Hat eine Gleichung eine Lösung?
Ein Computerbeweis

Peer Ueberholz

IMH - Institut für Modellbildung
und Hochleistungsrechnen

FB03

15.1.2009

zusammen mit

- Bruno Lang und Paul Willems
  Angewandte Informatik, FB Mathematik,
  Bergische Universität Wuppertal
- Mark Bull
  Edinburgh Parallel Computing Centre
  University of Edinburgh

1. Einführung

1.1 Computerbeweise

Als Computerbeweis bezeichnet man den Beweis einer Behauptung, das heißt einer mathematischen oder logischen Aussage, mit Hilfe eines Computerprogramms.

Frage: Lässt sich eine Aussage mit einen Computer beweisen?

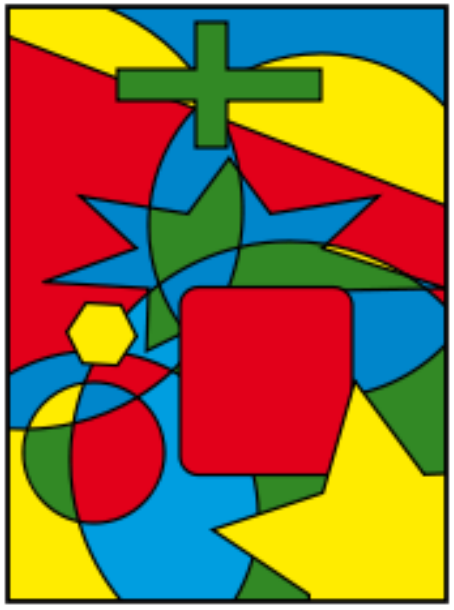
Antwort: Im Prinzip ja.

Vorgehen: Reduziere Problem auf eine endliche Menge von Fällen, die mit einen Computerprogramm überprüft werden können.

Der erste berühmte Computerbeweis: Das 4-Farben Problem

1852: Francis Guthrie

Kann eine Landkarte mit 4 Farben gefärbt werden, ohne dass zwei angrenzenden Länder die gleiche Farbe bekommen?



Heinrich Heesch:
Verfahren, um den Beweis mit Hilfe des Computers zu suchen.

Kenneth Appel und Wolfgang Haken (1977):
Untersuchten 1.936 problematische Fälle, die mit einen Computer einzeln geprüft wurden.

Benjamin Werner und Georges Gonthier (2004):
Formaler Beweis mit dem "Beweisassistenten" Coq konstruiert.

http://de.wikipedia.org/wiki/Vier-Farben-Satz
http://www.matheprisma.uni-wuppertal.de/MathePrisma/

Kann man einem Computerprogramm vertrauen?

Im Durchschnitt

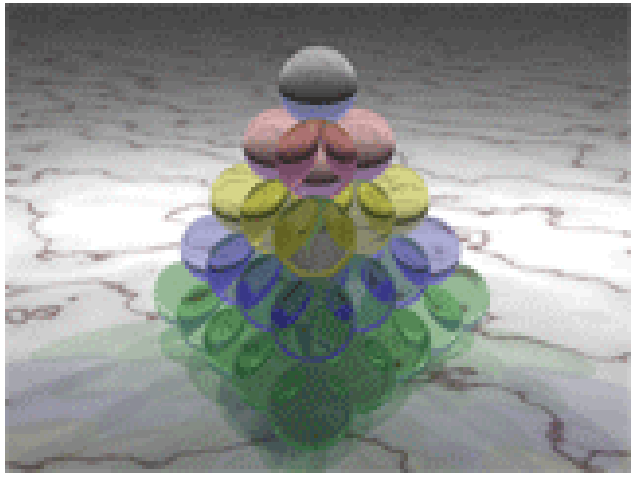
- macht jeder Programmierer pro Zeile 1,5 Fehler,
- enthält jedes Computerprogramm, welches auf dem Markt kommt, einen Fehler pro hundert Zeilen.

$\Rightarrow$ Software wird nie fehlerfrei sein.

Aber: Alfred Kempe fand 1879 einen scheinbaren Beweis für den 4-Farben Satz. Elf Jahre später, 1890, zeigte Percy Heawood, dass Kempes Beweisversuch fehlerhaft war.

Der zweite berühmte fast-Computerbeweis: Die Kepler-Vermutung

Frage: Wie packe ich Apfelsinen am Besten übereinander bzw. in eine Kiste?



Kepler: 1611
Eine Anordnung von gleich großen Kugeln hat die größte Dichte als kubisch-flächenzentrierte Packung und als hexagonale Packung.

Thomas Hales (1998):

- Es sind über 5.000 Kugelanordnungen zu untersuchen
- Für jede Kugelanordnungen minimiere eine Funktion mit 150 Variablen.

Die Gutachter des Beweises sind zu **99 Prozent** von der Richtigkeit des Beweises überzeugt.

$\Rightarrow$ Die keplersche Vermutung ist damit auf dem Wege, ein mathematischer Satz zu werden.

http://de.wikipedia.org/wiki/Keplersche_Vermutung

1.2 Nicht-lineare Gleichungssysteme

- Nicht-lineare Gleichungssysteme $F(z) = 0$ haben vielfältige Anwendungen, z.B. bei der Analyse von chemischen Prozessen.
- Aus Sicherheitsgründen müssen diese Gleichungen überprüfbar gelöst werden.,
- Es muss bewiesen werden, dass das Resultat stimmt.

Problem:

- Parameter der Gleichungen sind nur in bestimmten Bereichen bekannt.
- Rechengenauigkeit

$\Rightarrow$ **Intervall-Arithmetik**

Lösungsidee für nicht-lineare Gleichungen

Problem: Löse ein Gleichungssysstem $F(z) = 0$ bzw.

$$\begin{aligned} F_1(z_1, z_2, \ldots, z_n) &= 0 \\ F_2(z_1, z_2, \ldots, z_n) &= 0 \\ &\ldots \\ F_k(z_1, z_2, \ldots, z_n) &= 0 \end{aligned}$$

im Intervall

$$[z] = [\underline{z}_1, \overline{z}_1] \times \ldots [\underline{z}_n, \overline{z}_n].$$

Lösungsidee (2)

- Bestimme eine obere und untere Schranke für jede Gleichung F_i im Intervall $[z] \Rightarrow [F_i]$.
- Ist in jedem Intervall $[F_i]$ die 0 enthalten, könnte das Systeme eine Lösung haben.
  - Ist das Intervall bereits klein, versuche die Lösung zu verifizieren.
  - Andernfalls teile das Intervall und beginne von vorne für jedes Teilintervall.
- Ist in $[F_i]$ die 0 nicht enthalten, gibt es in dem Intervall $[z]$ keine Lösung.

Lösungsidee (3)

function Check ([z]) //teste, ob in einer Box [z] eine Lösung sein könnte
    if $0 \in [F_i]$ for each $i = 1, \ldots, k$
        //$[F_i]$ Umschließung aller Werte von F_i über [z]
    then
        if [z] ist kleiner als eine Schranke
            füge [z] in eine Liste möglicher Lösungen
        else
            unterteile [z] in Teilboxen $[z_1], [z_2]$
            call Check($[z_1]$)
            call Check($[z_2]$)

Problem dieser Methode

- Annahme: 10 Variablen, jede im Intervall [0,1024]
- Falls keine Intervalle weggeworfen werden können:
  - 2^{10} Intervall, wenn jede Variable halbiert ist.
  - $2^{100} \approx 10^{30}$ Intervalle, um die Breite jedes Intervalls auf 1 zu reduzieren.
- Alle Intervalle müssen gespeichert und untersucht werden
  $\Rightarrow$ Intelligente Algorithmen
  - bei der Untersuchung einzelner Intervalle
  - bei der Parallelisierung des Problems

  $\Rightarrow$ Parallelrechner
- Problem aller Branch-and-Bound Probleme wie z.B. von Optimierungsproblemen

2. SONIC

- **S**olver and **O**ptimizer for **N**onlinear Problems based on **I**nterval **C**omputation
- geschrieben in C++ (35.000 Zeilen)
- "allgemeiner" Intervall-Code, ausgelegt für hohe **Performance und Portabilität**.
- enthält einen Löser für nicht-lineare Gleichungen und Optimierungsprobleme

Die Methode "Constraint propagation"

I. Splitting Phase

Beispiel: System aufbrechen

$$\begin{cases} f_1 & := & x_1^2 - x_2 & = & 0 \\ f_2 & := & x_1^2 + x_2^2 - 1 & = & 0 \end{cases}$$

