Übersicht:

- **Molecular Dynamics – Moleküldynamik/Partikelsimulation** (Diskretisierung der Zeitintegration, kurz- vs. langreichweitige Potentiale)
- **Parallelisierung**: OpenMP (gemeinsamer Speicher) und MPI (verteilter Speicher)
- **Lattice Boltzmann**
- **Finite Elemente**
- **Multigrid**

Moleküldynamik:

- Größenordnungen:
  - Quantenphysik: Elementarpartikel / Atome
  - Klassische Moleküldynamik: Moleküle
  - Mesoskopische Methoden: Gruppen aus Molekülen (Lattice Boltzmann)
  - Klassische Systeme, die das System als Einheit sehen und nicht mehr einzelne Moleküle (Beschreibung mit partiellen Differentialgleichungen, Finite Differenzen/Elemente)
  - Astrophysik (Kreisschluss mit Partikelmethoden auf Ebene von Atomen/Molekülen)
- Computersimulation von Partikelmodellen:
  - Entwicklung eines Systems von interagierenden Partikeln über die Zeit durch Integration der Bewegungsgleichungen
  - System mit N Partikeln mit: Masse m, Position x und Geschwindigkeit v
  - Modellierung der Partikelinteraktionen mit „guten" Algorithmen - Ziele:
    - Mit gegebener Anzahl an Operationen möglichst die beste Genauigkeit
    - Eine bestimmte Genauigkeit mit möglichst wenigen Operationen
    - D.h. gutes Kosten-Nutzen-Verhältnis
  - Zwei wichtige Punkte – Hauptbestandteile der Simulation:
    - Zeitintegration: numerische Methoden zum „schnellen" Lösen der Differentialgleichungen (Positions- und Geschwindigkeitsupdate)
    - „Schnelle" Auswertung der Kräfte: $O(N^2)$ >> $O(N\log^a N)$ = asymptotisch quasi-optimal [Barnes Hut, …] > $O(N)$ = asymptotisch optimal [linked cell, …]
  - Basis-Algorithmus:

```
t = start_t;
< initialisiere Orte x und Geschwindigkeiten v >
while(t < t_end) {
   < Berechne x und v zum Zeitpunkt t + delta_t durch ein
   Integrationsverfahren aus x, v und F >
   t += delta_t;
}
```

  - Welche Methode verwenden? (kurz- oder langreichweitige Potentiale, Komplexität, Speicherverbrauch, Parallelisierung, …)
- Randbedingungen: periodisch, reflektierend, ausströmend, einströmend
- Physikalische Grundlagen:
  - Potential (potentielle Energie): konservativ, d.h. Zeitunabhängig
  - Kinetische Energie
  - Hamiltonfunktion als Summe über die kinetische Energie und das Potential
  - Daraus hergeleitet: die Newtonschen Bewegungsgleichungen
    - $v = \dot{x}$
    - $F = m\dot{v} = m\ddot{x}$
  - Energie ist eine Erhaltungsgröße des Systems: $\frac{dE}{dt} = 0$
  - Finite Differenzen Approximationen:

- $\left[\frac{dx}{dt}\right]_n^r = \frac{x(t_{n+1})-x(t_n)}{\delta t} = \frac{dx}{dt} + O(\delta t)$
- $\left[\frac{dx}{dt}\right]_n^z = \frac{x(t_{n+1})-x(t_{n-1})}{2\delta t} = \frac{dx}{dt} + O(\delta t^2)$
- $\left[\frac{d^2x}{dt^2}\right]_n = \frac{x(t_{n-1})-2x(t_n)+x(t_{n+1})}{(\partial t)^2} = \frac{d^2x}{dt^2} + O(\delta t^2)$

