate of $S$. 
Theorem 2. Let $0 < q < 1$, $s, i_{\text{min}} \in \mathbb{N}$, $x_0 \in \mathbb{R}^n$ and

$$S : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

be an iterative solver such that the sequence $(x_i)_{i \in \mathbb{N}}$ defined by

$$x_{i+1} = S(x_i)$$

converges to $x$ and satisfies

$$\|x_{i+2} - x_{i+1}\| \leq q\|x_{i+1} - x_i\|$$

for every $i > i_{\text{min}}$. Then, the algebraic error can be estimated by

$$\|x - x_i\| \leq \|x_{i+1} - x_i\|(1 - q)^{-1}.$$
Estimation of the Convergence

We want to find a small parameter such that

$$
\|x_{i+2} - x_{i+1}\| \leq q \|x_{i+1} - x_i\|.
$$

Several iterative solvers have the following property:

*There exists a constant $0 < q < 1$ and $s, i_{\text{min}} \in \mathbb{N}$ such that*

$$
\|x_{i+s+1} - x_{i+s}\| \leq q^s \|x_{i+1} - x_i\|
$$

*for every $i > i_{\text{min}}$.*

- Calculate $\tilde{q} = \frac{\|x_{i+2} - x_{i+1}\|}{\|x_{i+1} - x_i\|}$ for large $i$.

- Calculate $\tilde{q} = \left( \frac{\|x_{i+s+1} - x_{i+s}\|}{\|x_{i+1} - x_i\|} \right)^\frac{1}{s}$ for large $i$, $s \approx 5 - 20$.

Take $\tilde{q}$ as an approximation of $q$. 
Convergence Rate of Linear Iterative Solvers

Let

\[ S(x_i) = Cx_i + d \]

be an iterative solver (\( C \) matrix and \( d \) vector).

Then, the convergence rate \( q \) does not depend on the right hand side \( b \) and not on the start value \( x_0 \) (with the exception of choosing an eigenvector as \( x - x_0 \)).

**Example 4.** The Gauss-Seidel iteration is a linear iterative solver. To estimate the convergence rate, choose the right hand side \( 0 \) and the start vector \( x_0 = 1 \). Then,

\[ \tilde{q} = \frac{\|x_{i+1}\|}{\|x_i\|} \]

is an approximate value of the convergence rate \( q \) for large values \( i \).

**Remark:** To avoid overflow and underflow, additionally normalize the vectors \( x_i \).
Estimation of the Convergence

We want to find a small parameter $q$ such that

$$\|x_{i+2} - x_{i+1}\| \leq q\|x_{i+1} - x_i\|.$$ 

Another way to estimate the convergence rate is to study the behavior of the residuum as follows: Let

$$r_i = Ax_i - b$$

- Calculate $\tilde{q} = \frac{\|r_{i+1}\|}{\|r_i\|}$ for large $i$.
- Calculate $\tilde{q} = \left(\frac{\|r_{i+s}\|}{\|r_i\|}\right)^{\frac{1}{s}}$ for large $i$, $s \approx 1 - 20$.

Take $\tilde{q}$ as an approximation of $q$. 
Meshsize and Iteration Number

Assume we want to obtain a total error $\|u - u_{h,i}\| \leq \eta$!

1. For every meshsize $h$ calculate $\tilde{q}$ by

$$\tilde{q} = \left( \frac{\|u_{i_{h}+s+1} - u_{i_{h}+s}\|}{\|u_{i_{h}+1} - u_{i_{h}}\|} \right)^{\frac{1}{s}}$$

for large $i_{h}$ and suitable $s \approx 1 - 20$.

2. Calculate $i_{h}$ such that

$$\|u_{h,i_{h}+1} - u_{h,i_{h}}\|(1 - \tilde{q})^{-1} \leq \frac{1}{4} \eta (1 - 2^{-p}).$$

3. Choose $h$ such that $\|u_{h,i_{h}} - u_{h/2,i_{h}/2}\| \leq \frac{1}{4} \eta (1 - 2^{-p})$.

Then, we obtain $\|u_{h} - u_{h/2}\| \leq \frac{3}{4} \eta (1 - 2^{-p})$ and thus

$$\|u - u_{h,i_{h}}\| \leq \eta.$$
Is there a Bug in the Code or not?

If a simulation program does not simulate a physical process in a correct way, there can be different reasons for this:

- inaccuracy of the model.
- error in the mathematical solver.
- error (bug) in the code.

There exist different bugs:

- syntax error,
- wrong usage of memory,
- logical sequence of the code is not correct, or
- the mathematical formula is not implemented in a correct way.
Debugging with gdb

Compile with option \texttt{-g} and execute \texttt{gdb} code.

- \texttt{b \ ln} Set breakpoint at line number \textit{ln}.
- \texttt{r} Run code.
- \texttt{s} Make one step.
- \texttt{S} Make one step and do not go into functions.
- \texttt{p u} Print \textit{u}.
- \texttt{b} Backtrace how the code went to a certain point in the code.
- \texttt{up} Go up the stack frame.
- \texttt{down} Go down the stack frame.
- \texttt{c} Continue running the code.
Apply intelligent print statements!

Instead of using the command \texttt{p} in \texttt{gdb} write your own intelligent print statements, which \texttt{gdb} does not contain. Example:

\begin{verbatim}
Print_L_infty(u);
\end{verbatim}

Prints the $L^\infty$ norm of a vector $u$. 
Finding a Bug in a Radio
Finding a Bug in a Radio
Finding a Bug in a Radio
Finding a Bug in a Radio
Hierarchical Search of Bug

Location of the bug
Reduction of the Problem

A very important concept is the concept to reduce a *big* problem to a *smaller* one. Of course the hierarchical search can be treated as such a concept. But there are also other ways to reduce a problem.

- Skip parts of the code in an hierarchical way such that the resulting code still contains the bug.
- Comment out statements in the code. As an example omit coarse grid correction in a multigrid code.
- Write a smaller code which contains the bug.
- Find a problem with a smaller problem size, such that the bug appears!
- Find a problem with known exact solution or a more simple solution!

If a reduction of the code is not possible any more, then analyze the code.
Call valgrind by
valgrind --tool=memcheck --leak-check=yes run

Use of valgrind for:
- finding causes for segmentation faults.
- finding memory leaks
Warnings in an HPC code

Warnings in a code are very useful to avoid bugs in a code.

class vector {
    public:
        vector(int dim_);
        double operator[](int i) {
            if(i < 0) cout << "i negative" << endl;
            if(i >= dim) cout << "i too large" << endl;
            return a[i];
        }
    private:
        int dim;
        double *a;
}

But this implementation of operator[](int i) is very inefficient.
Warnings in an HPC code

To increase performance implement a developer version as in the following example:

```cpp
#define developer_version true
// #define developer_version false

... double vector::operator[] (int i) {
  if(developer_version) {
    if(i< 0) cout << "'i negative'" << endl;
    if(i>=dim) cout << "'i too large'" << endl;
  }
  return a[i];
}
```

or one can use assert as follows:
Avoid $==$ Sign

Try to avoid the $==$ sign. Instead use $\geq$ or $\leq$. 
Avoid `double` Comparison

Example:

Instead of

```c
double x, h;
h = 1.0 / 10.0;
for(x=0.0; x<=1.0; x=x+h) {
    ...
}
```

write

```c
double x, h;
h = 1.0 / 10.0;
for(int i=0; i<10; ++i) {
    x = i*h;
    ...
}
```
Mathematical Error or Bug in the Code?

Often it is difficult to decide, if there is

- an inaccuracy in the model,
- an error in the mathematical solver, or
- a bug in the code.

This is one of the reasons why a simulation code must be
developed in several modules. Each of the modules must
be tested in detail.
Implement a test frame for modules. This frame gives a module certain input data and requires certain output data. If the output data are correct, then the module is expected to be correct.
Integration and Regression Test

- Using the test frame, apply the same tests for each module while developing the code and adding new modules.
  This is called *integration test*.

- Store results of your tests in a data file. Compare new test results with older test results.
  This is called *regression test*.
Finding Test Functions

The general concept is to calculate the right hand side for a given exact solution. These exact solution are the test functions.

There are different kinds of test functions:

- functions 0, 1, x, y, ...
- functions with special properties:
  - \( u = \sin(x \pi) \sinh(y \pi) \),\( u = e^t \sin(x \pi) \)
  - \( u = x^2 * y^3 \)
- symmetric solutions like \( u = x^5 * y^5 \).
- general functions. Calculate right hand side by a computer manipulation program (maple, mathematica).

First, test your code with the simplest one!
Test Parameters

Consider the PDE:

\[
\frac{\partial u}{\partial t} = -\Delta u + aw - f
\]

\[
\frac{\partial w}{\partial t} = -\Delta w + bu - g
\]

Parameters in a FD discretization are:

- physical parameters \(a, b\).
- meshsize \(h\), timestep \(\tau\).
- number of grid points \(N\), number of timesteps \(m\).

