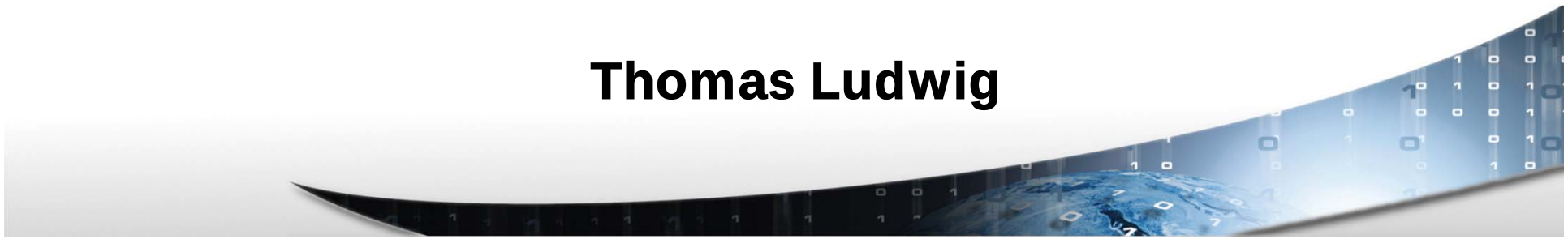
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DKRZ
DEUTSCHES
KLIMARECHENZENTRUM

Dialekte der Klimaforschung

Vom Fortran-Programm zum parallelen Programm

Thomas Ludwig



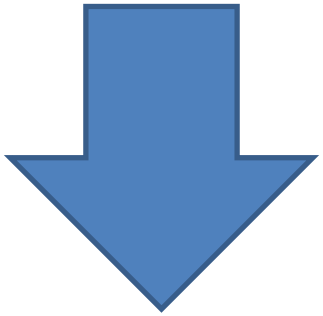
Inhalt

- Welche Dialekte werden transformiert?
- **Welche Anforderungen stellen wir?**
- Wozu diese Transformation?
- Wie ist der Zieldialekt bestimmt?
- Welche Varianten der Transformation gibt es?
- Wie sieht die Praxis aus?
- Was muss ich wissen?
- Wie gehe ich vor?
- **Was ist schwierig?**
- Was ist einfach?
- **Wie kontrolliere ich die Korrektheit?**
- Was bringt die Zukunft?

Welche Dialekte werden transformiert?

Fortran-Programm F-P (sequentiell)

Syntax A / Semantik A



Transformation

Paralleles Programm P-P (Fortran + Erweiterungen + Anpassungen für spezielle Hardware)

Syntax B / Semantik ?

Welche Anforderungen stellen wir?

Die Transformation sollte korrekt sein

d.h. die Semantik sollte erhalten bleiben

- Das parallele Programm sollte dieselbe Berechnung durchführen wie das sequentielle Programm
- Ein möglicher Test: beide Programme liefern bei identischer Eingabe eine identische Ausgabe

Eine Transformation sollte auf verschiedenen Maschinen effizient laufen

In der Praxis ist das typischerweise nicht gegeben !

Wozu diese Transformation?

- Feststellung: ein einzelner Rechner ist bei Klimasimulationen zu schwach, um diese mit der gewünschten räumlichen und zeitlichen Auflösung durchzuführen
  - Bei anderen wissenschaftlichen Fragestellungen gilt das nicht notwendigerweise !
- Deshalb: Berechnung durch Hochleistungsrechner nötig
- Diese sind heute immer mit dem Konzept des parallelen Rechnens realisiert

Die Transformation wandelt ein mathematiknahes Programm in ein rechnernahes Programm

Wie ist der Zieldialekt bestimmt?