- Standard Störmer-Verlet:
  - $F_n = m\frac{x_{n-1}-2x_n+x_{n+1}}{(\delta t)^2} \rightarrow x_{n+1} = 2x_n - x_{n-1} + \frac{F_n(\partial t)^2}{m}$
  - Initial bekannt sein müssen x_0 und x_1
  - Rundungsfehler, da ∂t sehr klein sein kann und damit auch $\frac{F_n(\partial t)^2}{m}$ - Viel kleiner als x_n und x_{n-1}
  - $v_n = \frac{x_{n+1}-x_{n-1}}{2\partial t}$
- Weniger anfällig für Rundungsfehler:
  - Leapfrog-Schema:
    - Geschwindigkeit wird zum Halbschritt berechnet (dafür allerdings explizit)
    - $v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + \frac{F_n \partial t}{m}$
    - $x_{n+1} = x_n + \partial t \cdot v_{n+\frac{1}{2}}$
    - Geschwindigkeiten und Positionen liegen so erst einmal zu verschiedenen Zeitpunkten vor
    - $v_n = \frac{1}{2}(v_{n+\frac{1}{2}} + v_{n-\frac{1}{2}})$
  - Geschwindigkeits-Störmer-Verlet:
    - $x_{n+1} = x_n + \partial t \cdot v_n + \frac{F_n(\partial t)^2}{2m}$
    - $v_{n+1} = v_n + \frac{(F_n+F_{n+1})\partial t}{2m}$
    - Position und Geschwindigkeit liegen zum gleichen Zeitpunkt vor
- Die drei Methoden haben gleichen Speicheraufwand (Geschwindigkeits-Störmer-Verlet braucht einen Hilfsvektor mehr)
- Alle drei Methoden sind von zweiter Ordnung, d.h. $O(\partial t^2)$
- Fehlerquellen: Rundungsfehler und Fehler durch Diskretisierung (=Approximation)
- Störmer-Verlet:
  - Zeitreversibel
  - Symplektisch: maßerhaltend im Phasenraum → ausgezeichnete Eigenschaften der Energieerhaltung → die numerische Approximation lässt sich als exakte Lösung von leicht gestörten Hamiltonsystemen betrachten
  - Einfach
  - Meist verwendete Verfahren zur Integration der Newtonschen Gleichungen

- Basis-Algorithmus mit Störmer-Verlet:

```
t = t_start;
< Initialisierung von x und v (und m) >
< Anlegen eines Hilfsvektors F_old >
< initiale Berechnung von F >
while(t < t_end) {
    < Schleife über alle Partikel>
        < Update von x gemäß Geschwindigkeits-Störmer-Verlet >
        F_old = F;
    < Berechnung von F >
    < Schleife über alle Partikel>
        < Update von v gemäß Geschwindigkeits-Störmer-Verlet >
    < Berechnung kinetischer Energie / Output / etc. >
    t += delta_t;
}
```

- Berechnung der Kräfte:
  - Beachte, dass gilt $F_{ij} = -F_{ji}$
  - $O(N^2)$ → bessere Methoden, um auf O(NlogN) oder gar O(N) zu kommen
- Abschneideradius – Cut-Off-Radius:
  - Langsam abnehmendes Potential: z.B. $V \sim \frac{1}{r} \quad \rightarrow \quad F \sim \frac{1}{r^2}$ (z.B. Gravitation und Coulomb)
  - Schnell abnehmendes Potential:
    - z.B. $V \sim \frac{1}{r^6} \quad \rightarrow \quad F \sim \frac{1}{r^7}$ (z.B Lennard-Jones mit $\frac{1}{r^6}$ und $\frac{1}{r^{12}}$)
    - Partikel, die „weit genug weg" sind, müssen nicht betrachtet werden:

$$U(r_{ij}) = \begin{cases} \dots & r_{ij} \leq r_{cut} \\ 0 & r_{ij} > r_{cut} \end{cases}$$

    - Dadurch wird die Gesamtenergie des Systems „leicht" verändert
- Linked-Cell-Algorithmus:
  - Teile das Gebiet in Zellen mit Seitenlänge ≥ r_{cut}
  - Interaktionen nur mit Partikeln in Nachbarzellen! → O(N)
  - Basis-Algorithmus:

```
Schleife über alle Zellen z
    Schleife über alle Nachbarn n
        Schleife über alle Partikel i in Zelle z
            Schleife über alle Partikel j in Nachbarzelle n
                if(r_ij <= r_cut) compute_F(i,j);
    Schleife über alle Partikel i in Zelle z
        Schleife über alle Partikel j in Zelle z
            if(i != j) compute_F(i,j);
```

