Verteilte und Parallele Programmierung

Mit Virtuellen Maschinen

PD Stefan Bosse

Universität Bremen - FB Mathematik und Informatik

Zelluläre Automaten

Einführung von Grundprinzipien beim Entwurfs von VP Systemen

Zelluläre Automaten als einfache VP Architektur und Datenverarbeitungsmodell

Grundprinzipen beim Entwurf von Verteilten Systemen

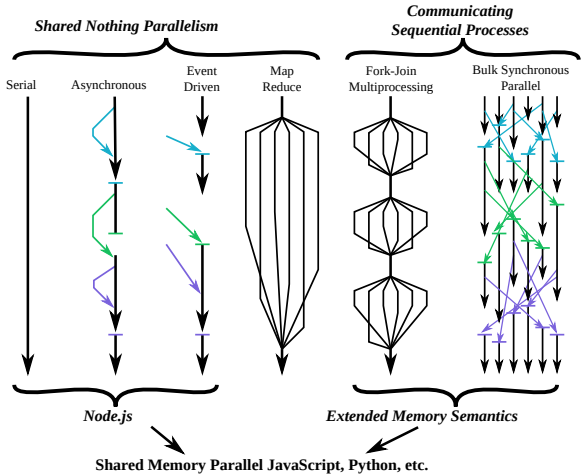
Nach Werner Vogel, Amazon [101]

1. **Dezentralisierung** ⇒ Verbesserung der Skalierung;
2. **Asynchronität** ⇒ Ein System schreitet unter "allen" Umständen voran;
3. **Autonomie** ⇒ Das System besteht aus Einheiten die autonom Entscheidungen mit lokalen Informationen treffen können;
4. **Lokalität** ⇒ Systemeinheiten arbeiten auf lokalen Daten;
5. **Wettbewerb** (Konk.) ⇒ Die Einheiten sind so gebaut, dass kaum oder kein Wettbewerb um geteilte Ressourcen auftritt (gesteuerter Wettbewerb);
6. **Fehlertoleranz** ⇒ Der Ausfall einzelner Einheiten ist der Normalfall und beeinträchtigt das Systemevrhalten nicht (und das Endergebnis);
7. **Parallelität** ⇒ P. erhöht die Performanz und Robustheit;

8. **Elementarzellenansatz** ⇒ Dekomposition eines großen und komplexen Systemes in viele kleine und einfache Einheiten;
9. **Symmetrie** ⇒ Die Einheiten des Systems sind identisch in Hinsicht auf Funktionalität und benötigen keine spezifische Konfiguration;
10. **Einfachheit** ⇒ KISS (Keep it Simple and Safe), die Elementarzellen sollen durch einfache Modelle und Algorithmen implementiert werden und möglichst geringe Abhängigkeiten untereinander besitzen.

Zelluläre Automaten vereinen fast alle dieser Grundprinzipien.

Parallele Semantik



[mogill.github.io/ems/Docs/index.html]

Abb. 1. Arten von Parallelität

Kommunikationsmodelle

Ungeteilte Parallelität

Die einzelnen Prozesse werden nebenläufig ohne gemeinsam geteilte Ressourcen ausgeführt (aber partitionierte Datenverteilung und Datenzusammenführung möglich → Map&Reduce Verfahren)

Communicating Sequential Processes (CSP)

Prozesse werden parallel aber mit geteilten Ressourcen und Synchronisation ausgeführt

Prozessmodelle

Fork-Join Multiprozesse

Die Ausführung beginnt mit einem einzelnen Prozess, der bei Bedarf neue Prozesse erstellt und dann auf deren Terminierung wartet.

Bulk Synchron Parallel

Die Ausführung beginnt mit einer Menge von Prozessen, die das Programm am Haupteinstiegspunkt startet und alle Anweisungen ausführt.

Benutzerdefiniert

Parallelität durch Ad-hoc-Prozesse und heterogene Anwendungen.