Der Zieldialekt bestimmt sich durch die Art der verwendeten Hardware

a) Rechner mit gemeinsamem Hauptspeicher

   Alle Prozessorkerne arbeiten auf einem gemeinsamen Hauptspeicher, in dem sich das Programm und die Daten befinden

b) Rechner mit Beschleunigerhardware

   Zusätzlich kann hier spezielle Hardware (z.B. spezielle Grafikkarten) verbaut sein, die besonders zu programmieren ist

c) Rechner mit verteiltem Hauptspeicher

   Man schaltet mehrere Rechner zusammen und vernetzt sie. In jedem läuft ein Programm auf lokalen Daten

Wie ist der Zieldialekt bestimmt?

In der Praxis finden wir folgendes:

- Der „Rechnerknoten“ (ein Einschub in unseren Rechnerschränken) ist vom Typ a)
- Wir möchten keine Rechner vom Typ b)
- Mehrere „Rechnerknoten“ unseres System konstituieren ein System vom Typ c) (das wahlweise auch als Mischform von a) und c) betrachtet werden kann)

Die Programmierkonzepte richten sich nach dem Typ !

Welche Varianten der Transformation gibt es?

Automatische Übersetzung durch Compiler

- Bringt sehr schlechte Leistung, da der Compiler die Semantik des Programms nicht genau erkennt (vgl. automatisches Übersetzen zwischen Sprachen)

Manuelle Übersetzung durch Programmierer

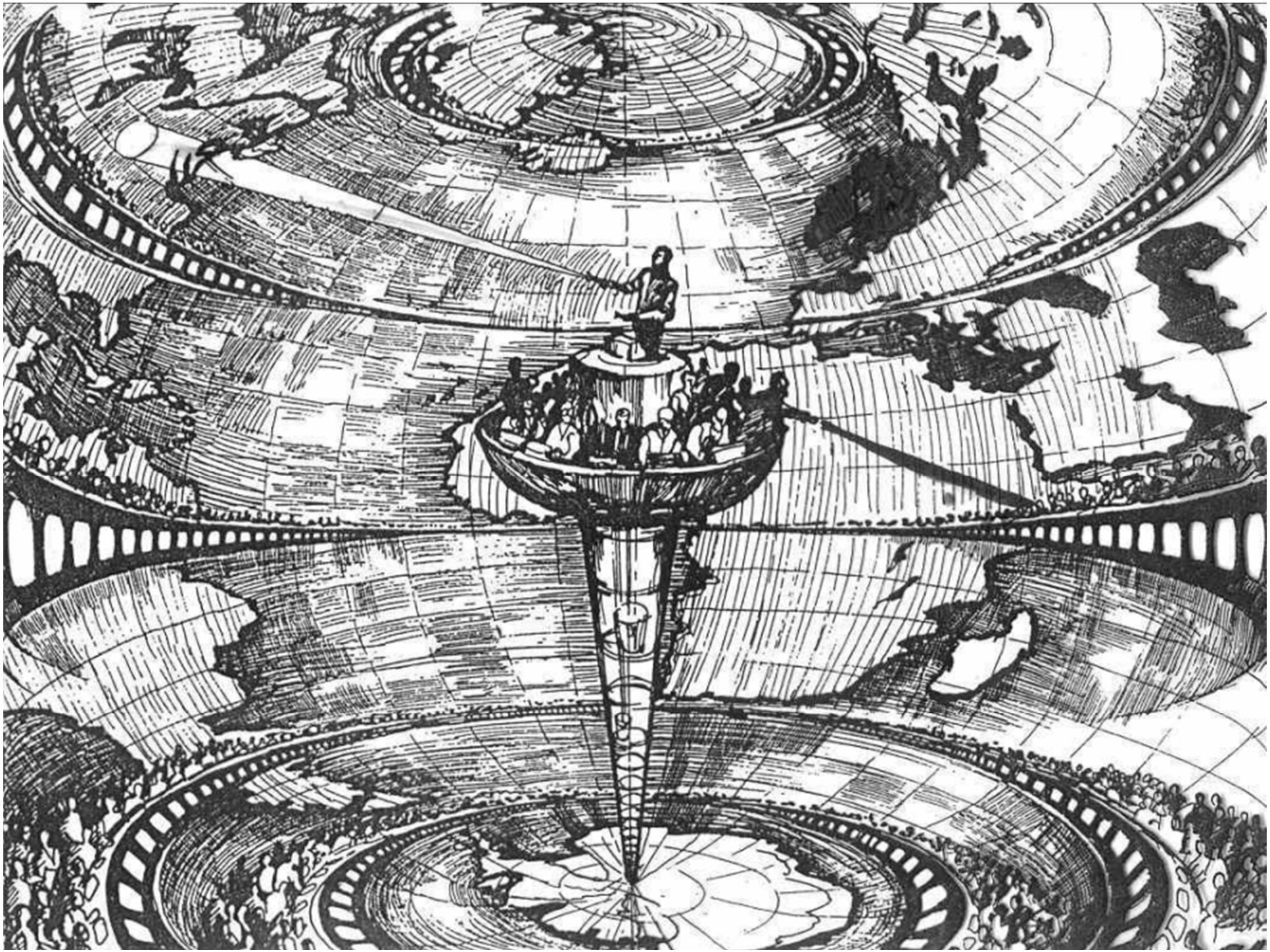
- Zeitaufwendig
- Intellektuell herausfordernd
- Fehleranfällig
- Schlecht, wenn er die Semantik des F-P nicht kennt und nicht erkennen kann
- Schlecht, wenn er die Besonderheiten des Rechners nicht kennt und nicht kennenlernen kann