First, test your code for physically not correct parameters:

- \(a, b = 0, \pm 1, \pm 10, \ldots\)
- \(N = 1\) and \(m = 1, \ldots\)
Test Part of the PDE

Instead of

\[
\frac{\partial u}{\partial t} = -\Delta u + aw - f \\
\frac{\partial w}{\partial t} = -\Delta w + bu - g
\]

first test the stationary scalar equation:

\[-\Delta u + aw = f\]

and the stationary system:

\[-\Delta u + aw = f \quad \text{and} \quad -\Delta w + bu = g.\]
Test Convergence

- Test the convergence of your discretization for different test functions and parameters in the equation.
- Test the convergence rate of your iterative solver for different parameters.
Test of an Unstructured Grid Code

Assume there is a bug in your unstructured grid code with a complicated unstructured grid like:

→ To Calculate the matrix elements in each step of the code by hand is too complicated!
Test of an Unstructured Grid Code

To test your code let your unstructured grid generator generate a simple structured grid like

or

and test your code.

→ Change $x$ and $y$ coordinates and test symmetry of your code!
General Software Development

requirement spec.  What does the user want to have?

design specification  How to realize the requirements by software?

Implementation  Write the realization!

module test  Test the realization!

integration test  Test the design spec!

system test  Test the requirement spec!
Implement and Test

Build up your code step by step!

**Example:** Write a code for

\[-\Delta u + aw = f\]

and then for

\[-\Delta u + aw = f\]
\[-\Delta w + bu = g\]

and at last for:

\[
\frac{\partial u}{\partial t} = -\Delta u + aw - f \\
\frac{\partial w}{\partial t} = -\Delta w + bu - g.
\]

Implement one module and test it!
Types of Modules

- vector library (contains matrix multiplication, use libraries like LAPACK)
- grid generator
- linear equation solver
- calculation of stiffness matrix
- parallelization module
- input, output
- applications (different fluid dynamics applications)
Problems in PDE Software Development

- Black box solvers which are independent of the PDE and the discretization would be very helpful for the software development (algebraic multigrid (AMG), direct solver). But the optimal solver depends on the PDE and its discretization.

- Optimal solvers use the data structure of the discretization.

- Complicated data structure is needed for adaptive parallel solvers with load balancing.

- It is difficult to describe suitable interfaces between solvers.

- A clear software design often is in contradiction to efficiency. Therefore, expression templates and other template constructions are needed!
Consider the vector class

```cpp
class vector {
    public:
        vector(int l);
        double operator[](int i) { return p[i]; } ...
    private:
        int length;
        double *p;
};
```

How should we implement an operator

```cpp
vector operator+(vector &a, vector &b)
```
in an efficient way?
Example: vector class complex:

class complex {
  public:
    complex(double& re, double& im);
    ...
    double Re, Im;
};

complex operator+(complex &a, complex & b) {
  return complex(a.Re + b.Re, a.Im + b.Im);
}

In case of longer vectors introduce the length of the vector as a template parameter.
Vector Class for Long Vectors

class vector {
    public:
        vector(int l) { p = new double[l];
            length = l; }
        double operator[](int i) { return p[i]; }...
    ...

    private:
        int length;
        double *p;
    };

Problem:

- Should `vector operator+(vector &a, vector &b)` allocate an auxiliary vector?

- Efficient implementation of `c = a+b+d;` requires only one loop!
Realization of an Efficient Operator+

Implement `operator+` such that it gives back an object, which is able to add two vectors:

```cpp
class add_vector {
public:
    add_vector(double& *a, double& *b)
        : pa(a), pb(b) {};
    double operator[](int i) const
        { return pa[i] + pb[i]; }
...
private:
    double *pa, *pb;
};
```
Expression Template - Wrapper Class

To construct expression templates, we first need a wrapper class, which represents all possible expressions:

template<class A>
class DExpr {
private:
    A a_;  
public:
    DExpr(const A& x)
        : a_(x) {}  
double operator[](int i) const
        { return a_[i]; }  
};
template<class A, class B>
class DExprSum {
    const A a_;  const B b_;  
    public:
        DExprSum(const A& a, const B& b) : a_(a), b_(b) {}  
        double operator[](int i) const {
            return a_.[i] + b_.[i];   
        }
};

template<class A, class B>
DExpr<DExprSum<DExpr<A>, DExpr<B> > >
operator+(const DExpr<A>& a,const DExpr<B>& b) {
    typedef DExprSum<DExpr<A>, DExpr<B> > ExprT;  
    return DExpr<ExprT>(ExprT(a,b));
}
Properties of Expression

- efficient implementation by inlining.
- parallelization by OpenMP is possible.
- user friendly interface.
The expression
\[ d = a + b + c; \]
leads to the following expression tree:

\[
DExpr\langle DExprSum\langle DExpr\langle DExprSum\langle Dvector, Dvector \rangle \rangle, Dvector \rangle \rangle
\]
First Simplification

template<class A, class B, class Op>
class DExprBinOp {
    const A a_; const B b_;

public:
    DExprBinOp(const A& a, const B& b) : a_(a), b_(b) {}  
    double operator[](int i) const {
        return Op::apply(a_[i], b_[i]);
    };
};

class DApSum {

public:
    DApSum() { }
    static inline double apply(double a, double b)  
        { return a+b; }
};
Second Simplification

template <class A> struct Expr{
    inline const A& operator~() const{
        return static_cast<const A&>(*this);
    }
};

class vector : public Expr<vector> {
    public:
        ...
        template <class A>
        void operator=(const Expr<A>& a) {
            for(int i=0;i<length;++i) {
                p[i] = (~a)[i];
            }
        }
        ...
};
template <class A, class B>
class DExprSum : public Expr<DExprSum<A,B> >{
    const A& a_; const B& b_; 

public:
    DExprSum(const A& a, const B& b)
        : a_(a), b_(b) {}  
    double operator[](int i) const {
        return a_[i] + b_[i];   
    }
}

template <class A, class B>
inline DExprSum<A,B> operator+ (const Expr<A>& a, const Expr<B>& b)
    return DExprSum<A,B>(~a, ~b);
}
r = A * u - f;

d = -r;

delta = product(r, r);

for(i=1; i<=iteration && delta > eps; ++i) {
    
g = A * d;
    
    tau = delta / product(d, g);
    
    r = r + tau * g;
    
    u = u + tau * d;
    
    delta_prime = product(r, r);
    
    beta = delta_prime / delta;
    
    delta = delta_prime;
    
    d = beta * d - r;

}
Automatic parallelization means that only a change of the included library leads to a parallel code.
Automatic Parallelization

Automatic parallelization means that only a change of the included library leads to a parallel code.

Example:

```cpp
template <class A>
void vector::operator=(const Expr<A>& a) {
    #pragma omp parallel for
    for(int i=0; i<length; ++i) {
        p[i] = (~a)[i];
    }
}
```
Remark on Efficiency

In some cases a straight forward implementation of expression templates leads to less an efficient codes than a direct implementation. The reason is that the compiler cannot see a difference between expressions like

\[ a = b + b + b + b; \]

and

\[ a = b + c + d + e; \]

To avoid this problem one can construct enumerated variables.

```cpp
variable<1> a;
variable<2> b;
...;
```

Here the class `variable<n>` has an additional template parameter `n`. 
Expression Templates for Vectors

Construct operators for operations between
- vectors
- matrix and vector and
- matrices.

Blitz++ is such a library.
Let us assume that we want to perform finite difference operations on a 2D-structured grid $\Omega_h$.

Implement expression templates such that

$$u[I][J] = 0.25 \times (u[I+1][J] + u[I-1][J] + u[I][J+1] + u[I][J-1])$$

performs a red black Gauss-Seidel iteration for Poisson’s equation on $\Omega_h$. Here,

- $u$ a vector on the grid $\Omega_h$
- $u[I][J]$ represents $u(ih, jh)$
- $u[I+1][J]$ represents $u((i + 1)h, jh)$
- ...

Automatic parallelization of the above expression template implementation is possible.
A Jacobi-iteration for Poisson’s equation has to be implemented as follows:

\[
\begin{align*}
    r[I][J] &= 0.25 \times (u[I+1][J] + u[I-1][J] + u[I][J+1] + u[I][J-1]) \\
    u[I][J] &= r[I][J];
\end{align*}
\]
One also can implement an operator Laplace_FD(u) representing the mathematical operator

$$\frac{1}{h^2} \left( 4 \times u(ih, jh) - u((i+1)h, jh) - u(ih, (j+1)h) - u((i-1)h, jh) - u(ih, (j-1)h) \right).$$

Let Laplace_FD_diag() be the corresponding diagonal coefficient vector of Laplace_FD(u). Then, a Gauss-Seidel iteration for $-\Delta u = f$ can be implemented as follows:

```c
u = u - (Laplace_FD(u)+f) / Laplace_FD_diag();
```

and Jacobi by

```c
r = u - (Laplace_FD(u)+f) / Laplace_FD_diag();
u = r;
```
A Suitable Interface for PDE’s

Consider the following implementation of Gauss-Seidel:

\[ u[I][J] = 0.25 \times (u[I+1][J] + u[I-1][J] + u[I][J+1] + u[I][J-1]) ; \]

Problems:

- What is the range of \( I \) and \( J \)?
- How, to set values at the boundary?
- How, to implement boundary conditions?
A Suitable Interface for PDE’s

A suitable language for implementing PDE solvers is a current research topic. An optimal interface language is unknown up to now!

Suggestions:

- geometric objects - algebraic objects
- restriction operator to connect geometric objects and algebraic objects.
- vectors on grids and pure algebraic vectors.
Geometric objects:

- `vector3D Ma(0.0,2.0,1.0);`
- `vector3D Mb(0.0,0.0,1.0);`
- `Ball ball_a(1.0, Ma);`  // domain with radius 1.0 at point Ma
- `Ball ball_b(1.2, Mb);`  // domain with radius 1.2 at point Mb

```
... Domain domain = ball_a || ball_b;
```

Algebraic objects:

- `vector v1(1000), v2(1000), v3(1000);`
- `v3 = v1 + v2;`
Discretization Grid - Subgrids!