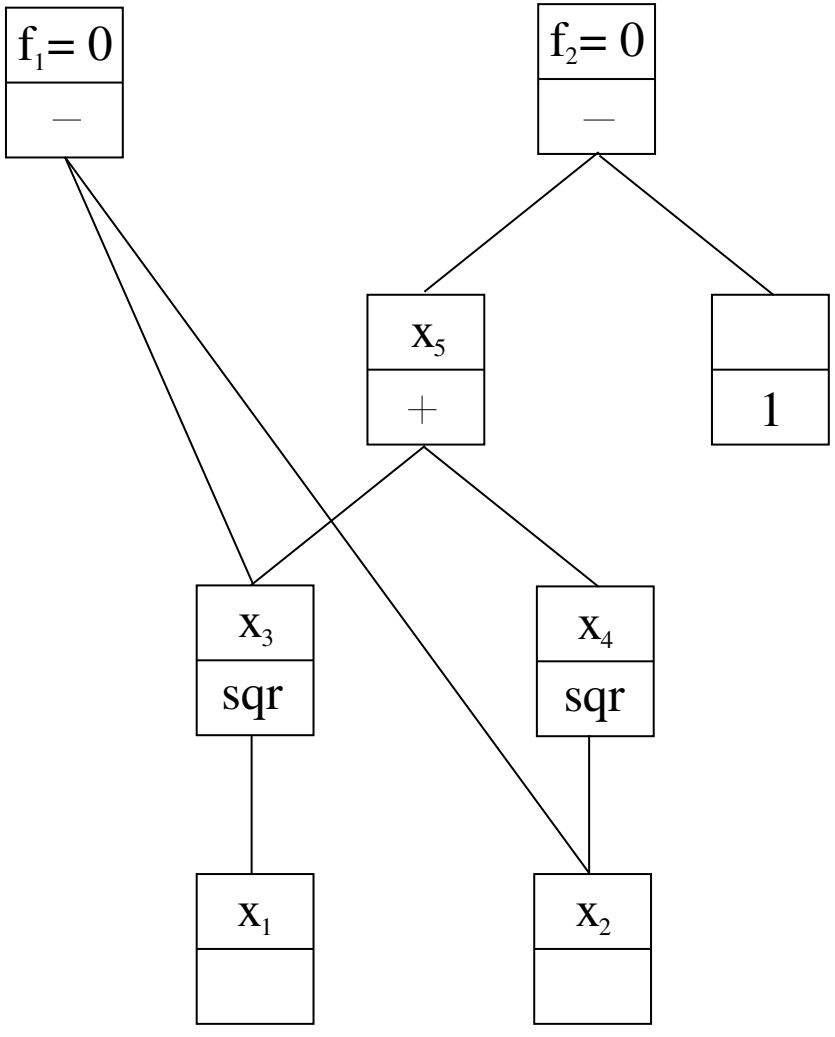
durch Hinzufügen neuer Variablen

$$(f_1)\begin{cases} x_3 & = & x_1^2 \\ 0 & = & x_3 - x_2 \end{cases} \qquad (f_2)\begin{cases} x_4 & = & x_1^2 \\ x_5 & = & x_2^2 \\ x_6 & = & x_4 + x_5 \\ 0 & = & x_6 - 1 \end{cases}$$

Gemeinsame Terme werden verknüpft und die Gleichungen zusammengefasst.

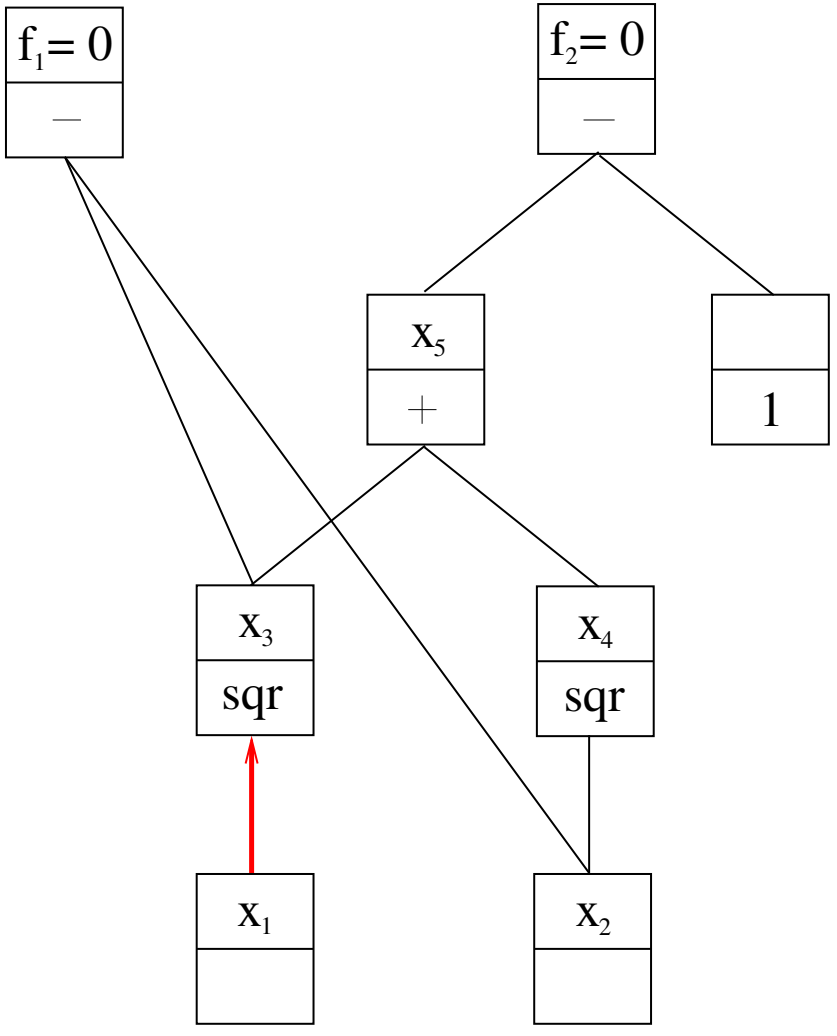
$$\begin{cases} 0 = x_3 - x_2 \\ 0 = x_5 - 1 \\ x_3 = x_1^2 \\ x_4 = x_2^2 \\ x_5 = x_3 + x_4 \end{cases}$$

II. "Forward Propagation"



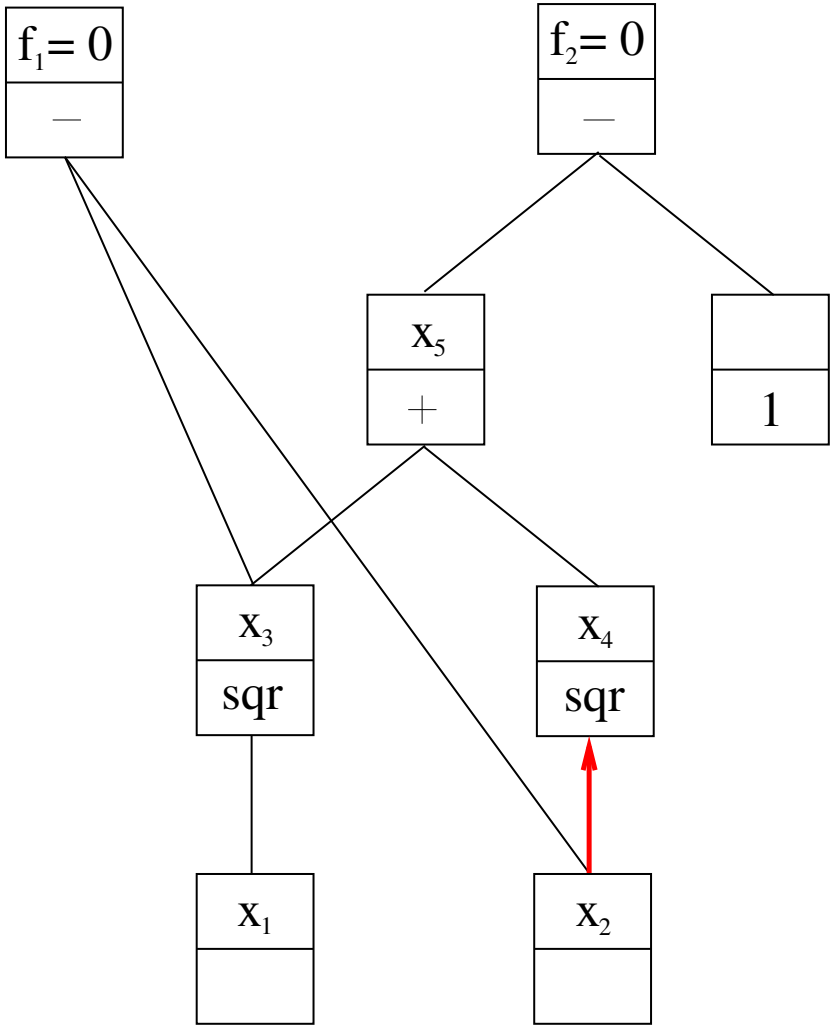
$$
\begin{aligned}
x_1 &= [-2, 2] \\
x_2 &= [-2, 2]
\end{aligned}
$$

$$
\begin{aligned}
0 &= x_5 - 1 \\
0 &= x_3 - x_2 \\
x_3 &= x_1^2 = (-\infty, \infty) \\
x_4 &= x_2^2 = (-\infty, \infty) \\
x_5 &= x_3 + x_4 = (-\infty, \infty)
\end{aligned}
$$