Zelluläre Automaten

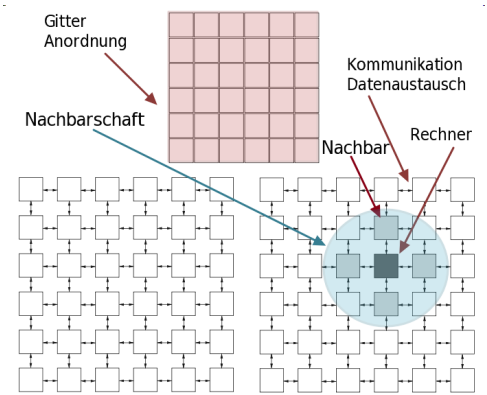
- Zelluläre Automaten besitzen diskrete Zustände und ändern ihren Zustand nur zu diskreten Zeitpunkten
- Einfachstes paralleles bzw. eher verteiltes Berechnungsmodell → **Verteilter Speicher mit expliziter Kommunikation**!
- Netzwerk aus einfachen kommunizierenden Rechnern (Zellen)
- Eine Zelle besitzt eine endliche (kleine) Menge von Zuständen $\sigma = \{ s_1,..,s_n \}$.
- Eine Berechnung mit Eingabe- und internen Daten (Perzeption und innerer Zustand) führt i.A. zu einer Änderung des Zustandes der Zelle → es gibt einen Zustandsübergang

- Übergangsregeln Φ können aus einfachen arithmetischen Operationen (Funktionen) bestehen, und beziehen den Zustand der Zellen aus der Nachbarschaft $N(i,j)=\{\sigma(i \pm \Delta_i, j \pm \Delta_j) \mid \Delta_i \neq 0 \vee \Delta_j \neq 0\}$ mit ein (durch Kommunikation):

$$\sigma_{i,j}(t+1) = \Phi(\{\sigma_{k,l}(t) | \sigma_{k,l}(t) \in N\})$$

Zelluläre Automaten

Architektur und Grundprinzip

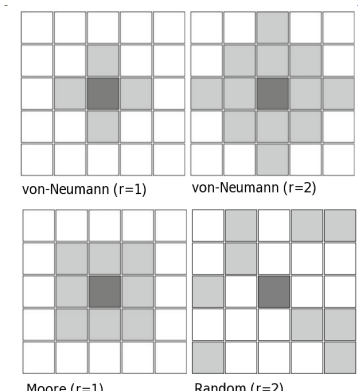


[Hoekstra,SCSCA,2010]

Abb. 2. Zellulärer Automat als Netzwerk aus einfachen kommunizierenden Berechnungseinheiten

Zelluläre Automaten

- Nachbarschaftsrelationen sind auch bei Agenten wichtige Eigenschaft



[Hoekstra,SCSCA,2010]

Abb. 3. Verschiedene Nachbarschaftsrelationen

Zelluläre Automaten

- Die Nachbarschaftskonnektivität und die Anzahl der Zustände je Zelle bestimmen die Anzahl der möglichen lokalen Regeln die zu einem Zustandsübergang in der Nachbarschaftsgruppe führen → wird sehr schnell sehr groß!!

Number of states σ	Number of neighbors n	σ^{σ^n}	Number of rules N_r
2	2	2^{2^2}	16
2	3	2^{2^3}	256
2	5	2^{2^5}	4 294 967 296
2	10	$2^{2^{10}}$	$1.797 \cdot 10^{308}$
5	2	5^{5^2}	$2.98 \cdot 10^{17}$
5	3	5^{5^3}	$2.35 \cdot 10^{87}$
5	5	5^{5^5}	$1.91 \cdot 10^{2184}$
10	2	10^{10^2}	10^{100}
10	3	10^{10^3}	10^{1000}
10	5	10^{10^5}	10^{100000}

[Hoekstra,SCSCA,2010]

Abb. 4. Anzahl der Regeln eines ZA in Abhängigkeit der Anzahl der Zustände pro Zelle und des Verbindungsgrades der Zellen

Zelluläre Automaten

Parallele Datenverarbeitung

- Den ZA kann man als verteilten Rechner betrachten:
  - Alle Zellen führen ihren Zustandsübergang parallel durch, d.h., die Anwendung der Übergangsfunktion $\Phi(\sigma)$
  - Daraus können sich unerwartete globale Zustände ergeben da der Zustand einer Zelle (i,j) von den Zuständen der Nachbarn abhängt, jedoch die Nachbarzellen ebenso von dem Zustand dieser Zelle abhängen
  - Ohne Synchronisation u.U. randomisierte Ergebnisse und Zustandsübergänge!!!