// Geometric objects:
Domain domain_a = ...; // Ω^a
Domain domain_b = ...; // Ω^b
Grid grid(domain_a, h); // ¯Ω^a
    // grid on domain_a with meshsize h
Subgrid subgrid(grid, domain_b); // ¯Ω^a ∩ Ω^b
Boundary_subgrid boundary(grid); // Γ^a_h = ¯Ω^a ∩ ∂Ω^a
Interior_subgrid interior(grid); // Ω^a_h
Boundary_subgrid Dirichlet(boundary, domain_b); // Γ^a_h ∩ Ω^b
Algebraic Vectors on a Grid

// Geometric objects:
Domain domain_a = ...; // Ω^a
Grid grid(domain_a, h); // Ω^a
    // grid on domain_a with meshsize h
Boundary_subgrid boundary(grid); // Γ^a_h = Ω^a_h ∩ ∂Ω^a
Interior_subgrid interior(grid); // Ω^a_h = Ω^a_h \ ∂Ω^a

// Variable: vector on a grid (algebraic vector with geometric information)
Variable u(&grid), f(&grid); // u, f ∈ R^|Ω^a_h|
coordinate_x X; coordinate_y Y; // coordinates
// Application of the restriction operator
u = X*X*Y*Y | boundary;
Poisson with Dirichlet Boundary Conditions

// Geometric objects:
Domain domain_a = ...; // \( \Omega^a \)
Grid grid(domain_a,h); // \( \bar{\Omega}^a_h \)
   // grid on domain_a with meshsize h
Boundary_subgrid boundary(grid); // \( \Gamma^a_h = \bar{\Omega}^a_h \cap \partial \Omega^a \)
Interior_subgrid interior(grid); // \( \Omega^a_h = \bar{\Omega}^a_h \setminus \partial \Omega^a \)

// Variable: vector on a grid (algebraic vector with geometric information)
Variable u(&grid), f(&grid); // \( u, f \in \mathbb{R}^{\bar{\Omega}^a_h} \)
coordinate_x X; coordinate_y Y; // coordinates

// Application of the restriction operator
u = X*X*Y*Y | boundary;
f = -2*(X*X+Y*Y) | interior;
for(int i=1;i<50;++i)
   u = u-(Laplace_FD(u)+f) / Laplace_FD_diag() | interior;
Parallelization Concepts

One can distinguish the following parallelization concepts:

- Shared memory parallelization
  - Parallelization with one main memory and several different processors
  - NUMA architecture (Non-Uniform Memory Access).

- Distributed memory parallelization

- Hybrid parallelization with a shared memory and a distributed memory

- Vectorization. One processor can perform parallel computations on long vectors.
MPI - Message Passing Interface

- MPI is a library language for C, C++ and FORTRAN.
- There exist different MPI libraries. MPICH and MPI-LAM are one of them.
- The MPI library is included by mpi.h.

Run an MPI code by

```bash
mpirun -np p code
```

Here \( p \) is the number of processors.
MPI - Message Passing Interface

- Every processor runs the same program with a different *rank*.
- Data are sent by MPI-functions from one processor to the other. All MPI-functions have the prefix `MPI_`.
- Data are sent from one processor to the other of a certain *communicator*. The *rank* of the processor depends on the communicator. Here, we use only the communicator `MPI_COMM_WORLD` which is of type `MPI_Comm`. 


Let us describe the most elementary MPI functions:

```c
int MPI_Init(int *argc, char ***argv);
int MPI_Comm_size(MPI_Comm comm, int *size);
int MPI_Comm_rank(MPI_Comm comm, int *rank);
int MPI_Comm_Finalize();
```

`size` is the total number $p$ of processors and `rank` the number from $0, \ldots, p - 1$.

The return value of these function is an information about the error. This will be discussed later.
MPI_Bcast and MPI_Reduce

MPI_Bcast sends data from processor with number root to all other processors.

MPI_Reduce applies an operation to data of all processors. The result is sent to root.

```c
int MPI_Bcast(void *buf, int count, MPI_Datatype datatype,
              int root, MPI_Comm comm);
int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op,
               int root, MPI_Comm comm);
```

Pointers (like buf) point to arrays of type datatype and length count. Possible data types for MPI_Datatype are:

- MPI_INT
- MPI_DOUBLE
- MPI_LONG
- MPI_CHAR
- ...

– p. 158/249
Example Numerical Integration

The trapezoidal rule is the following rule for numerical integration:

\[ \int_0^1 f(x) \, dx \approx h \sum_{i=1}^{n} f\left(h(i - 0.5)\right) \]

where \( h = \frac{1}{n} \). To parallelize this formula let us assume that \( n = kp \), where \( p \) is the number of processors. Then, we get

\[ \int_0^1 f(x) \, dx \approx \sum_{j=1}^{p} h \sum_{i=1}^{k} f\left(h(((j - 1)k + i) - 0.5)\right) \]
Send and Receive with Blocking

MPI_send sends data to the processor with destination rank dest and with tag (german: Anhänger, Etikett): tag. Valid tags are values from 0 to 32767.

MPI_Recv receives data from processor with source rank source. This function returns the status status.

```c
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag,
             MPI_Comm comm, MPI_Status *status);
```

status provides the following informations:

- status.MPI_SOURCE
- status.MPI_TAG

For other functions, status can provide informations about the error.
Example for Send and Receive

```c
if (my_rank != 0)
    MPI_Send(&my_integral, 1, MPI_DOUBLE, 0, 10 + my_rank, MPI_COMM_WORLD);
else {
    double source_integral;
    MPI_Status status;

    integral = my_integral
    for (int source = 1; source < p; ++source) {
        MPI_Recv(&source_integral, 1, MPI_DOUBLE, source, 10 + source, MPI_COMM_WORLD, &status);
        integral = integral + source_integral;
        cout << " I got message from: " << source << endl;
    }
}
```
if (my_rank != 0)
    MPI_Send(&my_integral, 1, MPI_DOUBLE, 0,
             10 + my_rank, MPI_COMM_WORLD);
else {
    double source_integral;
    MPI_Status status;

    integral = my_integral
    for (int source = 1; source < p; ++source) {
        MPI_Recv(&source_integral, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
                  MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        integral = integral + source_integral;
        cout << " I got message from: " << status.MPI_SOURCE << endl;
    }
}
Send and Receive without Blocking

int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Wait(MPI_Request *request, MPI_Status *status);
int MPI_Waitall(int count, MPI_Request *array_of_request, MPI_Status *array_of_statuses);
double *source_integral;

MPI_Request *req; MPI_Status *status;

req = new MPI_Request[p-1];
status = new MPI_Status[p-1];

source_integral = new double[p];

if(my_rank!=0) {
    MPI_Isend(&my_integral,1, MPI_DOUBLE,0,
               10+my_rank, MPI_COMM_WORLD,&req[0]);
    MPI_Waitall(1,req,status); }

else {
    source_integral[0] = my_integral;
    for(int source=1;source<p;++source) {
        MPI_Irecv(&source_integral[source],1, MPI_DOUBLE,source,
                  10+source, MPI_COMM_WORLD, &req[source-1]);
    }
    MPI_Waitall(p-1,req,status);
}

integral=0.0;
Error Handlers

There are two different error handlers:

- **MPI_ERRORS_ARE_FATAL** (default): This error handler forces to abort all MPI processes.
- **MPI_ERRORS_RETURN**: Now, the MPI-function returns an error information.

One can set the handler **MPI_ERRORS_RETURN** by

```c
MPI_Errhandler_set(MPI_COMM_WORLD, MPI_ERRORS_RETURN);
```
Let `errcode` be a return value of an MPI-function. Then,

- `errcode==MPI_SUCCESS` (This means there is no error.), or

- `errcode` can be decoded by

  ```c
  int MPI_Error_class(int errcode,
                      int *errorclass)
  ```

Possible values for `*errorclass` depend on the MPI implementation. In MPI-1 the following classes are defined:

- `MPI_SUCCESS`
- `MPI_ERR_RANK`
- `MPI_ERR_BUFFER`
- `...`
Test Incoming Message

Sometimes a process would like to know, whether there is a process sending a message. This can be tested by MPI_Iprobe.