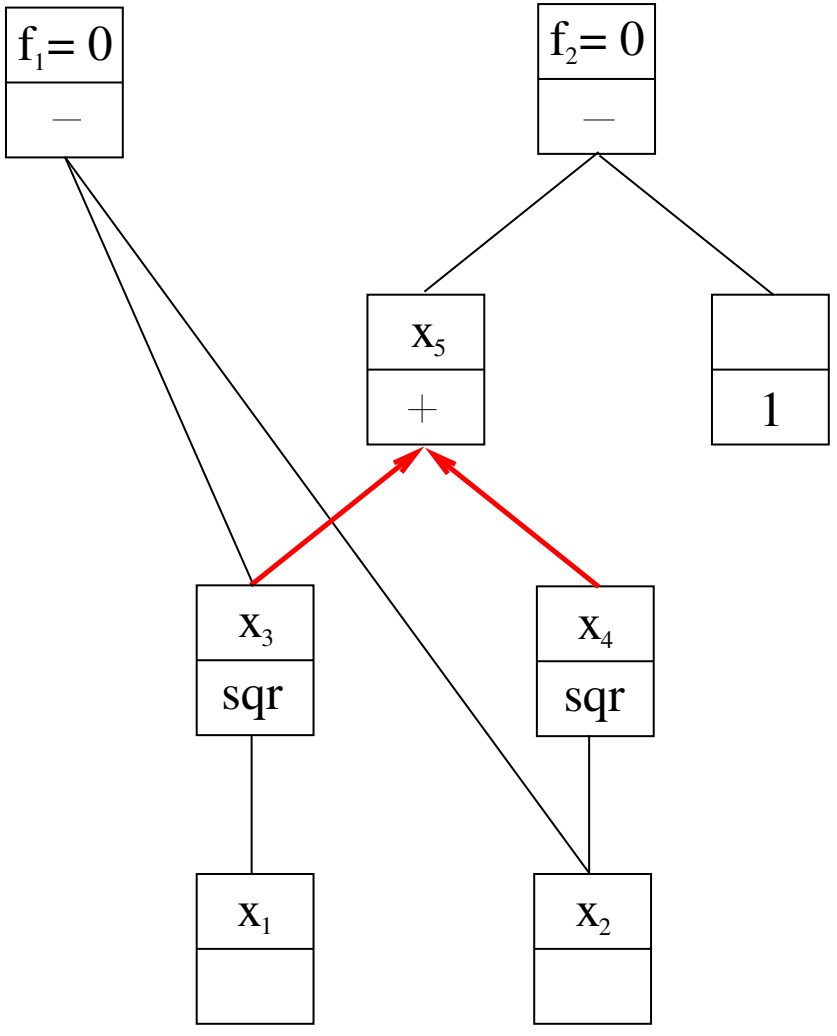
$$
\begin{aligned}
x_1 &= [-2, 2] \\
x_2 &= [-2, 2]
\end{aligned}
$$

$$
\begin{aligned}
0 &= x_5 - 1 \\
0 &= x_3 - x_2 \\
x_3 &= x_1^2 = [0, 4] \\
x_4 &= x_2^2 = (-\infty, \infty) \\
x_5 &= x_3 + x_4 = (-\infty, \infty)
\end{aligned}
$$



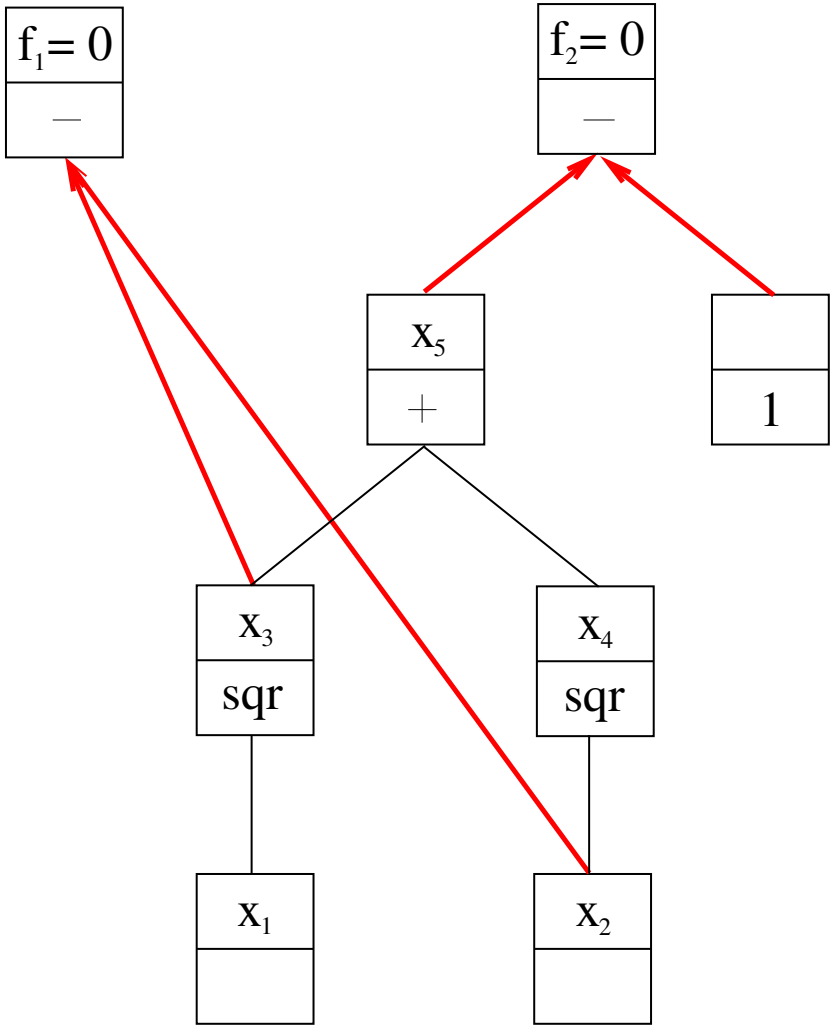
$$
\begin{aligned}
x_1 &= [-2, 2] \\
x_2 &= [-2, 2]
\end{aligned}
$$

$$
\begin{aligned}
0 &= x_5 - 1 \\
0 &= x_3 - x_2 \\
x_3 &= x_1^2 = [0, 4] \\
x_4 &= x_2^2 = [0, 4] \\
x_5 &= x_3 + x_4 = (-\infty, \infty)
\end{aligned}
$$



$$
\begin{aligned}
x_1 &= [-2, 2] \\
x_2 &= [-2, 2]
\end{aligned}
$$

$$
\begin{aligned}
0 &= x_5 - 1 \\
0 &= x_3 - x_2 \\
x_3 &= x_1^2 = [0, 4] \\
x_4 &= x_2^2 = [0, 4] \\
x_5 &= x_3 + x_4 = [0, 8]
\end{aligned}
$$



$$
\begin{array}{rcll}
 & & x_1 & = [-2, 2] \\
 & & x_2 & = [-2, 2] \\
 & & & \\
0 & = & x_5 - 1 & = [-1, 7] \\
0 & = & x_3 - x_2 & = [-2, 6] \\
x_3 & = & x_1^2 & = [0, 4] \\
x_4 & = & x_2^2 & = [0, 4] \\
x_5 & = & x_3 + x_4 & = [0, 8]
\end{array}
$$