  - Dynamik: Partikel sind Zellen nicht fest zugeordnet, sondern wechseln ihre Zugehörigkeit
  - Effiziente Implementierung:
    - Partikel in einem statischen Array (in den Zellen Zeiger auf dieses Array)
    - Zelle kennt alle Nachbarn und enthält eine schnelle (!) Liste aller Partikel der Zelle
    - Niemals über alle Zellen iterieren, sondern beim Zuordnen der Partikel zu den Zellen zu Begin jeder Iteration eine schnelle (!) Liste aller Zellen mit Partikeln erstellen → Iteration nur über diese Liste
    - Ausnutzen, dass gilt $F_{ij} = -F_{ji}$
  - Beispiele für das Lennard-Jones Potential:
    - Kollision zweier Körper
    - Fallender Tropfen
    - Zwei Gase mit unterschiedlicher Dichte
    - Rayleight-Taylor-Instabilität (zwei unterschiedlich schwere Flüssigkeiten übereinander)
    - Oberflächenwellen in granularen Medien
  - Parallelisierung:
    - Gemeinsamer (z.B. OpenMP) vs. verteilter Speicher (z.B. MPI)
    - Gemeinsamer Speicher: relativ simpel und straight-forward → SIMD
    - Verteilter Speicher:
      - Kommunikation (→ Netzwerk-Typ und Topologie!)
      - Naiv: Daten über alle Prozesse replizieren..
        - Ein Prozess arbeitet nur auf einem Teil der Daten
        - Nach jeder Iteration Abgleich unter allen Prozessen
        - Viel Kommunikation
        - Hoher Speicher-Aufwand der die Skalierung erheblich beschränkt
      - Datenpartitionierung:

- Prozess hält nur noch die Daten, auf denen er arbeitet (durch ein kurzreichweitiges Potential ist das auch nur ein Bruchteil aller Partikel)
- Ausgetauscht zwischen den Prozessen wird damit auch immer nur dieser Bruchteil

- Domain Decomposition – Gebietszerlegung:
  - Kommunikation an den Gebietsgrenzen („Ghost-Layer“)
  - Evt. ein Problem: inhomogene Verteilung der Partikel!

	Operationen	Kommunikation	Speicher
replizierte Daten	$O(N/P)$	$O(N)$	$O(N)$
Datenpartitionierung	$O(N/P)$	$O(N/P)$	$O(N/P)$
Gebietszerlegung	$O(N/P)$	2D: $O(\sqrt{N/P})$ 3D: $O((N/P)^{2/3})$	$O(N/P)$

- Langreichweitige Potentiale:
  - Kein „Cut-Off“, auch Wechselwirkungen mit den weit entfernten Partikeln müssen berücksichtigt werden
  - Naiv: $O(N^2)$
  - Gitterbasierende Methoden:
    - Darstellung des (elektrostatischen) Potentials $\Phi(\vec{x})$ als partielle Differentialgleichung [Poisson-Gleichung – elliptische partielle Differentialgleichung zweiter Ordnung]: $-\Delta\Phi(\vec{x}) = \frac{1}{\varepsilon_0}\rho(\vec{x})$ mit Ladungsdichte $\rho(\vec{x})$
    - Aus diesem Potential lassen sich Energie und Kräfte leicht bestimmen (Gradientenbildung)
    - Randbedingung:
      - Gebiet „sehr groß“ machen und dann Dirichlet-Randbedingungen
      - Periodische Randbedingungen
    - Partielle Differentialgleichung: Diskretisieren und Lösen (→ SiWiR I) [schnelles direktes Verfahren: FFT]
    - Glattheit der Lösung hängt dann stark von der Glattheit von $\rho(\vec{x})$ ab
    - Wenn $\rho(\vec{x})$ nicht glatt ist, sondern z.B. zusammengesetzt aus Delta-Distributionen:
      - Idee: Zerlegung in einen glatten langreichweitigen und einen singulären kurzreichweitigen Anteil $\rightarrow V = V_{long} + V_{short}$
      - Kurzreichweitigen Anteil dann lösen mit einer Linked-Cell-Methode
  - Baumtechniken:
    - Quad- bzw. Octree Aufteilung des Gebiets
    - Idee: Menge weit entfernter Partikel als ein einziges „Super-Partikel“ betrachten
    - → Barnes-Hut! [O(NlogN)]