Example

```c
if (my_rank == 0) {
    MPI_Status status; int flag = false;
    MPI_Iprobe(MPI_ANY_SOURCE, 2, MPI_my_rank, &flag, &status);
    if (flag == true) {
        int rank_from = status.MPI_SOURCE;
        MPI_Recv(buffer, num_data, MPI_DOUBLE, rank_from, 2, MPI_my_rank, &status);
    }
}
```
A parallel debugger is *totalview*. The running state on every processor is reported on a different window.
Parallelization of PDE-Solvers

Let us assume that a PDE is discretized on the discretization grid $\Omega_h$. A distributed memory parallelization of algorithms on $\Omega_h$ is based on a partition of $\Omega_h$:

$$\Omega_h = \bigcup_{i=1}^{p} \Omega_{ih}.$$

An optimal partitioning depends on
- the “sequential flow” of the algorithm,
- the amount of data to be sent, and
- the amount of computations, which have to be performed on each partition. This computational amount should be balanced on the partitions (load balancing).
Optimal Partitioning for Relaxation Methods

point approach

cell approach
Cell Partitioning

\[ \Omega_h = \{(ih, jh) \mid i, j = 0, \ldots, N-1\}, \]

where \( h = \frac{1}{N-1} \) and

\[ N = \sqrt{pn}, n, N \in \mathbb{N}. \]

\[ \Omega_{h}^{k,s} = \{((kn + i)h, (sn + j)h) \mid i, j = 0, \ldots, n - 1\}, \]

where \( k, s = 0, \ldots, \sqrt{p} - 1 \). Then,

\[ \Omega_h = \bigcup_{k,s=0}^{\sqrt{p}-1} \Omega_{h}^{k,s}. \]
Cell Partitioning

For the evaluation of stencil operators, data of points on neighbor processors are needed. These are the data at ghost points:

\[ \hat{\Omega}_{h}^{k,s} \backslash \Omega_{h}^{k,s} \]

where

\[ \hat{\Omega}_{h}^{k,s} = \left\{ (kn + i)h, (sn + j)h \right\} \mid i, j = -1, \ldots, n \} \cap \Omega_{h} \]

for \( k, s = 0, \ldots, \sqrt{p} - 1 \).
In a Jacobi iteration, data have to be sent and received from neighbor processors. Let $N, S, NW, \ldots$ be the indices of the neighbor processor with index $M = (k, s)$.

Then, before every Jacobi iteration the data at points $\Omega^M_h \cap \Omega^P_h$ have to be sent from processor $P$ to processor $M$.

Let us denote this procedure $\text{Send}(P)$;
Example Code

Implement first `MPI_Irecv` then `MPI_Isend`!

```c
num_message = 0;
if(rank_source != -1 && number_receive>0) {
    MPI_Irecv(receive_info ,number_receive,
              MPI_DOUBLE,rank_source,26,comm,
              &req[num_message]);
    ++num_message;
}
if(rank_destination != -1 && number_send>0) {
    MPI_Isend(send_info,number_send,
              MPI_DOUBLE,rank_destination,26,comm,
              &req[num_message]);
    ++num_message;
}
MPI_Waitall(num_message,req,status);
```
Two Sending Approaches

1. Approach

Send(E); Send(W); Send(N); Send(S);
Send(NE); Send(NW); Send(SE); Send(SW);
Waitall();

2. Approach

Send(E); Send(W);
Waitall();
Send_(N); Send_(S);
Waitall();

This approach updates data also from NE, NW, ..., if Send_ also sends the updated data from processor E, W.
Sending for 4 Color Gauss-Seidel
Point Partitioning

point approach

\[ \Omega_h = \{(ih, jh) \mid i, j = 0, ..., N \}, \]

where \( h = \frac{1}{N} \) and \( H = \frac{1}{\sqrt{p}} \)

\( N = \sqrt{pn}, n, N \in \mathbb{N}. \) Define

\[ \bar{\Omega}^{k,s} = [Hk, H(k + 1)] \times [Hs, H(s + 1)] \]
\[ \hat{\Omega}^{k,s} = [Hk, H(k + 1)[\times[Hs, H(s + 1)[ \]

\[ \Omega^{k,s}_h = \Omega_h \cap \left( \bar{\Omega}^{k,s} \setminus \bigcup_{(k', s') \neq (k, s)} \hat{\Omega}^{k',s'} \right). \] Then,

\[ \Omega_h = \bigcup_{k,s=0}^{\sqrt{p}-1} \Omega^{k,s}_h. \]
Load Balancing

\[ \Omega_h = \{(ih, jh) \mid i, j = 0, \ldots, N - 1\}, \]

where \( h = \frac{1}{N} \). Let \( p = p_1p_2 \).

Make a partitioning with \( p_1 \) processors in x-direction and \( p_2 \) processors in y-direction.

- Same load balancing for every processor.
- \( D_{\text{send}} = 2 \frac{N}{p_1} + 2 \frac{N}{p_2} = 2N\left(\frac{1}{p_1} + \frac{1}{p_2}\right) \) data to be sent.
Let us consider the cg iteration:

\[ r = A \times u - f; \]
\[ d = -r; \]
\[ \text{delta} = \text{product}(r, r); \]
\[ \text{for}(i=1; i<=\text{iteration} && \text{delta} > \text{eps}; ++i) \{ \]
\[ g = A \times d; \]
\[ \text{tau} = \text{delta} / \text{product}(d, g); \]
\[ r = r + \text{tau} \times g; \]
\[ u = u + \text{tau} \times d; \]
\[ \text{delta}_\text{prime} = \text{product}(r, r); \]
\[ \text{beta} = \text{delta}_\text{prime} / \text{delta}; \]
\[ \text{delta} = \text{delta}_\text{prime}; \]
\[ d = \text{beta} \times d - r; \]
\[ \}

When is an update of ghost values needed?
enum Update_typ { no_update, update };
class vector : public Expr<vector> {
    public:
        vector(int l) { update_var = no_update; };
        Update_typ expression_update_typ() const {
            return update_var; }
    private:
        Update_typ update_var;
        int id;
    ... }

Update_typ DExprSum::expression_update_typ() const {
    return a_.expression_update_typ() ||
    b_.expression_update_typ() ];
Update_typ DExprLaplace_FD::expression_update_typ() const {
    return update; };

MPI with Expression Templates
class Update_handler;

template <class A>
void vector::operator=(const Expr<A>& a) {
    if((~a).expression_update_type()) {
        Update_handler handler_update;
        (~a).Give_update_data(handler_update);
        handler_update.Make_update();
    }
    for(int i=0; i<length; ++i) {
        p[i] = (~a)[i];
    }
}
Raytracing

Raytracing is used in

- Computer graphics: How does light look at an image plane?
- Simulation of light in engineering applications: How is light absorbed in a medium (example: laser crystal).

The main idea of ray tracing is that light is modeled by several rays of light.
Forward and Backward Raytracing

- **Forward Raytracing**: Light propagates from a light source in several directions until either vanishes by absorption or it impings at the image plane or leaves out of the computational domain.

- **Backward Raytracing**: Find the rays which impinging at the image plane by back tracing rays beginning from all points of the image plane in all possible directions.
Concept of Forward Raytracing

A ray starts at a point \( P \) and propagates in direction \( \vec{d} \) with intensity \( I \). The path of the ray can be described by

\[
P + \lambda \vec{d}, \quad \lambda \in \mathbb{R}
\]

The following situations can happen:

- The ray propagates out of the computational domain.
- The ray impings at an object and vanishes.
- The ray impings at an objects and is reflected in one or more directions.
- The ray propagates from a medium A to medium B with different refraction indices.
- Light of the ray is absorbed while propagating through a medium.
Ray out of the Computational Domain

computational domain
Ray Impings on Object
Perfect Specular Reflection

In case of perfect specular reflection, there is only one reflected ray which satisfies:

\[ \alpha_{in} = \alpha_{out}. \]
Perfect Diffuse Reflection

The Lambert reflection describes a diffusive reflection of light by several rays:

![Diagram showing diffuse reflection]
Perfect Specular Transmission

Perfect specular transmission satisfies Snell’s law:

\[ \alpha_{in} \cdot n_A = \alpha_t \cdot n_B. \]
Perfect Diffusive Transmission
General Situation

Diagram showing the general situation with vectors $n_A$, $n_B$, and $a_{in}$, and angle $\alpha_t$. Point P is indicated in the diagram.
Light Sources

There exist different kinds of light sources:

- point light source
- multimode light source
- Gaussian beam light of low order (not multimode).

→ This kind of light cannot be modeled by ray tracing.
Point Light Sources

one direction

several directions
Multimode Light Sources

Set of rays starting at points $P_i$, $i = 1, ..., n$ to every direction with angle $\phi$ between $-\alpha$ and $\alpha$:

The numerical aperture NA is defined by:

$$NA = n_r \cdot \sin(\alpha),$$

where $n_r$ refraction index of the medium. Example: Light of multimode fiber.
Discretization of Light Source

Assume that a light source consists of an infinite number of rays starting at points $P_i \in \Omega_{source}$ in directions $\vec{d}_i \in \Phi_{P_i}$. Assume that the intensity of the light source is constant close to the light source.
To discretize the light source, we approximate the light source by a finite number of rays:

N rays starting at $P_i$ in direction $\vec{d}_i$,
where $i = 1, \ldots, N$.

If the total power of the light source is $I$, then the power of each discretized ray is $I/N$.
Often, the starting points $P_i$ and the directions $\vec{d}_i$ can be chosen by random numbers.
Assume that $P_i \in \Omega_{source} \subset [a_x, b_x] \times [a_y, b_y]$ and
$\vec{d}_i \in \Phi_{P_i} = [a_\phi, b_\phi]$.

Then, random values for $P_i$ and the directions $\vec{d}_i$ can be
constructed by a random number generator for an interval $[a, b]$. 
Absorption of Light

Assume that light propagates through absorbing medium.

Discretize absorbing medium by cells of meshsize $h$. The power of light absorbed in a cell is:

$$P_{abs}(cell) = P(A)(1 - \exp(-\alpha \overline{AB})).$$
Let us assume that the ODE

\[ y'(t) = f(t, y(t)), \quad t \geq t_0 \]

\[ y(t_0) = y_0 \]

is given, where \( y : [t_0, \infty[ \rightarrow \mathbb{R}^n \).