Wie sieht die Praxis aus?

Meist **Datenparallelismus**

- Folgt dem gesunden Menschenverstand der Aufteilung von Arbeit: jeder erledigt einen Teil des Gesamten
- Im Rechner: Auf allen Prozessorkernen läuft gleichzeitig dasselbe Programm ab; die Daten werden aufgeteilt und den Prozessorkernen zugeteilt; die Teilergebnisse werden eingesammelt und zusammengeführt

Siehe Richardson´s Forecast Factory ☺



Was muss ich wissen?

- **Programmierkonzepte** (nach Wichtigkeit)
  - Für Rechner vom Typ c) : Programmierung mit Nachrichtenaustausch
  - Für Rechner vom Typ a) : Programmierung mit gemeinsamem Speicher
  - Für Rechner vom Typ b) : Spezialkonstrukte für die Spezialhardware
- **Parallele Umsetzung wichtiger mathematischer Konstrukte**
  - Matrizenmanipulationen
  - Numerische Lösungsverfahren
  - ...

Wie gehe ich vor?

Typischerweise

1. Suche Programmteile mit dem größten Rechenbedarf
2. Analysiere die Datenstrukturen
3. Programmiere eine Verteilung der Daten
4. Programmiere ein Einsammeln der Daten

Nebenbemerkung: die **Länge** des Programmcodes wächst dabei nur um ein paar Prozent, die **Komplexität** wächst um viele Größenordnungen! Grund ist das zeitliche Nebeneinander der Abarbeitungsstränge

Was ist schwierig?

Neuer Effekt bei parallelen Programmen:

Zeitabhängiges Verhalten / Nichtdeterminismus

Woher kommt das?

- Die Programmteile arbeiten jetzt ja gleichzeitig an verschiedenen Orten und müssen sich gelegentlich synchronisieren

Was bewirkt das?

- Mal ist der eine etwas eher fertig, mal der andere (wie im richtigen Leben)

Was ist schwierig? Zeitabhängigkeit

Was ist davon die Folge?