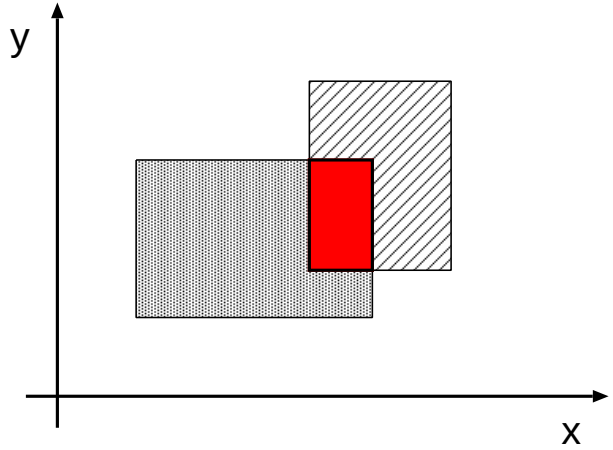
III. "Backward Propagation"

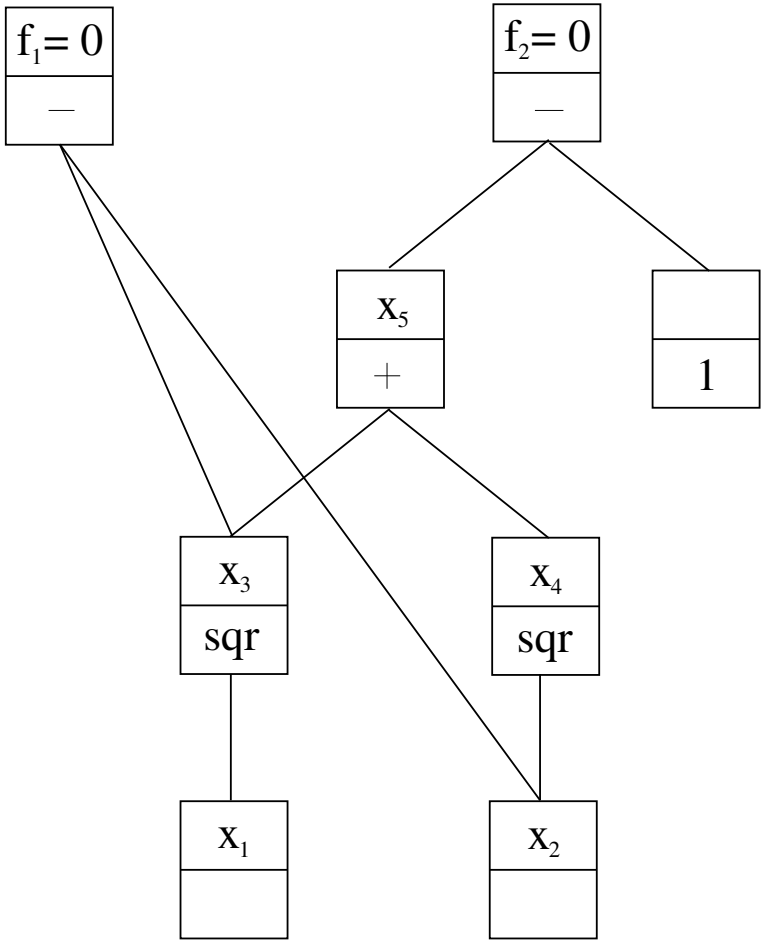
Idee: Propagiere Werte zurück von oben nach unten
Beispiel: Ein Knoten z hat den Term

$$z = x - y.$$

Löse mit vorgegebenem z folgende Gleichungen für die Variablen x, y:

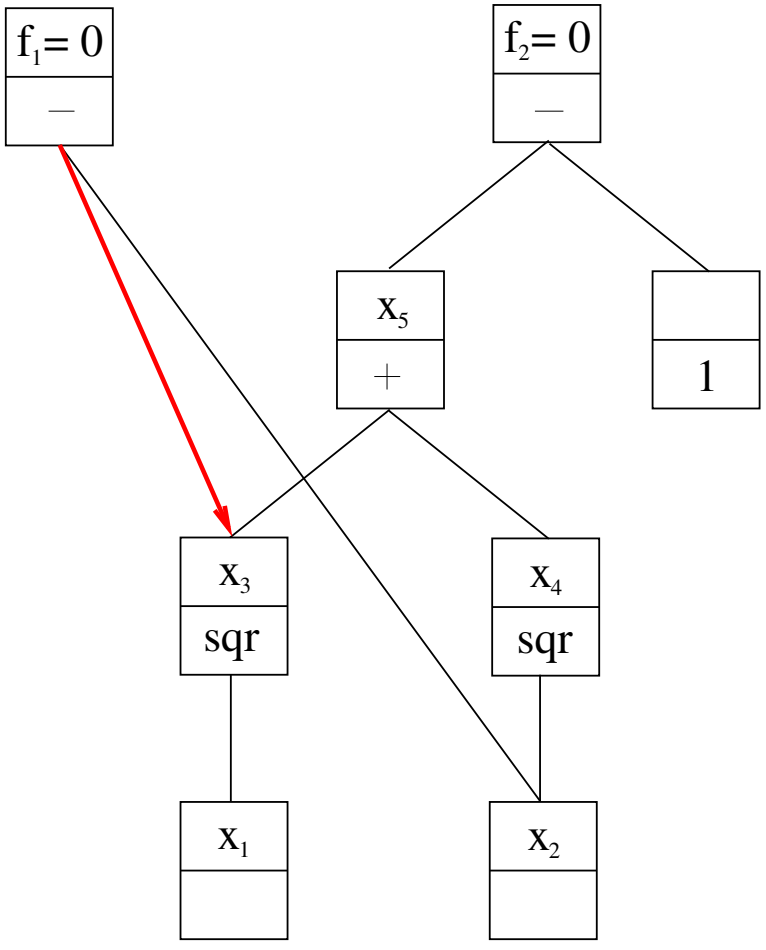
$$\begin{aligned} x &= (z + y) \cap x \\ y &= (x - z) \cap y \end{aligned}$$





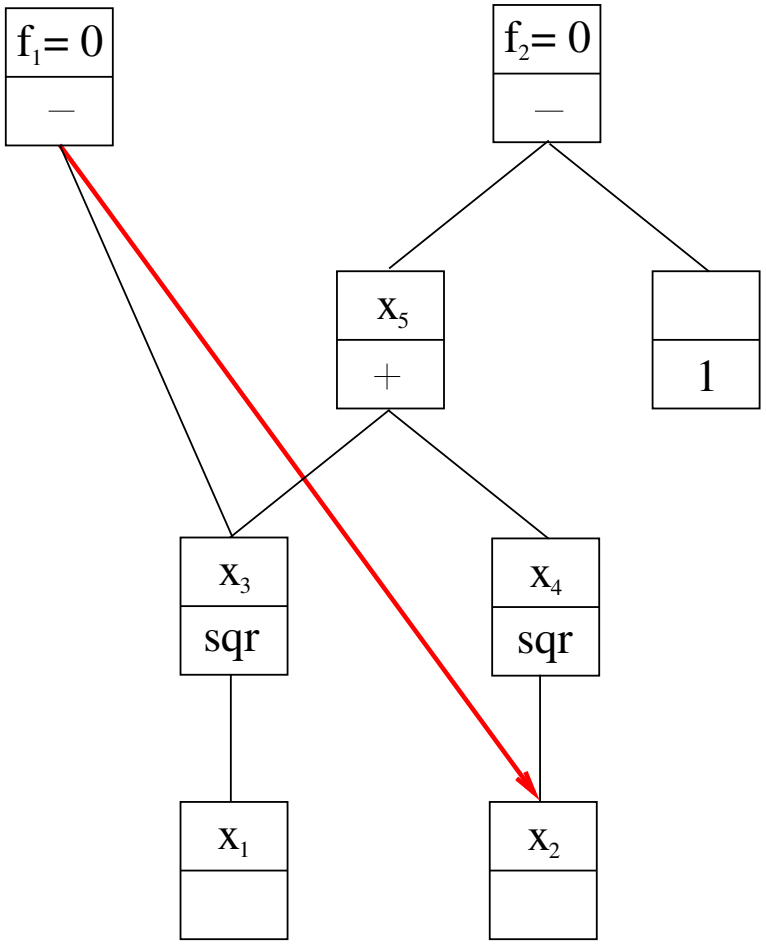
$$
\begin{aligned}
x_1 &= [-2, 2] \\
x_2 &= [-2, 2]
\end{aligned}
$$

$$
\begin{aligned}
0 &= x_5 - 1 \\
0 &= x_3 - x_2 \\
x_3 &= x_1^2 = [0, 2] \\
x_4 &= x_2^2 = [0, 4] \\
x_5 &= x_3 + x_4 = [0, 8]
\end{aligned}
$$