To discretize this ODE, let \( \tau > 0 \) be a time step.

Let us denote \( y_i \) the approximation of \( y(t_i) \), where \( t_i := \tau i + t_0 \).

Types of solvers:

- simplest method: Euler method
- Runge Kutta methods (one step method)
- multi-step methods
- implicit, explicit methods
Examples

- explicit Euler: $y_{i+1} = y_i + \tau f(t_i, y_i)$.
- implicit Euler: $y_{i+1} = y_i + \tau f(t_{i+1}, y_{i+1})$.
- classical Runge Kutta method
  
  \[ k_1 = \tau f(x_i, y_i) \]
  \[ k_2 = \tau f(x_i + 1/2\tau, y_i + 1/2k_1) \]
  \[ k_3 = \tau f(x_i + 1/2\tau, y_i + 1/2k_2) \]
  \[ k_4 = \tau f(x_i + \tau, y_i + k_3) \]
  \[ y_{i+1} = y_i + 1/6k_1 + 1/3k_2 + 1/3k_3 + 1/6k_4. \]

- Simpson’s method:
  
  \[ y_{i+1} - y_{i-1} = \frac{\tau}{3} (f(t_{i+1}, y_{i+1}) + 4f(t_i, y_i) + f(t_{i-1}, y_{i-1})]. \]

- Middle point method: $y_{i+1} - y_{i-1} = 2\tau f(t_i, y_i)$. 

– p. 199/249
Stability of a Multi-Step Method

To analyze the stability of a multi-step method of length $s$, consider the ODE

$$y' = 0, \quad y(0) = y_0.$$ 

Assume that the multi-step method leads to the recursion formula

$$\sum_{i=0}^{s} a_i y_{i+j} = 0 \quad \forall j \in \mathbb{N}_0.$$ 

for this ODE.
Stability of a Multi-Step Method

Definition 3. The multi-step method is stable, if for all start values $y_0, ... y_{s-1}$, the sequence $y_i$ is bounded.

Theorem 3. A multi-step method is stable if all roots of the polynomial

$$\sum_{i=0}^{s} a_i z^i$$

are simple roots and contained in the disc

$$\{ z \in \mathbb{C} \mid |z| \leq 1 \}.$$

(A more general stability theorem is given in Stoer/Burlisch, Einführung in die Numerische Mathematik II).
Stiff ODE’s

Let us linearize $f(y, t')$ at a certain point $\hat{t}, \hat{y}$ by Taylor series in $y$ direction:

$$f(\hat{y}, \hat{t}) \approx b + A(y - \hat{y}).$$

The ODE is a stiff ODE, if $A$ has negative eigenvalues of different size.

**Definition 4.** *The ODE solver is a stable ODE solver for stiff equation systems, if*

$$\lim_{i \to \infty} y_i = 0 \quad \forall \tau > 0 \quad \text{and} \quad y_i > 0 \quad \forall \tau > 0, \ i \in \mathbb{N}$$

*for the ODE*

$$y' = \lambda y, \quad y(0) = 1,$$

*where $\lambda < 0$.***
Analysis of ODE Solvers

Apply a given ODE solver to the ODE

\[ y' = \lambda y, \quad y(0) = 1. \]

Often this leads to an iteration formula of the form

\[ y_{i+1} = y_i g(\lambda \tau). \]

Then, stability means

\[ |g(z)| < 1 \quad \forall \text{Re}(z) < 0. \]
Examples

- explicit Euler: (not stable for stiff ODE’s)

\[ y_{i+1} = y_i + hf(t_i, y_i) \]

- implicit Euler: (stable for stiff ODE’s)

\[ y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}). \]
Let $\Omega \subset \mathbb{R}^d$ be a domain. The standard parabolic PDE is:

$$\frac{\partial u}{\partial t} = \alpha^2 \Delta u + f(t, \vec{x}), \quad \vec{x} \in \Omega, t \geq t_0,$$

initial condition

$$u(t_0, \vec{x}) = u_0(\vec{x}), \quad \vec{x} \in \Omega,$$

boundary condition

$$u(t, \vec{x}) = g(t, \vec{x}), \quad \vec{x} \in \partial \Omega, t \geq t_0,$$

where, $g, f, u_0$ are given functions.
Discretization of a Parabolic PDE

- Let $\bar{\Omega}_h \subset \bar{\Omega}$ be a discretization grid.
  Let $\Omega_h = \bar{\Omega}_h \cap \Omega$.

- $t_i := r_i + t_0$.

- Let us denote $\bar{u}_h(t_i, \bar{x}_h), i \in \mathbb{N}_0, \bar{x}_h \in \Omega_h$ the approximate solution.
  Furthermore, let us abbreviate $u_h(t_i) = (\bar{u}_h(t_i, \bar{x}_h))_{\bar{x}_h \in \Omega_h}$.

- Let us discretize $\triangle w$ by
  $$\bar{L}_h w_h,$$
  where $\bar{L}_h$ is a $|\Omega_h| \times |\bar{\Omega}_h|$ matrix and $w_h \in \mathbb{R}^{|\Omega_h|}$.
  (e.g. finite difference discretization).

- In case of homogeneous boundary conditions ($g = 0$) $\bar{L}_h$ can be replaced by the $|\Omega_h| \times |\Omega_h|$ matrix $L_h$. 
Discretization of a Parabolic PDE

\[ u_h(t_0, \vec{x}_h) = u_0(\vec{x}_h), \quad \vec{x}_h \in \bar{\Omega}_h \]
\[ u_h(t_i, \vec{x}_h) = g(t_i, \vec{x}_h) \quad \vec{x}_h \in \bar{\Omega}_h \setminus \Omega_h, \quad i \in \mathbb{N}_0. \]

- forward difference method (explicit Euler)

\[ \bar{u}_h(t_{i+1}) = \bar{u}_h(t_i) + \tau \left( \alpha^2 \bar{L}_h \bar{u}_h(t_i) + f_h(t_i) \right). \]

- backward difference method (implicit Euler)

\[ \bar{u}_h(t_{i+1}) = \bar{u}_h(t_i) + \tau \left( \alpha^2 \bar{L}_h \bar{u}_h(t_{i+1}) + f_h(t_{i+1}) \right). \]

- Crank-Nicolson

\[ \bar{u}_h(t_{i+1}) = \bar{u}_h(t_i) + \frac{\tau}{2} \left( \alpha^2 \bar{L}_h \bar{u}_h(t_i) + f_h(t_i) + \alpha^2 \bar{L}_h \bar{u}_h(t_{i+1}) + f_h(t_{i+1}) \right). \]
Fourier Stability Analysis

To analyze the stability of the previous discretization, let us consider

\[
\frac{\partial u}{\partial t} = \alpha^2 \Delta u, \quad \vec{x} \in \Omega, \, t \geq t_0, \\
u(t_0, \vec{x}) = u_0(\vec{x}), \quad \vec{x} \in \Omega, \quad \text{initial condition} \\
u(t, \vec{x}) = 0, \quad \vec{x} \in \partial \Omega, \, t \geq t_0, \quad \text{boundary condition} \\
\Omega := ]0, \pi[^2.
\]

The exact solution of this PDE is

\[
u(t, (x, y)) = \sum_{\nu,\mu=0}^{\infty} a_{\nu,\mu} \sin(\nu x) \sin(\mu y) \, e^{-\alpha^2(\nu^2+\mu^2)(t-t_0)}, \quad \text{where}
\]

\[
u_0(x, y) = \sum_{\nu,\mu=0}^{\infty} a_{\nu,\mu} \sin(\nu x) \sin(\mu y).
\]

Observe that \(u(t, (x, y)) \geq 0,\) if \(a_{\nu,\mu} \geq 0 \forall \nu, \mu.\)
Fourier Stability Analysis

Let $\Omega_h := \{ h(i, j) \mid i, j = 1, m - 1 \}$ be the discretization grid.

Lemma 1. $L_h$ has the eigenvectors

$$e_{\nu,\mu} = \left( \sin(\nu \pi x_i) \sin(\mu \pi y_j) \right)_{(x_i, y_j) \in \Omega_h},$$

where $\nu, \mu = 1, \cdots, m - 1$,

with eigenvalues

$$\lambda_{\nu,\mu} = -\frac{4}{h^2} \left( \sin^2 \left( \frac{\pi \nu h}{2} \right) + \sin^2 \left( \frac{\pi \mu h}{2} \right) \right).$$

The eigenvalues can be estimated by

$$\frac{8}{h^2} > -\lambda_{\nu,\mu} > 2\pi^2.$$
The functions \((e_{\nu,\mu})_{\nu,\mu=1,...,m-1}\) form a basis of \(\mathbb{R}^{\Omega_h}\). Thus we can write

\[
    u_0 = \sum_{\nu,\mu=1}^{m-1} c_{\nu,\mu}(t_0) e_{\nu,\mu}.
\]

**Definition 5.** The discretization of the parabolic equation is stable, if the following condition holds:

Let the coefficients \(c_{\nu,\mu}(t_0)\) be nonnegative.