- Betrachtet man den ZA als Simulator:
  - Die Zellen werden der Reihe nach (sequenziell) in einer bestimmter Ausbreitungsrichtung (z.B. von links unten nach rechts oben) der Reihe nach aktiviert!
- Es gibt keinen gemeinsamen globalen Speicher; durch die Kommunikation mit den Nachbarn (ersteinmal nur lesender Zugriff auf Nachbarspeicher) erhält man lokalen Gruppenspeicher (glokaler Bereich)

Zelluläre Automaten

Zustände

- Eine Zelle des ZA ist ein endlicher Zustandsautomat
- Ein Zustandsautomat hat einen Kontrollzustand γ und Datenzustand δ, d.h., $\sigma(t)=\langle\gamma,\delta\rangle(t)$
- Rein "funktionale" Automaten besitzen nur einen Datenzustand der sich durch eine Übergangsfunktion schrittweise ändert (einfachste Zelle), d.h., $\sigma(t)=\delta(t)$

$$\sigma_{i,j}(t+1) = f(\sigma_{i,j}(t))$$

- Die Anzahl möglicher Regeln (also möglicher Zustandsübergänge) ist dann abhängig von der Anzahl der Zustände |σ| und Nachbarn *n*:

$$N_r = |\sigma|^{|\sigma|^n}$$

Zelluläre Automaten

Beispiel: Bildverarbeitung auf ZA

- Die einzelnen Pixel des Bildes (Sensoren) sind den Zellen zugeordnet
- Typische Operationen: Rauschunterdrückung, Kantenfilter, Histogrammberechnung usw.
- Eine Zelle verändert immer nur seinen eigenen Datenwert durch Anwendung einfacher Regeln der Menge Φ mit Nachbarzellendaten

Zelluläre Automaten

Das Problem: Anders als bei klassischen Algorithmen sind diese Regeln nicht bekannt! Diese Transformationsregeln werden daher häufig gelernt (d.h. die geeigneten Regeln z.B. für Kantentendetektion aus der Menge aller möglichen Kombination von Pixeloperationen ausgewählt)

[Rosin,2002]

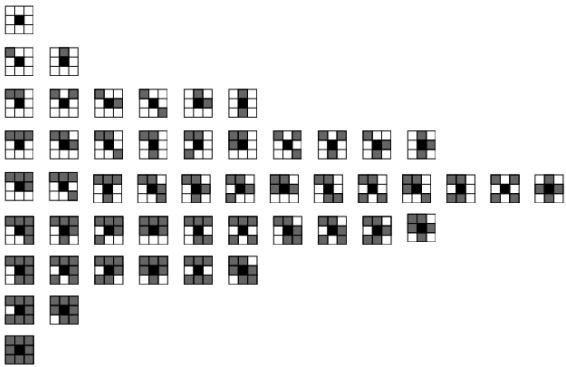


Abb. 5. Alle 51 möglichen Muster (Regeln) damit ein Pixel seinen Wert (binär, SW Bild) invertiert (Mooresche Nachbarschaft mit 8 Nachbarn); gezeigt sind nur zentrale schwarze Pixel, durch Invertierung weitere 51 Regeln für weiße Pixel

Zelluläre Automaten

Rauschunterdrückung

[Rosin,2002]

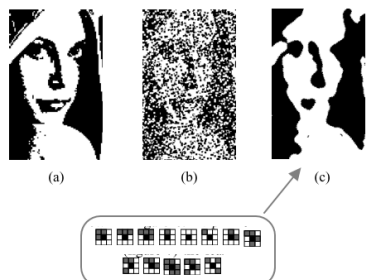


Abb. 6. Rauschunterdückung durch einen binären ZA: (a) Originalbild (b) Verrauschtes Bild (c) Entrauschtes Bild mit dem unten gezeigten Regelsatz

Zelluläre Automaten

Kantendetektion

[Rosin,2002]