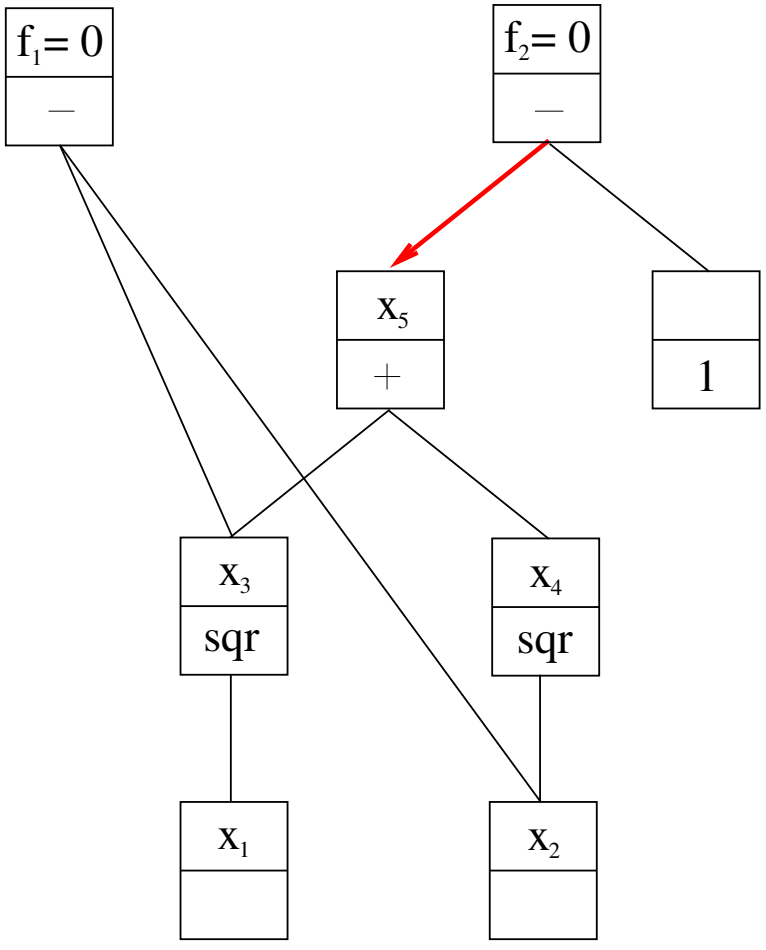
$$
\begin{array}{rcll}
 & & x_1 & = [-2,2] \\
 & & x_2 & = [-2,2] \\
0 & = & x_5 - 1 & \\
0 & = & x_3 - x_2 & \\
x_3 & = & x_1^2 & = [0,2] \\
x_4 & = & x_2^2 & = [0,4] \\
x_5 & = & x_3 + x_4 & = [0,8]
\end{array}
$$

$$
\begin{array}{rcl}
x_3 & = & (x_2 + 0) \cap x_3 \\
 & = & [-2,2] \cap [0,4] \\
 & = & [0,2]
\end{array}
$$



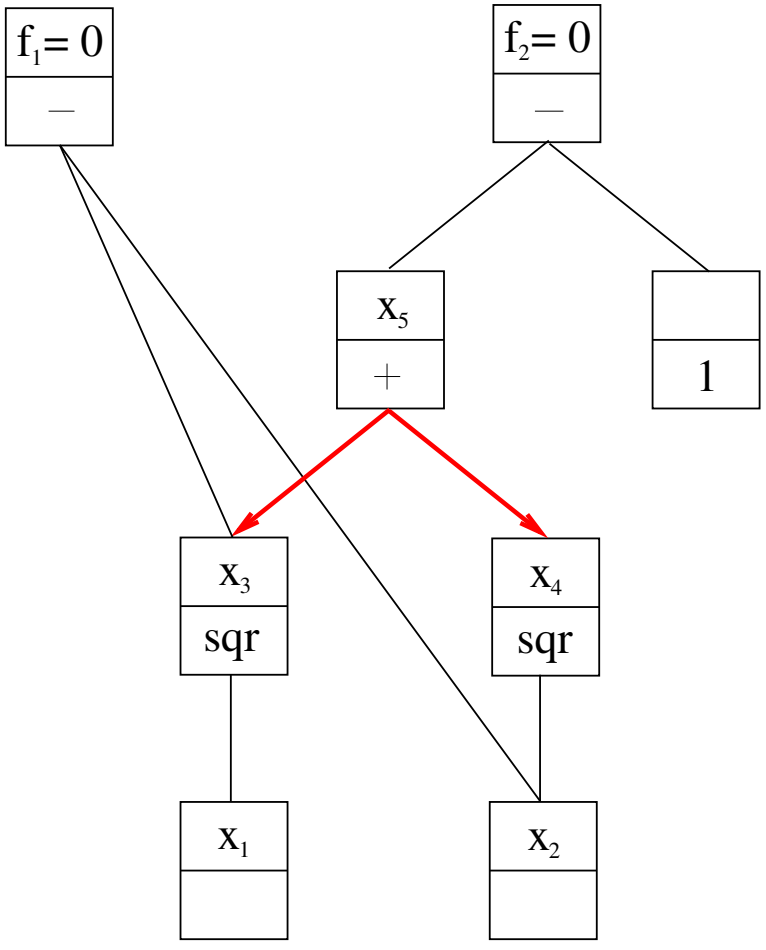
$$
\begin{array}{rcll}
 & & x_1 & = [-2,2] \\
 & & x_2 & = [0,2] \\
0 & = & x_5 - 1 & \\
0 & = & x_3 - x_2 & \\
x_3 & = & x_1^2 & = [0,2] \\
x_4 & = & x_2^2 & = [0,4] \\
x_5 & = & x_3 + x_4 & = [0,8]
\end{array}
$$

$$
\begin{array}{rcl}
x_2 & = & (x_3 - 0) \cap x_2 \\
 & = & [0,2] \cap [-2,2] \\
 & = & [0,2]
\end{array}
$$



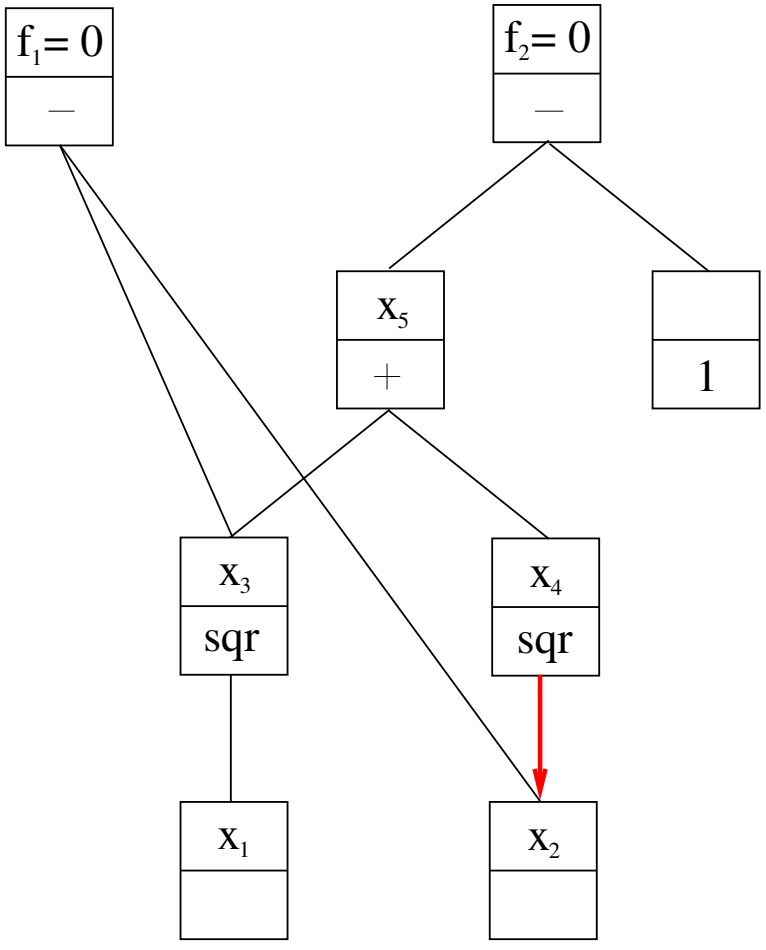
$$
\begin{aligned}
x_1 &= [-2, 2] \\
x_2 &= [0, 2] \\
0 &= x_5 - 1 \\
0 &= x_3 - x_2 \\
x_3 &= x_1^2 = [0, 2] \\
x_4 &= x_2^2 = [0, 4] \\
x_5 &= x_3 + x_4 = [1, 1]
\end{aligned}
$$