Then, the coefficients of the approximate solution for \(f = 0, g = 0\) are nonnegative

\[
    c_{\nu,\mu}(t) \geq 0 \quad \forall \nu, \mu, \ t > t_0.
\]
The Fourier analysis of the forward difference method

\[ u_h(t_{i+1}) = u_h(t_i) + \tau (\alpha^2 L_h u_h(t_i) + f_h(t_i)) \]

leads to the explicit Euler formula \((f = 0, g = 0)\):

\[ c_{\nu,\mu}(t_{i+1}) = (1 + \tau \alpha^2 \lambda_{\nu,\mu}) c_{\nu,\mu}(t_i) \]

Stability is obtained if \(|1 + \tau \alpha^2 \lambda_{\nu,\mu}| < 1\) and therefore

\[ \tau < \frac{2}{\alpha^2 |\lambda_{\nu,\mu}|} \]

Thus the condition

\[ \tau < \frac{2}{\alpha^2 \frac{8}{h^2}} = \frac{2h^2}{8\alpha^2} \]
The analysis of the backward difference method

\[ u_h(t_{i+1}) = u_h(t_i) + \tau (\alpha^2 L_h u_h(t_{i+1}) + f_h(t_{i+1})) \]

leads to the implicit Euler formula \((f = 0, g = 0)\):

\[ c_{\nu,\mu}(t_{i+1}) = c_{\nu,\mu}(t_i) \frac{1}{1 - \tau \alpha^2 \lambda_{\nu,\mu}}. \]

Stability is obtained independent of \(\tau\) since

\[ 0 < \frac{1}{1 - \tau \alpha^2 \lambda_{\nu,\mu}} < 1. \]
Analysis of Crank-Nicolson

The analysis of Crank-Nicolson

\[ u_h(t_{i+1}) = u_h(t_i) + \tau \frac{1}{2} \left( \alpha^2 \bar{L}_h u_h(t_i) + f_h(t_i) + \alpha^2 \bar{L}_h u_h(t_{i+1}) + f_h(t_{i+1}) \right) \]

leads to the formula \((f = 0, \ g = 0)\):

\[ c_{\nu,\mu}(t_{i+1}) = c_{\nu,\mu}(t_i) \frac{1 + \frac{1}{2} \tau \alpha^2 \lambda_{\nu,\mu}}{1 - \frac{1}{2} \tau \alpha^2 \lambda_{\nu,\mu}}. \]

Stability is obtained independent of \(\tau\) since

\[ \left| \frac{1 + \frac{1}{2} \tau \alpha^2 \lambda_{\nu,\mu}}{1 - \frac{1}{2} \tau \alpha^2 \lambda_{\nu,\mu}} \right| < 1. \]

But for large \(|\alpha^2 \lambda_{\nu,\mu}|: \left| \frac{1 + \frac{1}{2} \tau \alpha^2 \lambda_{\nu,\mu}}{1 - \frac{1}{2} \tau \alpha^2 \lambda_{\nu,\mu}} \right| \to 1. \]
Hyperbolic PDE

Let $\Omega \subset \mathbb{R}^d$ be a domain.
The standard hyperbolic PDE is:

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \Delta u + f(t, \vec{x}), \quad \vec{x} \in \Omega, t \geq t_0,$$

$$u(t_0, \vec{x}) = u_0(\vec{x}), \quad \vec{x} \in \Omega,$$

$$\frac{\partial u}{\partial t}(t_0, \vec{x}) = u_1(\vec{x}), \quad \vec{x} \in \Omega,$$

$$u(t, \vec{x}) = g(t, \vec{x}), \quad \vec{x} \in \partial \Omega, t \geq t_0,$$

where, $g, f, u_0, u_1$ are given functions.
Discretization of a Hyperbolic PDE

- Let $\tilde{\Omega}_h \subset \tilde{\Omega}$ be a discretization grid.
  Let $\tilde{\Omega}_h = \Omega_h \cap \Omega$.

- $t_i := \tau i + t_0$.

- Let us denote $\tilde{u}_h(t_i, \tilde{x}_h)$, $i \in \mathbb{N}_0$, $\tilde{x}_h \in \Omega_h$ the approximate solution.
  Furthermore, let us abbreviate $u_h(t_i) = (\tilde{u}_h(t_i, \tilde{x}_h))_{\tilde{x}_h \in \Omega_h}$.

- Let us discretize $\triangle w$ by
  \[
  \bar{L}_h w_h,
  \]
  where $\bar{L}_h$ is a $|\Omega_h| \times |\Omega_h|$ matrix and $w_h \in \mathbb{R}^{|\Omega_h|}$.
  (e.g. finite difference discretization).

- In case of homogeneous boundary conditions ($g = 0$)
  $\bar{L}_h$ can be replaced by the $|\Omega_h| \times |\bar{\Omega}_h|$ matrix $L_h$. 
Discretization of a Hyperbolic PDE

First initial condition and boundary condition:

\[
\begin{align*}
    u_h(t_0, \vec{x}_h) &= u_0(\vec{x}_h) & \vec{x}_h &\in \bar{\Omega}_h \\
    u_h(t_i, \vec{x}_h) &= g(t_i, \vec{x}_h) & \vec{x}_h &\in \bar{\Omega}_h \setminus \Omega_h, \; i \in \mathbb{N}_0.
\end{align*}
\]

Second initial condition:

\[
    u_h(t_1, \vec{x}_h) = u_0(\vec{x}_h) + \tau u_1(\vec{x}_h) + \frac{1}{2} \tau^2 \alpha^2 \Delta u_0(\vec{x}_h).
\]

Discretization of the PDE:

\[
    u_h(t_{i+1}) = 2u_h(t_i) - u_h(t_{i-1}) + \tau^2 \alpha^2 \left( \bar{L}_h u_h(t_i) + f_h(t_i) \right).
\]
To analyze the stability of the previous discretizations, let us consider the case \( f = 0, g = 0, u_1 = 0 \). Then, the exact solution of this PDE is

\[
\begin{align*}
    u(t, (x, y)) &= \sum_{\nu, \mu=0}^{\infty} a_{\nu, \mu} \sin(\nu x) \sin(\mu y) \cos \left( \alpha(t - t_0) \sqrt{\nu^2 + \mu^2} \right), \\
    u_0(x, y) &= \sum_{\nu, \mu=0}^{\infty} a_{\nu, \mu} \sin(\nu x) \sin(\mu y).
\end{align*}
\]

Observe that

\[
a_{\nu, \mu} \sin(\nu x) \sin(\mu y) \cos \left( \alpha(t - t_0) \sqrt{\nu^2 + \mu^2} \right)
\]

is bounded for \( t \to \infty \).
The functions \((e_{\nu,\mu})_{\nu,\mu=1,...,m-1}\) form a basis of \(\mathbb{R}^{|\Omega_h|}\). Thus, we can write

\[
u,\mu=1\]
\[
\begin{align*}
u,\mu=1
u_0 &= \sum_{\nu,\mu=1}^{m-1} c_{\nu,\mu}(t_0) e_{\nu,\mu}.
\end{align*}
\]

**Definition 6.** The discretization of the hyperbolic equation is stable, if the following condition holds:

Assume that

\[
\begin{align*}
\begin{cases}
c_{\nu,\mu} = \begin{cases} c \neq 0 & \text{for } (\nu, \mu) = (\nu', \mu') \\
0 & \text{for } (\nu, \mu) \neq (\nu', \mu')
\end{cases}
\end{cases}
\end{align*}
\]

Then, the approximate solution for \(f = 0, g = 0, u_1 = 0\) is bounded for \(t \to \infty\).
Analysis of the Discretization

The Fourier analysis of the discretization

\[ u_h(t_{i+1}) = 2u_h(t_i) - u_h(t_{i-1}) + \tau^2 \alpha^2 \left( \bar{L}_h u_h(t_i) + f_h(t_i) \right) \]

leads to the formula \((f = 0, \ g = 0, \ u_1 = 0)\):

\[ c_{\nu,\mu}(t_{i+1}) = (2 + \tau^2 \alpha^2 \lambda_{\nu,\mu})c_{\nu,\mu}(t_i) - c_{\nu,\mu}(t_{i-1}). \]

Stability is obtained, if the roots of

\[ z^2 - (2 + \tau^2 \alpha^2 \lambda_{\nu,\mu})z + 1 \]

are simple and contained in the disc \( \{ z \in \mathbb{C} \mid |z| \leq 1 \} \). This implies the CFL (Courant, Friedrich, Lewy) condition:

\[ \tau < \frac{1}{\sqrt{2}} \frac{h}{|\alpha|}. \]
Order of Consistency and Convergence

- There are slightly different definitions of consistency for different types of ODE solvers and types of PDE’s.
- There are different definitions for stability.
- In numerical analysis one proves: 
  \[ \text{consistency} + \text{stability} \Rightarrow \text{convergence} \]
Definition 7. Let $L(u)$ be a differential operator on $\Omega$ and $L_h(u_h)$ a discrete approximation of this operator on the discretization grid $\Omega_h$. Furthermore, let $R_h : C(\overline{\Omega}) \to \mathbb{R}^{\Omega_h}$ be the pointwise restriction operator. Then, the consistency order of $L_h$ is of order $O(h^p)$, if there exists a constant $C > 0$ such that

$$\| R_h(L(u)) - L_h(R_h(u)) \| \leq C h^p.$$ 

Example 5. Consider the differential operator $\frac{\partial}{\partial x}$. The consistency order of central differences is $O(h^2)$ and the consistency order of upwind or downwind differences is $O(h)$. 
Definition 8. Let $y' = f(t, y)$ be an ODE on the domain $[t_0, \infty[$.