Lattice Boltzmann:

- Zellularer Automat:
  - Zellzustand zum Zeitpunkt t+1 hängt vom Zustand der „Nachbarschaft“ und von sich selbst zum Zeitpunkt t ab → zeitabhängig [Überführungsfunktion / Update-Regeln]
  - Inhärent parallel
  - Reguläres Grid (Zellraum) mit Zellen
  - Zellzustand: ganzzahlig oder reell
  - Von-Neumann-Nachbarschaft ($\begin{bmatrix} . & o & . \\ o & o & o \\ . & o & . \end{bmatrix}$) und Moore-Nachbarschaft ($\begin{bmatrix} o & o & o \\ o & o & o \\ o & o & o \end{bmatrix}$)

- Zugriffsmuster (→ Stencil) ähnlich wie bei Lösern (Jacobi) von elliptischen PDEs
- Update-Regeln zeit- und ortsunabhängig
- Zum modellieren physikalischer Systeme (Modelle sind einfach zu erstellen)

- Lattice Gas Methode (LGCA):
  - Zellen sind leer oder enthalten Gaspartikel (diese haben diskrete Geschwindigkeiten und Positionen) → alles diskret!
  - Stream-Step: in jedem Zeitschritt wandern Partikel abhängig von ihrer Geschwindigkeit in Nachbarzellen
  - Mehrere Partikel in einer Zelle kollidieren → Kollision muss elementaren physikalischen Regeln folgen: Masse-, Impuls- und Energieerhaltung
  - In der Praxis: sehr feines Zellgitter nötig, sehr rechenintensiv
- Lattice Boltzmann Methode (LBM):
  - Repräsentation der Partikel als reelle Zahl → Dichte
  - Geschwindigkeiten, Positionen und Zeit diskret
  - Stream- und Collide-Step
  - Alternative zu klassischen Flüssigkeitssimulationen (PDE [Navier-Stokes Gleichungen] diskretisieren und lösen [sehr großes System])
  - Vorteile:
    - Einfacher Algorithmus, leicht zu parallelisieren
    - Einfach anzupassen an komplizierte Geometrie, free-surface-flow
  - Nachteile:
    - Einige Erweiterungen (adaptive Grids) fehlen/noch nicht entwickelt/kompliziert
    - Sehr rechenintensiv
  - Model:
    - $f_i(x,t) \geq 0$ reeller Wert, der die durchschnittliche Anzahl an Partikeln angibt, die von Zelle x zum Zeitpunkt t in Richtung c_i wandern (probability distribution function)
    - Dichte: $\varrho(x,t) = \sum_{i=0}^{N} f_i(x,t)$
    - Impuls-Dichte (momentum density): $\varrho(x,t)u(x,t) = \sum_{i=0}^{N} c_i f_i(x,t)$
    - Kollisionsmodell:
      - Partikel-Partikel wäre sehr kompliziert
      - → single time relaxation approximation
      - Equilibrium distribution: $f_i^{eq}(x,t) = f_i^{eq}(\varrho(x,t), u(x,t))$
      - Gleichung zum updaten der Partikel-Verteilungen – zeit-, ort- und geschwindigkeitsdiskrete Lattice-Boltzmann Gleichung:
        $$f_i(x + c_i\Delta t, t + \Delta t) = f_i(x,t) - \frac{1}{\tau}(f_i(x,t) - f_i^{eq}(x,t))$$
      - $\frac{1}{\tau}$ ist Relaxations-Konstante (steuert Einfluss vom Equilibrium)
      - Kollisions-Operator: $-\frac{1}{\tau}(f_i(x,t) - f_i^{eq}(x,t))$
      - Collide-Step: $\widetilde{f_i}(x, t+\Delta t) = f_i(x,t) - \frac{1}{\tau}(f_i(x,t) - f_i^{eq}(x,t))$
      - Stream-Step: $f_i(x + c_i\Delta t, t + \Delta t) = \widetilde{f_i}(x, t+\Delta t)$
    - Randbedingung:
      - bounce-back (no-slip): Partikelgeschwindigkeiten, die die Wand treffen, werden einfach umgedreht
      - free-slip
    - Modelle: D2Q9, D3Q15 (kein Übergang in der Mitte von Kanten), D3Q27
  - Algorithmus:

```
< initial f^eq(x,t) berechnen basierend auf gegebenem ϱ(x,t) und u(x,t) >
< f(x,t) initial auf f^eq(x,t) setzen >
```

```
for(unsigned int i = 0; i < max_time_steps; ++i) {
    collide_step();
    stream_step();
    < ϱ(x,t) und u(x,t) neu berechnen >
    < f^eq(x,t) neu berechnen >
}
```