$$
\begin{aligned}
x_5 &= [1, 1] \cap [0, 8] \\
&= [1, 1]
\end{aligned}
$$



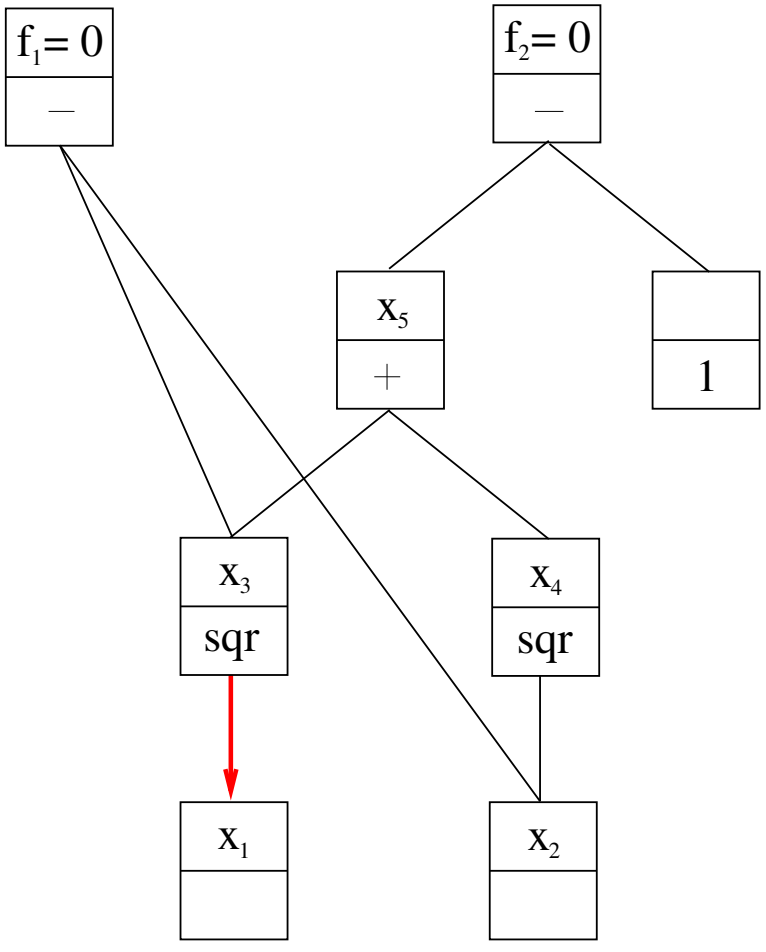
$$
\begin{array}{rcll}
 & & x_1 & = [-2,2] \\
 & & x_2 & = [0,2] \\
0 & = & x_5 - 1 & \\
0 & = & x_3 - x_2 & \\
x_3 & = & x_1^2 & = [0,1] \\
x_4 & = & x_2^2 & = [0,1] \\
x_5 & = & x_3 + x_4 & = [1,1]
\end{array}
$$

$$
\begin{array}{rcll}
x_3 & = & (x_5 - x_4) \cap x_3 & = [0,1] \\
x_4 & = & (x_5 - x_3) \cap x_4 & = [0,1]
\end{array}
$$



$$
\begin{array}{rcll}
 & & x_1 & = [-2,2] \\
 & & x_2 & = [0,1] \\
0 & = & x_5 - 1 & \\
0 & = & x_3 - x_2 & \\
x_3 & = & x_1^2 & = [0,1] \\
x_4 & = & x_2^2 & = [0,1] \\
x_5 & = & x_3 + x_4 & = [1,1]
\end{array}
$$

$$x_2 = \pm\sqrt{x_4} \cap x_2 = [0,1]$$



$$
\begin{array}{rcll}
 & & x_1 & = [-1,1] \\
 & & x_2 & = [0,1] \\
0 & = & x_5 - 1 & \\
0 & = & x_3 - x_2 & \\
x_3 & = & x_1^2 & = [0,1] \\
x_4 & = & x_2^2 & = [0,1] \\
x_5 & = & x_3 + x_4 & = [1,1]
\end{array}
$$

$$x_1 = \pm\sqrt{x_3} \cap x_1 = [-1,1]$$

Ergebnis:

$$x_1 = [-2,2] \rightarrow [-1,1]$$
$$x_2 = [-2,2] \rightarrow [0,1]$$

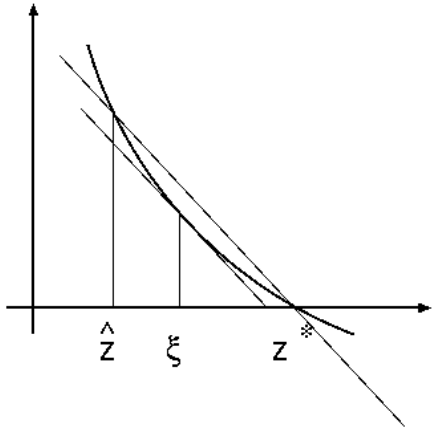
Newton Methode

Idee: Anwendung der Newton Methode auf Intervalle

- Annahme: z^* ist Nullstelle von f im Intervall $[z]$
- f' ist die Ableitung von f über z
- 1. Ordnung Taylor Entwicklung um einen beliebigen Punkt $\hat{z} \in [z]$ ergibt (Mittelwertsatz)

$$0 = f(z^*) = f(\hat{z}) + \cdot f'(\xi) * (z^* - \hat{z})$$

wobei ξ zwischen $\hat{z}$ und z^* liegt



- Ersetze diese Gleichung durch eine Intervallgleichung.
- $[f']$ ist das Intervall, welches alle Ableitungswerte von f über $[z]$ einschließt
- $[z]$ ersetzt die unbekannte Nullstelle z^*
- Es ergibt sich die lineare Gleichung

$$[f'](([z] - \hat{z}) = -f(\hat{z})$$

- Bei einem Gleichungssystem mit mehreren Unbekannten ist $[f']$ eine Matrix, $[z]$ und $f(\hat{z})$ sind Vektoren.
- Löse das System mit der Newton-Gauß-Seidel Methode

Viele weitere Methoden sind in SONIC implementiert

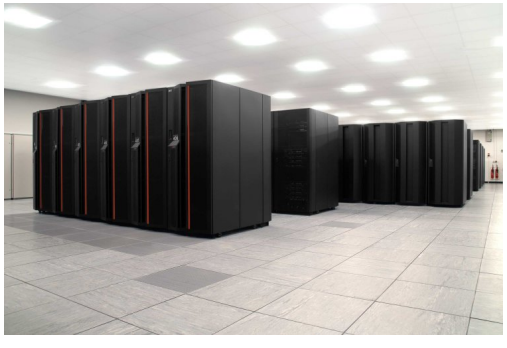
3. Parallelisierung

3.1. Warum in Edinburgh?

- Schöne Stadt
- 4 Universitäten
- Edinburgh Parallel Computing Centre EPCC
  - 70 feste Mitarbeiter für Projekte auf Parallelrechnern
  - einige hundert Gäste und Zeitstellen
  - Mitglied im Europa-Projekt HPC-Europa

Ein paar Rechner am EPCC

- Hektor
  - Cray-XP4 mit 5664 dual-core Opteron CPUs
  - 33.2 TByte RAM
  - 576 TByte Platten
  - 54.65 TFlops HPL
- HPC-X
  - IBM eServer mit 2560 POWER5 CPUs
  - 12,94 Tflops HPL