Let $y_i \rightarrow \Psi(y_i) = y_{i+1}$ be a mapping, which calculates an approximate solution $y_{i+1}$ at $t_{i+1} = t_i + \tau$ for a given approximation $y_i$ at $t_i$.

Then, the consistency error is of order $O(\tau^p)$, if there exists a constant $C > 0$ such that

$$
\left| \tau^{-1}(y^{ex}(t_{i+1}) - y_{i+1}) \right| \leq C \tau^p,
$$

where $y^{ex}$ is an exact solution of the ODE with initial condition $y^{ex}(t_i) = y_i$. 
Definition 9. Let \( y' = f(t, y) \) be a parabolic PDE on the domain \([t_0, \infty[\). Let \( y_i \to \Psi(y_i) = y_{i+1} \) be a mapping, which calculates an approximate solution \( y_{i+1} \) at \( t_{i+1} = t_i + \tau \) for a given approximation \( y_i \) at \( t_i \). Then, the consistency error is of order \( O(\tau^p) \), if there exists a constant \( C > 0 \) such that

\[
\left\| \tau^{-1}(y^{ex}(t_{i+1}) - y_{i+1}) \right\| \leq C \tau^p,
\]

where \( y^{ex} \) is an exact solution of the parabolic PDE with initial condition \( y^{ex}(t_i) = y_i \).
Let $\Omega \subset [a_x, b_x] \times [a_y, b_y] = Q$ be an open bounded domain.

Discretize $Q$ by a structured grid $Q_h$ of meshsize $h$.

Denote $\Omega_h := Q_h \cap \Omega$ the interior points.

The set of regular points is:

$$\Omega_h^r := \{ z \in \Omega_h \mid z + (h, 0), z + (-h, 0), z + (0, h), z + (0, -h) \in \Omega_h \}.$$

and the set of near boundary points: $\Omega_h^n := \Omega_h \setminus \Omega_h^r$.

Let the set of boundary points $\Gamma_h$ be the set

$$\{(x, y + \tau) \in \partial \Omega \mid (x, y) \in \Omega_h^n, (x, y + h) \not\in \Omega_h, (x, y), (x, y + \tau)[\subset \Omega]\} \cup \{(x, y - \tau) \in \partial \Omega \mid (x, y) \in \Omega_h^n, (x, y - h) \not\in \Omega_h, (x, y), (x, y - \tau)[\subset \Omega]\} \cup \{(x + \tau, y) \in \partial \Omega \mid \ldots\} \cup \{(x - \tau, y) \in \partial \Omega \mid \ldots\}.$$
Shortly-Weller Discretization

- For every point \( M = (x, y) \in \Omega_h \) denote the north point by

\[
N := \begin{cases} 
(x, y + \tau) & \text{if } (x, y + h) \not\in \Omega_h \\
(x, y + h) & \text{if } (x, y + h) \in \Omega_h.
\end{cases}
\]

Analogously, define the points \( N, S, W \).

- Let the mesh sizes \( h_N, h_S, h_W, h_E \) be defined such that

\[
\begin{align*}
N &= (x, y + h_N), & \text{where } M &= (x, y), \\
S &= (x, y - h_S), & \text{where } M &= (x, y), \\
E &= (x + h_E, y), & \text{where } M &= (x, y), \\
W &= (x - h_W, y), & \text{where } M &= (x, y).
\end{align*}
\]
Let us discretize the equation

\[-\Delta u = f, \quad u|_{\partial \Omega} = g\]

as follows

1. \(u(z) = g(z)\) for all \(z \in \Gamma_h\).
2. For every \(z \in \Omega_h^i\) let

\[-\Delta_h u_h(M) = \left( \frac{2}{h_N h_S} + \frac{2}{h_W h_E} \right) u(M) - \frac{2}{h_N (h_N + h_S)} u(N) - \frac{2}{h_S (h_N + h_S)} u(S) - \frac{2}{h_W (h_W + h_E)} u(W) - \frac{2}{h_E (h_W + h_E)} u(E).\]
Theorem 4. In general, the discretization matrix of the Shortly-Weller discretization is not symmetric.

The order of consistency is:

\[
\|(R_h(L(u)) - L_h(R_h(u)))(M)\| = O(h) \quad \forall M \in \Omega^i_h
\]

\[
\|(R_h(L(u)) - L_h(R_h(u)))(M)\| = O(h^2) \quad \forall M \in \Omega^i_h.
\]

If \( u \in C^4(\bar{\Omega}) \), then the convergence is of order \( O(h^2) \):

\[
\|R_h(u) - u_h\|_\infty = O(h^2).
\]
Direct Solvers for PDE’s

FD discretization of Poisson’s equation \( Mx = b \), where \( M \) is a matrix of size \( N = n^d \), \( d \) dimension.

<table>
<thead>
<tr>
<th>Method</th>
<th>Storage</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss elimination</td>
<td>( N^2 = n^{2d} )</td>
<td>( N^3 = n^{3d} )</td>
</tr>
<tr>
<td>Band Gauss elimination</td>
<td>( Nn^{d-1} = n^{2d-1} )</td>
<td>( Nn^{2(d-1)} = n^{3d-2} )</td>
</tr>
<tr>
<td>at ( d = 2 )</td>
<td>( n^3 )</td>
<td>( n^4 )</td>
</tr>
<tr>
<td>at ( d = 3 )</td>
<td>( n^5 )</td>
<td>( n^7 )</td>
</tr>
<tr>
<td>Nested dissection ( d &gt; 2 )</td>
<td>( n^{2d-2} )</td>
<td>( n^{3d-3} )</td>
</tr>
<tr>
<td>at ( d = 2 )</td>
<td>( n^2 \log n )</td>
<td>( n^3 )</td>
</tr>
<tr>
<td>at ( d = 3 )</td>
<td>( n^4 )</td>
<td>( n^6 )</td>
</tr>
<tr>
<td>Iterative multigrid</td>
<td>( n^d )</td>
<td>( n^d )</td>
</tr>
</tbody>
</table>
Let us write $Mx = b$ as

$$
\begin{pmatrix}
A & B \\
C & D
\end{pmatrix}
\begin{pmatrix}
t \\
x_{co}
\end{pmatrix}
=
\begin{pmatrix}
p \\
q
\end{pmatrix},
$$

where

$$
M = \begin{pmatrix}
A & B \\
C & D
\end{pmatrix},
\quad
b = \begin{pmatrix}
q \\
p
\end{pmatrix},
\quad
x = \begin{pmatrix}
t \\
x_{co}
\end{pmatrix}
$$

Here $co$ is an abbreviation for coarse.
Block Elimination

The block decomposition leads to

\[ M_{co} := D - CA^{-1}B \]
\[ b_{co} := p - CA^{-1}q. \]

One has to solve

\[ M_{co}x_{co} = b_{co} \quad (2) \]
\[ t = A^{-1}(q - Bx_{co}) \quad (3) \]

Here \( co \) is an abbreviation for coarse.
Equation (2) can be solved recursively or by Gauss-Elimination.
\( A^{-1} \) has to be calculated by Gauss-Elimination.
Block Elimination

Let

\[ \{1, 2, \ldots, N\} = A \cup B \]

Then

\[ \mathbb{R}^N = V_A \oplus V_B := \{ v + w \mid v \in V_A \text{ and } w \in V_B \} \]

where

\[ V_A = \left\{ \sum_{i \in A} e_i \lambda_i \mid \lambda_i \in \mathbb{R} \right\}, \]
\[ V_B = \left\{ \sum_{i \in B} e_i \lambda_i \mid \lambda_i \in \mathbb{R} \right\} . \]
Block Elimination

Spaces decomposition: \[ V_k = W_k \oplus V_{k-1}. \]
Then: \[ \mathbb{R}^N = W_{k_{\text{max}}} \oplus \ldots \oplus W_1 \oplus V_0 \]

\[
M_k = \begin{pmatrix}
A_k & B_k \\
C_k & D_k
\end{pmatrix}
\]

where

\[ A_k : W_k \to W_k, \quad B_k : V_{k-1} \to W_k, \]
\[ C_k : W_k \to V_{k-1}, \quad D_k : V_{k-1} \to V_{k-1}, \]
\[ M_k : V_k \to V_k \]
Nested Dissection

Decomposition of discretization grid:

\[ \Omega_h = \Omega^0 \cup \Omega^1 \cup \Omega^2 \cup \Omega^3 \]
Nested Dissection

Number the finest grid $\Omega_h = \{1, 2, \ldots, N\}$. Then, decompose:

$$\{1, 2, \ldots, N\} = \Omega^0 \cup \Omega^1 \cup \Omega^2 \cup \Omega^3$$

$$M_k = \begin{pmatrix} A_k & B_k \\ C_k & D_k \end{pmatrix}, \quad x_k = \begin{pmatrix} t_k \\ x_{k-1} \end{pmatrix}, \quad b_k = \begin{pmatrix} q_k \\ p_k \end{pmatrix}.$$  

$$M_{k-1} := D_k - C_k A_k^{-1} B_k$$

$$b_{k-1} := p_k - C_k A_k^{-1} q_k.$$  

$$M_{k-1} x_{k-1} = b_{k-1}$$

$$t_k = A_k^{-1} (q_k - B_k x_{k-1})$$

$M_0^{-1}$ and $A_k^{-1}$ have to be calculated by Gauss-Elimination.
Nested Dissection

The computational amount of nested dissection is dominated by computation of $M_0^{-1}$ and $A_k^{-1}$. Let us estimate this computational amount:

Let $n = 2^{k_{\text{max}}}$.

$$M_0$$

is matrix of size $O(2^{(d-1)k_{\text{max}}}) = O(n^{d-1})$.