- Performance-Optimierungen (vor allem in Hinblick auf „spatial locality“):
  - Collision-optimized Data-Layout: Daten einer Zelle hintereinander
  - Propagation-optimized Data-Layout: Die gleichen Richtungen aller Zellen blockweise hintereinander
  - Blocking (geometrisch und zeitlich)
  - Grid-Compression: nicht mehr mit zwei Grids arbeiten und diese die ganze Zeit „swappen“, sondern dafür nur noch ein Grids verwenden, dass eine Zelle in jeder Richtung größer ist (→ beim „streamen“ schieben)
  - Inhärent parallel → Parallelisierung!
- Einordnung:
  - Makroskopische Methoden: Direkte Löser der Navier-Stokes Gleichungen
  - Mesoskopische Methoden: Lattice Boltzmann
  - Mikroskopische Methoden: Simulation einzelner Fluidpartikel → Moleküldynamik

Finite Elemente:

- Ein ganz klarer Nachteile von finiten Differenzen: die Differenzengleichungen werden (sehr) kompliziert bei nicht-äquidistanten Stützstellen und nicht-einfachen Gebieten
- Die Methode der finiten Elemente ist in der Hinsicht flexibler
- Herleitung…:
  - … anhand von: $-\Delta u = f \ \ in\, \Omega, \ \ u = 0 \ auf \ \partial\Omega$
  - Multiplikation mit Testfunktion v ($v = 0 \ auf \ \partial\Omega$) und Integration:
    $$-\int_\Omega \ \Delta u \cdot v \ dx = \int_\Omega \ fv \ dx$$
  - Partielle Integration (Satz von Green):
    $$-\int_\Omega \ \Delta u \cdot v \ dx = \int_\Omega \ \nabla u \cdot \nabla v \ dx - \int_{\partial\Omega} v \nabla u \cdot n \ d\sigma = \int_\Omega \ \nabla u \cdot \nabla v \ dx$$
    $$(\Rightarrow \int_{\partial\Omega} v \nabla u \cdot n \ d\sigma = 0)$$
  - Schwache Formulierung (Lösung muss nur noch einmal und nicht mehr zweimal differenzierbar sein):
    $$\int_\Omega \ \nabla u \cdot \nabla v \ dx = \int_\Omega \ fv \ dx$$
  - Als Approximation wird eine Lösung von endlich-dimensionalen Problemen gesucht
  - Einführung einer endlich-dimensionalen Basis $span\{\phi_1, \dots, \phi_n\}$: $u_h = \sum_{i=1}^n u_i \phi_i$
  - Daraus folgt: $\sum_{i=1}^n u_i \int_\Omega \ \nabla\phi_i \cdot \nabla\phi_j \ dx = \int_\Omega \ f\phi_j \ dx \quad \forall\, j \in \{1,2,\dots,n\}$
  - Mit $A = \begin{bmatrix} \int_\Omega \ \nabla\phi_1 \cdot \nabla\phi_1 \ dx & \cdots & \int_\Omega \ \nabla\phi_n \cdot \nabla\phi_1 \ dx \\ \vdots & \ddots & \vdots \\ \int_\Omega \ \nabla\phi_1 \cdot \nabla\phi_n \ dx & \cdots & \int_\Omega \ \nabla\phi_n \cdot \nabla\phi_n \ dx \end{bmatrix}$, $\tilde{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$ und $\tilde{f} = \begin{bmatrix} \int_\Omega \ f\phi_1 \ dx \\ \vdots \\ \int_\Omega \ f\phi_n \ dx \end{bmatrix}$ folgt:

    $\Longrightarrow A\tilde{u} = \tilde{f}$ (A: globale Steifigkeitsmatrix [hier: symmetrisch und pos.-def.])
  - Basiselemente ϕ_i müssen so gewählt werden, dass A möglichst schwach besetzt ist (→ Basisfunktionen sollen einfach zu berechnen sein und möglichst kleine Träger haben)
  - Eine Möglichkeit: stückweise lineare ϕ_i (in 2D: führt zu „Hütchen“), d.h.
    - Gebiet im 2-Dimensionalen in Dreiecke (Lineares Lagrangesches Finites Element) zerlegen mit n inneren Punkten p_i

        - $\phi_j(p_i) = \begin{cases} 1 & , i = j \\ 0 & , i \neq j \end{cases}$, linear abfallend zu den Nachbarpunkten
        - Dann gilt: $\int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, dx \neq 0 \Leftrightarrow p_i$ und p_j sind benachbart
    - Viele andere Basisfunktionen denkbar (stückweise Polynome, …)
    - Andere Gebietsaufteilung möglich (Quadrate, …)
    - Erstellen der globalen Steifigkeitsmatrix:
        - Erstellen der lokalen Steifigkeitsmatrix eines jeden Elements: Auswerten aller relevanten $\int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, dx$ auf dem Gebiet dieses Elements (Dreiecke in Kombination mit „Hütchenfunktionen" [2D] → führt zu einer 3x3 Matrix)
        - Lokale Steifigkeitsmatrizen in die globale einsortieren: Aufaddieren der jeweils äquivalenten Einträge
    - Möglichkeiten die rechte Seite $\tilde{f}$ zu berechnen:
        - Wenn es möglich ist analytisch!
        - Durch numerische Integration der $\int_\Omega f\phi_j \, dx$, $j \in \{1,2,\dots,n\}$
        - $f(x)$ auch darstellen als $\sum_{i=1}^{n} f_i\phi_i$ (→ $Au = Mf$ [wenn $f = \lambda u$: Inverse Power Iteration: findet das λ, so dass $|\lambda|$ minimal ist mit $Au = \lambda Mu$])
    - → Lösen des Gleichungssystems $A\tilde{u} = \tilde{f}$ …
- Randbedingungen (homogen = 0):
    - Dirichlet: Wert ist vorgegeben
    - Neumann: Wert der Richtungsableitung der nach außen gerichteten Normalen ist gegeben
- Grundsätzliches Vorgehen der Finiten Elemente Methode:
    - Aufstellen der schwachen Formulierung der Differentialgleichung
    - Passende Finite Elemente wählen und Mesh erstellen (z.B. Triangulierung des Gebiets)
    - Damit verbunden: Wahl der (endlich vielen) Basisfunktionen (z.B. stückweise linear)
    - Lokale Steifigkeitsmatrizen erstellen (→ inhärent parallel)
    - Daraus die globale Steifigkeitsmatrix erzeugen (schwach besetzt [meist: sym. & pos.-def.])
    - Lösen des linearen Gleichungssystems
- Fehlerabschätzung und Konvergenz:
    - Konvergenz ist abhängig von den gewählten Finiten Elementen
    - Für lineare Finite Elemente gilt: $\|u - u_h\| \to 0 \quad f\ddot{u}r \quad h \to 0$
    - H^0-Norm: $\|f(x)\|_0 = \sqrt{\int_\Omega [f(x)]^2 dx}$
    - H^1-Norm: $\|f(x)\|_1 = \sqrt{\int_\Omega [f(x)]^2 + [f'(x)]^2 dx}$
    - Für lineare Finite Elemente gilt:
        - $\|u - u_h\|_0 \leq \|e_h\|_0 \leq Ch^2\|f\|_0 \rightarrow$ $O(h^2)$ in H^0-Norm
        - $\|u - u_h\|_1 \leq \|e_h\|_1 \leq Ch\|f\|_1 \rightarrow$ $O(h)$ in H^1-Norm
    - Konvergenz ändert sich bei Singularitäten!
- Methode der Finiten Elemente ist erweiterbar auf:
    - Allgemeinere Randbedingungen
    - Allgemeinere Finite Elemente (2D: Quadrate, …) [höhere Stetigkeit an den Knotenpunkten → Splines]
    - Allgemeinere Differentialgleichungen (4te Ordnung, …)
    - Höhere Dimensionen (3D: Tetraeder, Prismen)
    - Gekrümmte Ränder
    - Nicht-lineare partielle Differentialgleichungen
    - Kombination mit schnellen Lösern → Multigrid