Noch ein paar Rechner am EPCC

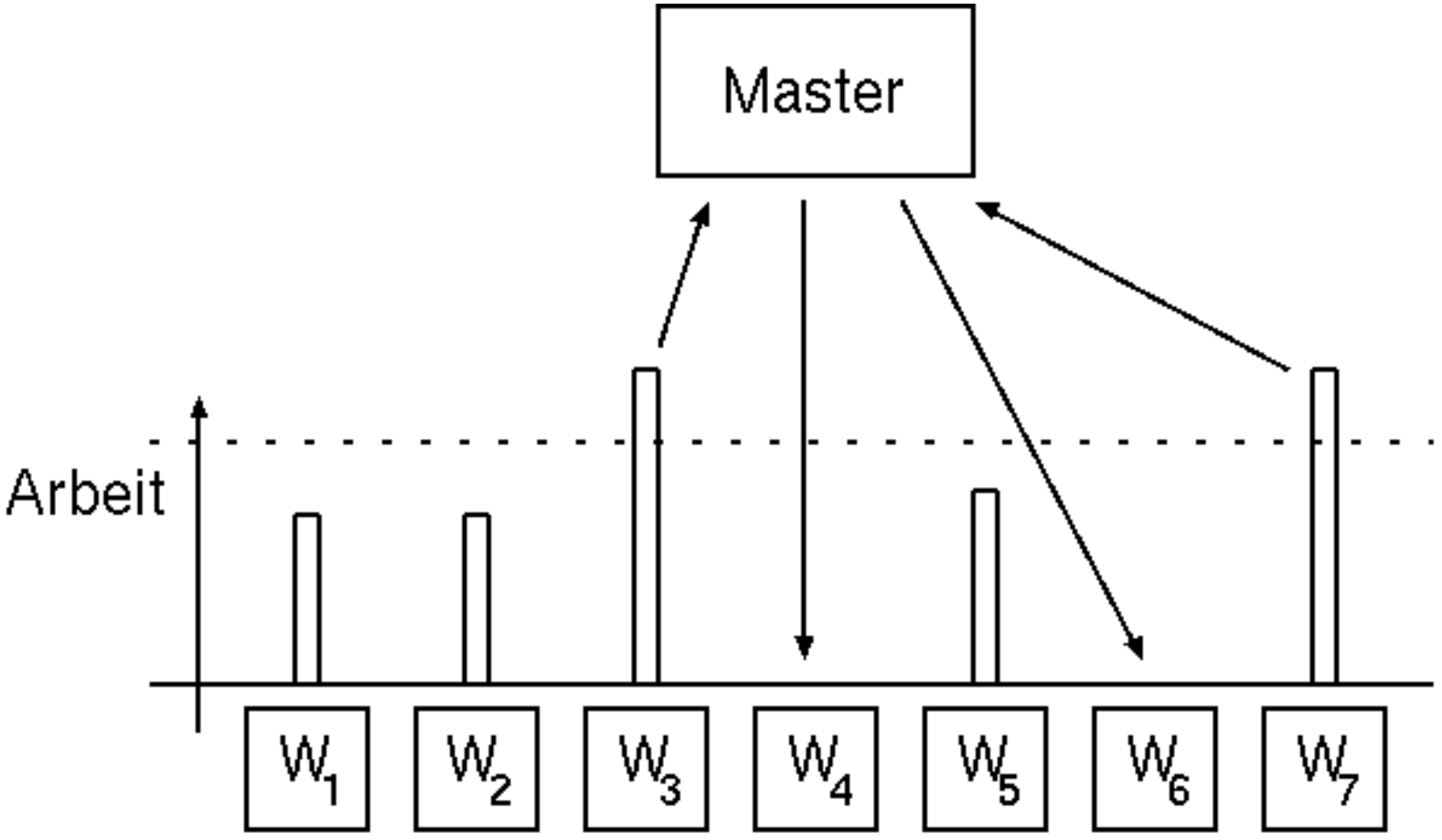
- QCDOC
  - IBM-Eigenbau mit 12000 Power440 CPUs
  - 10 TFlops HPL
- BlueGene seit 2005
  - IBM mit 2048 Power440 CPUs
  - 4.71 TFlops HPL

3.2. Parallelisierung des Algorithmus

- Nutze, dass die Analysen der Boxen voneinander unabhängig sind und lasse jeden Prozessor eine Teilbox berechnen
- Einfach zu implementieren, aber
  - einige Boxen können schnell ausgeschlossen werden
    $\Rightarrow$ Prozessoren sind idle
  - einige Boxen können sehr viele neue Boxen erzeugen
    $\Rightarrow$ Prozessoren haben zu wenig Speicher

Standard Master-Worker Ansatz

- **Worker**
  - Bearbeite eine "private" Teilliste von Boxen mit einer maximalen Länge l_{max}.
  - Schicke zu viele Boxen zum Master
- **Master**
  - Sammele die Boxen ein und verteile sie an die untätigen Worker.

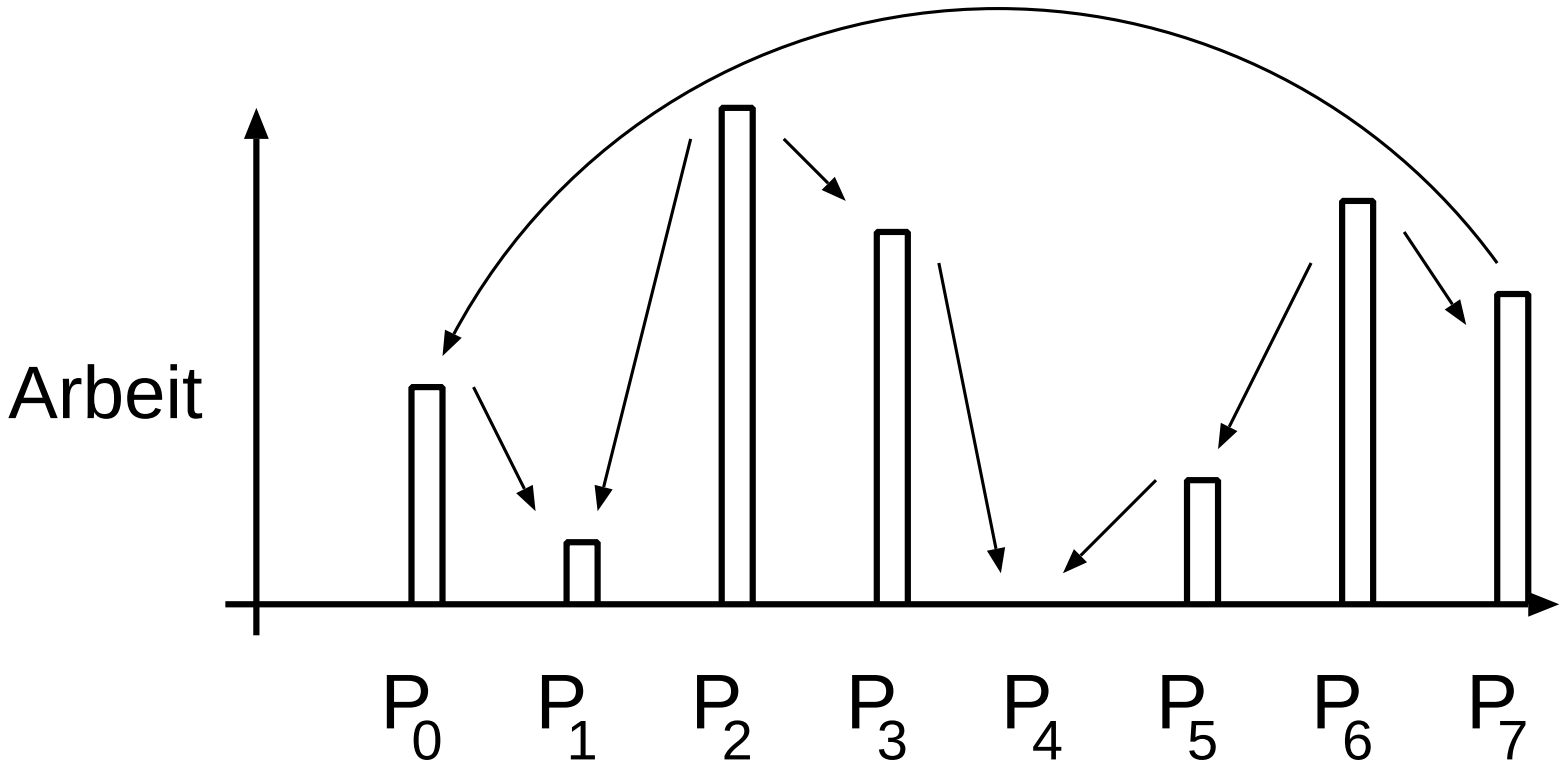


Funktioniert nur bei “mittlerer” Anzahl von Prozessoren gut (ein paar Hundert)

Diffusionsalgorithmus

Prinzip:

- Ordne Prozessoren in einem Gitter an.
- Initialisiere jeden Prozessor mit einer Menge von Arbeit (hier Boxen).
- Tausche Informationen über die noch zu bearbeitende Arbeitsmenge aus (hier Anzahl der zu analysierenden Boxen).
- Lasse Arbeit von Prozessoren mit hoher Arbeitsmenge zu solchen mit niedriger Arbeitsmenge “diffundieren”.


Arbeit
P0
P1
P2
P3
P4
P5
P6
P7

Ein paar zu klärende Probleme

- Wer schickt wann welche Nachricht an wen?
- Was passiert, wenn der andere Prozess nicht empfangsbereit ist?
- Was passiert, wenn noch Nachrichten im Netzwerk sind?
- Fragt der Prozess ohne Arbeit seine Nachbarn nach Arbeit oder schickt ein Prozess mit zuviel Arbeit einen Teil an einen oder an alle seine Nachbarn?
- Wie können die Prozesse schnell genug mit Arbeit versorgt werden, falls mehreren benachbarten Prozesse gleichzeitig die Arbeit ausgeht?
- Wie kann sicher gestellt werden, dass alle Boxen bearbeitet wurden **und** keine Nachrichten mit Arbeit mehr in Netz hängen?