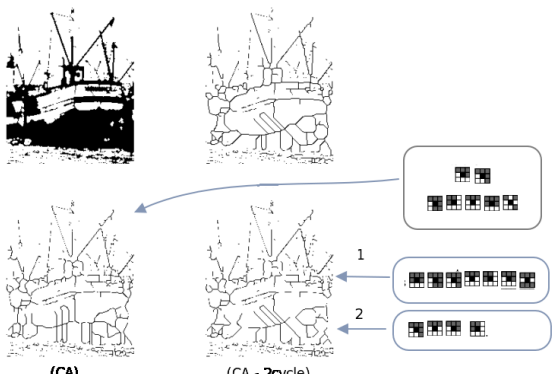


Abb. 7. Kantendetektion (oben,links) Originalbild (oben,rechts) Konv. Alg. (unten, links) CA - ein Zyklus ZA (unten rechts) zwei Zyklen CA

Zelluläre Automaten

Die Programmierung

- Um einen ZA zu programmieren muss dessen Größe (Netzwerk Zeilen und Spalten), die Nachbarschaftsverbindungen (Reichweite und Richtung) sowie die Zustandsübergangsfunktion einer Zelle beschrieben werden
  - Annahme: Uniformer ZA, d.h. gleiches Φ gilt für alle Zellen!
- Visualisierung mit einer festen Kachelgitterwelt und einer Farbpalette
- Die Zustandsübergangsfunktion ist dreischrittig und unterteilt (optional):
  - Before
  - Activity
  - After

Die Ausführung

- Es gibt einen Scheduler der die Zellen in einer bestimmten oder randomisierten Reihenfolge aktiviert
- Dabei kann es bis zu drei Phasen geben:
  - Voraktivierung (Before),
  - Hauptaktivierung (Activity),
  - Nachaktivierung (After).

> Die Reihenfolge der Zellenaktivierungen kann aufgrund der Nachbarschaftskommunikation zu unetrschiedlichen globalen Zuständen führen!

ZA Modell

```
type model = {
  rows : number, columns : number,
  neighbors: number,
  palette : string [],
  cell : cell class,
  data? : {},
  start : { name:string, distribution:number }
}
type cell class = {
  method before (),
  method activity (neighbors),
  method after (),
  method color () -> number,
  method initialize (x,y),
}
```

Code 1. Modell eines ZA der cellauto Bibliothek (Typsignatur)

ZA Beispiel in Lua

```
local cell=CA.Cell({
  name = "wall",
  size = 4,
  state = { open = 0, wasOpen = 0 }
})
function cell:before ()
  self.wasOpen = self.open
end
function cell:activity (neighbors,x,y)
  local surrounding = self.ask(neighbors, 'count', 'wasOpen', true)
  self.open = (self.wasOpen and surrounding >= 4) or surrounding >= 6
end
function cell:initialize (x,y)
  self.open = self.data[y][x] > 0.40
end
function cell:color ()
  return conditional(self.open,1,2)
end
```

Code 2. Definition der Zellenklasse: Das dynamische Höhlensystem

- Das Modell definiert die:
  - Welt (2D Kachelgitter),
  - Farbpalette,
  - Globale Daten, und den
  - Zellenkonstruktor

```
model = {
  rows = 100,
  columns = 100,
  neighbors = 8,
  palette = {'white','red'},
  scheduler = 'UPRIGHT',  -- two phase all before; all process
  cell = cell,
  data = math.matrix(100,100,math.random),
  start = { name = 'wall', distribution = 100 }
}
```

Code 3. Das ZA Modell in Lua

WorkBook Live

CLEAR LOAD + - ca1 ca2

Zusammenfassung

- Verteilte und Parallele Systeme (VP) basieren auf einem gemeinsamen Satz von Entwurfsregeln → KISS Prinzip!
- Zelluläre Automaten sind ein Beispiel für eine parallele und verteilte Datenverarbeitungsarchitektur nach dem KISS Prinzip:
  - Zerlegung eines komplexen Problems auf einfache kommunizierende Elementarzellen
  - Abbildung von Datenmatrizen auf Zellengitternetzwerk
  - Jede Zelle verarbeitet nur lokale Daten
  - Jede Zelle kommuniziert nur mit den unmittelbaren Nachbarn (kurzreichweitige Komm. und Datenabhängigkeit)
- ZA können sequenziell simuliert werden (u.A. mit Asynchronität)
- ZA sind Virtuelle Maschinen und können Rechencluster bilden!