Multigrid:

- Grundlegendes Problem der einfachen iterativen Methoden (Jacobi, Gauss-Seidel):
  - Gute Eliminierung der hochfrequenten/oszillierenden Fehlerkomponenten
  - Schlechte Eliminierung der niederfrequenten/glatten Fehlerkomponenten
- Je gröber das Grid, desto oszillierender „erscheinen“ glatte Fehlerkomponenten eines feinen Grids [Umgekehrt (aliasing) können oszillierende Fehler auf dem gröberen Grid glatter „erscheinen“]
- Erste Idee: auf einem sehr groben Gitter beginnen und damit eine Startlösung für ein feineres Gitter erzeugen → „nested iteration“ [Aber: was wenn bei einem feinen Grid angekommen noch immer glatte Komponenten im Fehler sind?]
- Es gilt für Fehler und Residuum: $e = x - \tilde{x} \quad \rightarrow \quad r = b - A\tilde{x} = Ax - A\tilde{x} = Ae$
- „two-grid correction scheme“:

< "Smoothing": ν_1-malige Relaxation von $A^h x^h = b^h$ auf Ω^h um $\tilde{x}^h$ zu erhalten >
< Berechne Residuum: $r^h = b^h - A^h \tilde{x}^h$ >
< Restriktion: $r^h \rightarrow r^{2h}$ >
< Löse $A^{2h} e^{2h} = r^{2h}$ auf Ω^{2h} >
< Interpolation/Prolongation: $e^{2h} \rightarrow e^h$ >
< Korrektur von $\tilde{x}^h$: $\tilde{x}^h = \tilde{x}^h + e^h$ >
< ν_2-malige Relaxation von $A^h x^h = b^h$ auf Ω^h mit initialem Startwert $\tilde{x}^h$ >

- Smoothing: Reduzierung hochfrequenter Fehlerkomponenten
- Restriktion:
  - Direkte Übernahme des entsprechenden Werts
  - Mittelung über alle Nachbarn (1D: $\frac{1}{4}[1 \quad 2 \quad 1]$ - 2D: $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$)
- Interpolation/Prolongation: lineare (1D) bzw. bilineare (2D) Interpolation
- Rekursion (V-Cycle): $A^{2h} e^{2h} = r^{2h}$ wieder nach gleichem Schema lösen! [Basisfall: das Problem ist grob genug und kann direkt gelöst werden!]
- Komplexität / Konvergenzordnung von FMG (Full-Multigrid V-Cycle [= nested iteration & V-Cycle]) Mehrgitterverfahren: O(N)