Lösungsansatz

- Ordne die Prozessoren in einem d-dimensionalen Gitter an, wobei d von der Anzahl der verwendeten Prozessoren abhängt.
- Verwende ausschließlich nicht-blockierende Kommunikation.
- Verschicke Nachrichten nur dann, wenn das Kommunikationsnetz leer ist.
- Schicke nur Arbeit zu einem Nachbarn, wenn der Unterschied in Arbeit groß ist.
- Führe an zentraler Stelle Buch über die Menge der Nachrichten im Netz, um einen korrekten Abschluss sicher zu stellen.

Performance

- Bei Systemen mit sehr ungleichmäßiger Lastverteilung muss die Dimension des Prozessor-Gitters hoch sein, um eine ausreichende Diffusionsgeschwindigkeit der Boxen zu erreichen.
- Der zentraler Algorithmus zur Überprüfung der verbleibenden Arbeitsmenge beeinträchtigt die Performance nicht.
- **Ausgezeichnete Skalierung der Rechengeschwindigkeit (maximal 10% Verlust durch Kommunikation) bis zu 2048 Prozessoren.**

4. Anwendung: Sphärisches t-Design

$t = 3$: Insgesamt 19 Gleichungen, bestehend aus einer Summe von Termen, jeder Term ist ein Produkt aus maximal $2t = 6$ Sinus/Cosinus-Funktionen.

Beispiel für 3 Sinus/Cosinus-Paare:

$$\begin{aligned}
\sin(\theta_1) * \cos(\phi_1) + \sin(\theta_2) * \cos(\phi_2) + \sin(\theta_3) * \cos(\phi_3) &= 0 \\
\cos(\theta_1) + \cos(\theta_2) + \cos(\theta_3) &= 0 \\
(\sin(\theta_1) * \cos(\phi_1))^2 + (\sin(\theta_2) * \cos(\phi_2))^2 & \\
+(\sin(\theta_3) * \cos(\phi_3))^2 - 1 &= 0 \\
\sin(\theta_1) * \cos(\phi_1) * \cos(\theta_1) + \sin(\theta_2) * \cos(\phi_2) * \cos(\theta_2) & \\
+ \sin(\theta_3) * \cos(\phi_3) * \cos(\theta_3) &= 0 \\
\ldots &= 0
\end{aligned}$$

Allgemeine Gleichung:

$$N\gamma_{i,j,k} = \sum_{i=1}^{N} (\sin(\theta_i) * \cos(\phi_i))^k (\sin(\theta_i) * \sin(\phi_i))^l \cos(\phi_i)^m, k+l+m \leq t$$

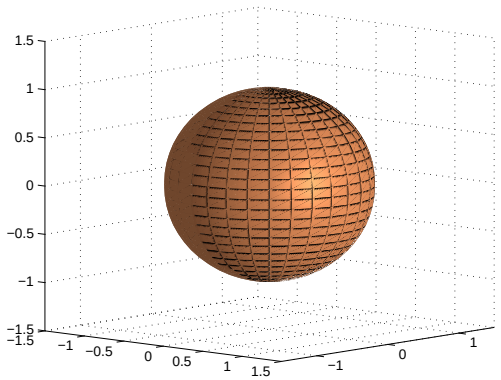
Bedeutung:

- Ein Winkelpaar θ_i und ϕ_i entspricht den Winkelkoordinaten eines Punktes auf einer Kugel.
- Die Anzahl der Winkelpaare N und damit der Summanden entspricht der Anzahl von Punkten auf der Kugel.
- Die maximale Anzahl t in einem Term entspricht dem Grad eines Polynoms in den Koordinaten x_i, y_i und z_i eines Punktes.
- Hat das Gleichungssystem eine Lösung, spricht man von einen N-Punkte sphärischem t-Design.

Eine Seite nur für Mathematiker:

S sei die Kugeloberfläche einer Einheitskugel in 3 Dimensionen:

$$S = \{\mathbf{x} \in \mathbb{R}^3 : \|\mathbf{x}\|_2 = 1\}.$$



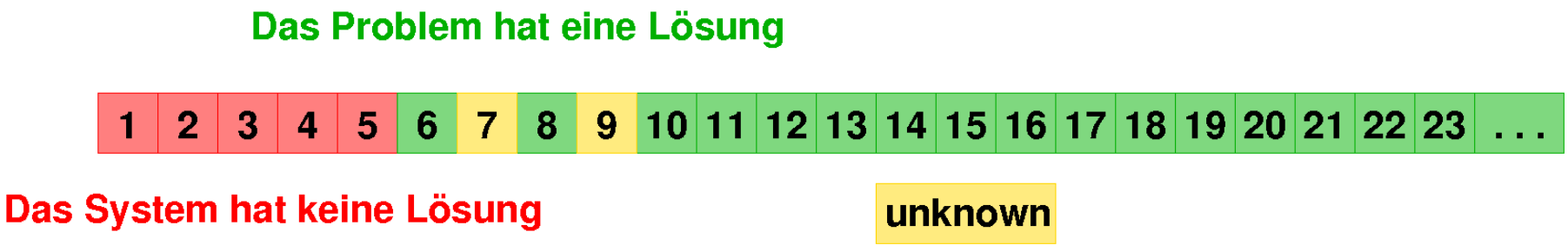
Unter einem **sphärischen t-Design** versteht man einen Satz von N Punkten $\mathbf{x}_1, \ldots, \mathbf{x}_N$ auf der Kugeloberfläche, so dass die numerische Quadratur-Regel

$$\int_S p(\mathbf{x}) d\mu \approx \frac{4\pi}{N} \cdot \sum_{i=1}^{N} p(\mathbf{x}_i)$$

exakt ist für alle Polynome $p = p(x, y, z)$ vom Grad $\leq t$.

Unser Testproblem für SONIC:

- Für $t = 3$ (maximal 6 Sinus/Cosinus-Faktoren pro Summand) gilt für folgende Punktzahl N (Anzahl der Summanden)



- Berechne die beiden offenen Fälle für N=7 und N=9 Punkte

Ergebnisse mit SONIC:

$N = 7$: 19 Gleichungen, 14(11) Unbekannte
Boxen untersucht: 922.656
Rechenzeit ≈ 1 Stunde
Lösungen: 0

$N = 9$: 19 Gleichungen, 18(15) Unbekannte
Boxen untersucht: 385.821.409.493
Rechenzeit ≈ 450.000 CPU-Stunden [1]
Lösungen: 0

Es existiert kein 7/9-Punkte sphärisches 3-Design.

[1] Diese Rechnung wurde nur möglich durch die Unterstützung der Grid-Gruppe der Uni-Wuppertal mit ihrem 512-Kerne Grid-Cluster

Zusammenfassung

- Mit SONIC steht in Tool zu Verfügung, mit dem sehr effizient nicht-lineare Gleichungssysteme oder Optimierungsprobleme überprüfbar gelöst werden können.
- Der am EPCC entwickelte Algorithmus zur Parallelisierung des Problems zeigt sehr hohe Effizienz bis zu einer hohen Anzahl von Prozessoren.
- Die Anwendung auf das t-Design-Problem konnte erfolgreich eine Lücke offener Werte schließen.
- Es ist wichtig, sich regelmäßig mit aktuellen Forschungs- und Entwicklungsgebieten zu beschäftigen, auch um aktuelle Gebiete in die Lehre einfließen zu lassen.

Literatur

1. M. Mönnigmann, W. Marquardt, C.H. Bischof, T. Beelitz, B. Lang, and P. Willems: *A Hybrid Approach for Efficient Robust Design of Dynamic Systems*, SIAM Rev. 49(2), 236–254 (2007).
2. C. Xu, F. Lau: *Load Balancing in Parallel Computers, Theory and Practice*, Kluwer Academic Publishers (1997).
3. P. Ueberholz, P. Willems, M. Bull and B. Lang: *Non-Blocking Load Balancing for Branch-and-Bound Algorithms*, erscheint in Proceedings PARA 08
4. T. Beelitz, B. Lang, P. Ueberholz and P. Willems: *Closing the Case $t = 3$ for 3D Sperical t-Design Using a Result-Verifying Nonlinear Solver*, erscheint in Reliable Computing