- Wenn es fehlerhafterweise so programmiert wurde:
  - Extrem schwer auffindbare Programmfehler (Heisenbug)
  - Einer von 5, 10, 100, 1000 Programmläufen stürzt ab

  (bei sequentiellen Programmen ist bei gleicher Eingabe das Resultat immer identisch)
- Wenn es absichtlich so programmiert wurde:
  - Die Programmergebnisse sind immer „richtig“ aber immer etwas unterschiedlich (wegen `a*(b*c) != (a*b)*c`)

  Warum wurde es denn dann so programmiert?
  - Weil man noch mehr Leistung herausquetschen kann

Was ist schwierig? Zeitabhängigkeit

Ganz problematisch:

- Häufig wissen wir nicht, ob wir Nichtdeterminismus versehentlich hineinprogrammiert haben
  - Z.B. können Bibliotheksfunktionen sich in manchen Fällen nichtdeterministisch verhalten

Die Folge:

- Fehlersuche und Verifikation der Korrektheit des Programms sind in solchen Fällen beliebig schwierig

Was ist schwierig? Die sequentiellen Algorithmen

Problem

- Die im F-P gewählten Implementierungen mathematischer Verfahren lassen sich nicht gut parallelisieren
- Besser parallelisierbar wäre eine andere Implementierung

Folge

- Man ändert die Implementierung der Mathematik im P-P
- Die Ergebnisse sind unvergleichbar mit denen des F-P
- Wiederum kann man nur schwer die korrekte Transformation sicherstellen

Was ist schwierig? Qualität der Parallelisierung

Lastungleichheit

- Die Prozessorkerne sind ungleich belastet und man muß dynamisch Lasten zwischen ihnen verschieben
- Gefahr eines geänderten Berechnungsergebnisses

Skalierbarkeit

- Das Programm liefert bei n Prozessorkernen annähernd n-fache Leistung, bei 2n Prozessorkernen aber deutlich weniger als 2n-fache Leistung
- Normalerweise muss die Implementierung der Mathematik wieder geändert werden

Was ist einfach? (1/2)

„Massiver Parallelismus“

Beispiel

- Ich habe ein paralleles Programm, das auf n Prozessorkernen prima läuft
- Ich bekomme einen neuen Rechner mit 100*n Prozessorkernen
- Ich kann das alte Programm in 100 Varianten gleichzeitig laufen lassen (z.B. bei Ensemblerechnungen)

Was ist einfach (2/2)?

Sonst nichts.

Wie kontrolliere ich die Korrektheit? (1/2)

Meist liefert das P-P wegen der Änderungen in der Mathematik bereits minimal andere Ergebnisse als das F-P

- Die Korrektheit der Transformation vom sequentiellen zum parallelen Programm ist damit meist prinzipiell schwer zu zeigen und darüber hinaus schwer zu kontrollieren

Was man zumindest fordert:

- Bei steigender Maschinengröße sollte das Ergebnis immer dasselbe bleiben
  - Das ist nicht immer garantierbar

Wie kontrolliere ich die Korrektheit? (2/2)

Was man sich wünscht

- Beim Wechsel wichtiger Softwarekomponenten des Systems bleiben die Ergebnisse identisch
  - Immer wieder mal nicht
- Beim Wechsel von einem Parallelrechner auf einen anderen bleiben die Ergebnisse identisch
  - Meist nicht

Was bringt die Zukunft?

Entwicklung bei Rechnern

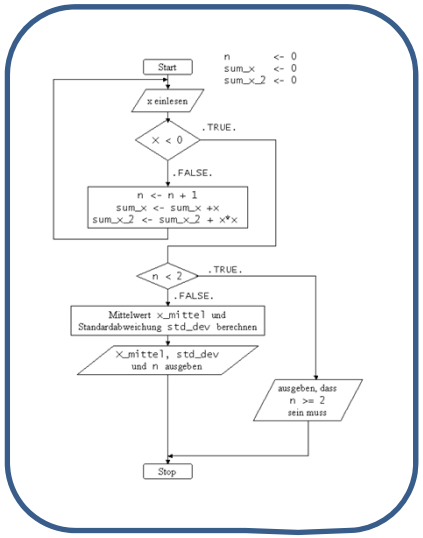
- Die Komplexitäten in allen Aspekten erhöhen sich weiter
- Die Maschinengrößen erhöhen sich weiter