$M_0^{-1}$ computation:

$O(n^{2d-2})$ storage requirement.

$O(n^{3d-3})$ computational requirement.
Nested Dissection

Let $n = 2^{k_{\text{max}}}$. $A_k$ has a block-structure and consists of $2^{d(k-1)}$ blocks of size $O(2^{(d-1)(k_{\text{max}}-k)})$.

Storage requirement for $A_k^{-1}$ computation:

$$O\left(\sum_{k=0}^{k_{\text{max}}} 2^{d(k-1)} (2^{(d-1)(k_{\text{max}}-k)})^2\right) = O(N \sum_{k=0}^{k_{\text{max}}} 2^{(d-2)k})$$

$$= O(N k_{\text{max}}) = O(n^2 \log(n)) \text{ if } d = 2$$

$$= O(N 2^{(d-2)k_{\text{max}}}) = O(n^{2d-2}) \text{ if } d > 2$$
Nested Dissection

Let $n = 2^{k_{max}}$.

$A_k$

has a block-structure and consists of

$$2^{d(k-1)}$$

blocks of size $O(2^{(d-1)(k_{max}-k)})$.

Computational requirement for $A_k^{-1}$ computation:

$$O\left(\sum_{k=0}^{k_{max}} 2^{d(k-1)}(2^{(d-1)(k_{max}-k)})^3\right) = O(N \sum_{k=0}^{k_{max}} 2^{(2d-3)k})$$

$$= O(n^{3d-3})$$
Implementation has to take into account that all matrices are block matrices.
→ recursive implementation is needed.
For reasons of simplicity assume \( d = 2, \Omega = [0, 1]^2 \).
Define the cells (Zelle)
\[
\mathcal{Z}_{i,j}^k = [ih_k, jh_k] \times [(i + 1)h_k, (j + 1)h_k],
\]
where \( h_k = 2^{-k} \) and
\[
I = (i, j) \in \mathcal{I}^k := \{(i, j) \mid i, j = 0, ..., 2^k - 1\}.
\]
Observe that \( \mathcal{Z}_{0,0}^0 = [0, 1]^2 \) and
\[
\mathcal{Z}_{i,j}^k = \mathcal{Z}_{i,j}^{k+1} \cup \mathcal{Z}_{i+1,j}^{k+1} \cup \mathcal{Z}_{i,j+1}^{k+1} \cup \mathcal{Z}_{i+1,j+1}^{k+1}.
\]
Define

\[ A_{i,j}^{k_{\text{max}}} := Z_{i,j}^{k_{\text{max}}} \cap \Omega_h \]

\[ B_{i,j}^k := A_{i,j}^k \cap \partial Z_{i,j}^k \]

\[ I_{i,j}^k := A_{i,j}^k \setminus B_{i,j}^k \]

\[ A_{i,j}^{k-1} := B_{i,j}^k \cup B_{i+1,j}^k \cup B_{i,j+1}^k \cup B_{i+1,j+1}^k \quad \text{for } k \leq k_{\text{max}}. \]

Furthermore, we can define

\[ \Omega_0 := B_{0,0}^0 \]

\[ \Omega_k := I_{i,j}^{k-1}. \]
Implementation of Nested Dissection

Let

\[ V(B) := \text{span}\{e_i \mid i \in B\} \]

Then define matrices, which map spaces to spaces:

\[ A^k_I : V(\mathcal{I}_I^k) \to V(\mathcal{I}_I^k), \quad B^k_I : V(\mathcal{B}_I^k) \to V(\mathcal{I}_I^k), \]
\[ C^k_I : V(\mathcal{I}_I^k) \to V(\mathcal{B}_I^k), \quad D^k_I : V(\mathcal{B}_I^k) \to V(\mathcal{B}_I^k), \]
\[ M^k_I : V(A^k_I) \to V(A^k_I) \]

These matrices are stored with respect to the standard basis \(\{e_i\}\). Extend matrix \(M : V(B) \to V(A)\) according

\[ M(e_i) := \begin{cases} M(e_i) & \text{if } e_i \in V(B) \\ 0 & \text{else.} \end{cases} \]
Implementation of Nested Dissection

\[
M_{i,j}^{k-1} := \sum_{I=i,j,...,i+1,j+1} D_{I}^{k} - C_{I}^{k} (A_{I}^{k})^{-1} B_{I}^{k}
\]

\[
b_{i,j}^{k-1} := \sum_{I=i,j,...,i+1,j+1} p_{I}^{k} - C_{I}^{K} (A_{I}^{k})^{-1} q_{I}^{k}.
\]

\[
t_{I}^{k} = (A_{I}^{k})^{-1} (q_{I}^{k} - B_{I}^{k} x_{I}^{k-1})
\]

On coarsest grid one has to solve exactly

\[
M^{0} x^{0} = b^{0}
\]

Equation

\[
M^{k-1} x^{k-1} = b^{k-1}
\]

has to be solved recurively from coarse to fine grid.
Implementation of Nested Dissection

How to define $M_{i,j}^{k_{\text{max}}}$

- In case of Finite Elements, these are the local stiffness matrices.
- In case of Poisson’s equation take the $4 \times 4$ matrix

$$
\frac{1}{h^2} \begin{pmatrix}
1 & -0.5 & 0 & -0.5 \\
-0.5 & 1 & -0.5 & 0 \\
0 & -0.5 & 1 & -0.5 \\
-0.5 & 0 & -0.5 & 1
\end{pmatrix}
$$
class VectorIndex : public ExprAlg<VectorIndex> {
    public:
    template <class Ind> VectorIndex(const Ind& index) {
        size = index.getSize(); Sn = index.getIndices();
        data = new double[size];
        s = new int; Smy = new int;
    }
    template <class A> void operator=(const ExprAlg<A>& a);

    private:
        double* data;
        int size; // Laenge Vektor
        int *Sn; // Nummern der globalen Indizes
        int *Smy; // fuer Auswertung: globaler Index
        int *s; // fuer Auswertung: lokaler Index
};
void VectorIndex::startI(int max_size) const {
    (*s) = 0;  // Initialize the index
    if(size>0) (*Smy) = Sn[(*s)];
}

double VectorIndex::getValueI(int Sglobal) const {
    while(Sglobal > (*Smy) && (*s) < size) {
        ++(*s);  // Increment the index
        (*Smy) = Sn[(*s)];  // Update the element
    }
    if((*Smy) > Sglobal || (*s)>=size) return 0;
    return data[(*s)];
}

template <class A>
void VectorIndex::operator=(const ExprAlg<A>& a) {
    const A& ao(a); ao.startI(size);
    for(int ss = 0;ss < size;++ss ) {
        data[ss] = ao.getValueI(Sn[ss]);
    }
} // ----> sorted Indizes!!!
Implementation of matrices with Indizes:

class MatrixIndex : public ExprAlg<...> { 
    public:
        template <class Ind>
        MatrixIndex(const Ind& indexI, const Ind& indexJ);
    ....
}

Operators like =,+,- are implemented such that they can be applied to vectors and matrices with respect to different index set:

\[ v = b + c \]

Here iteration is performed for the index set \( A \) of \( v \). If \( b \) or \( c \) is not defined at a certain index \( i \in A \), then `getValueI(i) return 0.0`. 

Observe that if $v$ is defined for a index set $A$. Then $v$ is contained in the corresponding vector space:

$$v \in V(A)$$

A class `IndexSet` is needed which

- stores indizes in a sequential order and
- allows union of two index set by `merge sort`. 
Implementation of Nested Dissection

The sets

\[ A_I^k = B_I^k \cup I_I^k \]

have to be represented by objects of class `IndexSet` and constructed recursively.

The matrices

\[
\begin{align*}
A_I^k &: V(I_I^k) \to V(I_I^k), \\
B_I^k &: V(B_I^k) \to V(I_I^k), \\
C_I^k &: V(I_I^k) \to V(B_I^k), \\
D_I^k &: V(B_I^k) \to V(B_I^k), \\
M_I^k &: V(A_I^k) \to V(A_I^k)
\end{align*}
\]

have to be represented by objects of class `MatrixIndex` and constructed recursively.
Implementation of Nested Dissection

The sets $A^k_I, B^k_I, \mathcal{I}^k_I$ and matrices $A^k_I, B^k_I, C^k_I, D^k_I$, and $M^k_I$ have to be stored as members of leaves in an quadtree.

class Leaf {
public:
    Leaf(...); ... VectorIndex* x;    ///> W
    ...
    MatrixIndex* A;    ///> W -> W
    MatrixIndex* B;    ///> Vb -> W
    ...
private:
    std::vector<Leaf*> children;
    IndexVector allIndizes;    ///> set A
    IndexVector interiorIndizes; ///> set I
    IndexVector boundaryIndizes; ///> set B
};
Implementation of Nested Dissection

- Nested Dissection has to be implemented by traversing through a quadtree with leaves of object `class Leaf`.

- Efficiency mainly depends on the efficient implementation of
  - matrix multiplication and
  - Gauss-algorithm implementation to compute $A^{-1}$. Using a linear algebra library on index sets. To this end cache efficient implementation is very important!