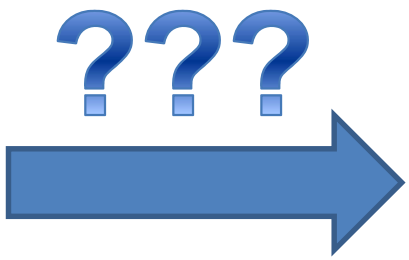
Entwicklung bei den Programmiersprachen

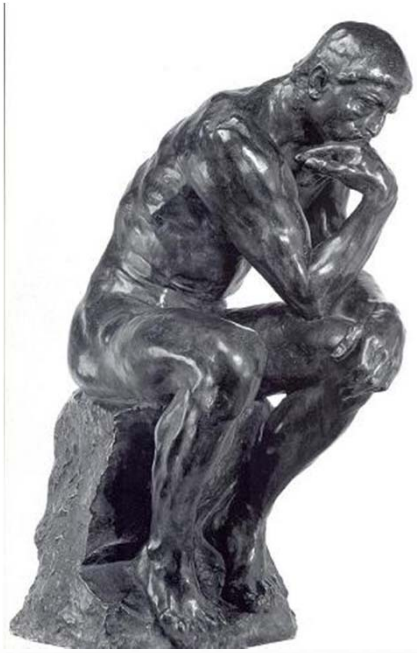
- Entwicklung neuer Konzepte
  - Aber wer soll die alten Programme umschreiben?

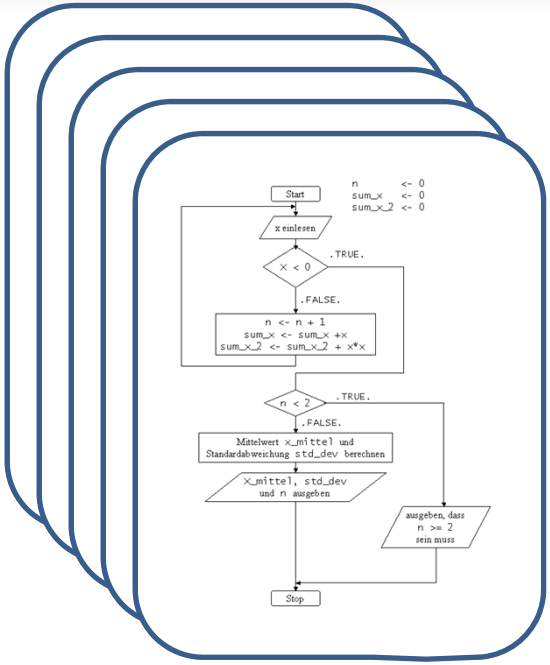
Entwicklung automatischer Verfahren

- Schwieriger denn je wegen der Rechnerentwicklung


Start
n <- 0
sum_x <- 0
sum_x_2 <- 0
x einlesen
X < 0
.TRUE.
.FALSE.
n <- n + 1
sum_x <- sum_x +x
sum_x_2 <- sum_x_2 + x*x
n < 2
.TRUE.
.FALSE.
Mittelwert x_mittel und
Standardabweichung std_dev berechnen
X_mittel, std_dev
und n ausgeben
ausgeben, dass
n >= 2
sein muss
Stop


???




Start
n <- 0
sum_x <- 0
sum_x_2 <- 0
x einlesen
X < 0
.TRUE.
.FALSE.
n <- n + 1
sum_x <- sum_x +x
sum_x_2 <- sum_x_2 + x*x
n < 2
.TRUE.
.FALSE.
Mittelwert x_mittel und
Standardabweichung std_dev berechnen
X_mittel, std_dev
und n ausgeben
ausgeben, dass
n >= 2
sein muss
Stop