

Verteilte und Parallele Programmierung

Mit Virtuellen Maschinen

PD Stefan Bosse

Universität Bremen - FB Mathematik und Informatik

Parallele und Verteilte Zelluläre Automaten

Wie kann das ZA Modell für die Parallelisierung und Verteilung genutzt werden?

Wie können geteilte und verteilte Speichermodelle sinnvoll eingesetzt werden?

Wie kann CALUA parallelisiert werden?

Das 1:1 Modell

- Die einzelnen Zellen eines ZA sind zunächst völlig unabhängig voneinander → **Kontrollpfadparallelität**
- Die **Datenabhängigkeit** einer Zelle ist auf die Zellen seiner unmittelbaren Umgebung beschränkt → **Kurzreichweitige Datenabhängigkeit**
- **Synchronisation** erfolgt primär (implizit) durch einen **zentralen Takt**, der aber nur die einzelnen Phasen des ZA (*before, activity, after*) einleitet.
- Weitere implizite Synchronisation beim Zugriff auf Zustand (Variablen) von Nachbarzellen

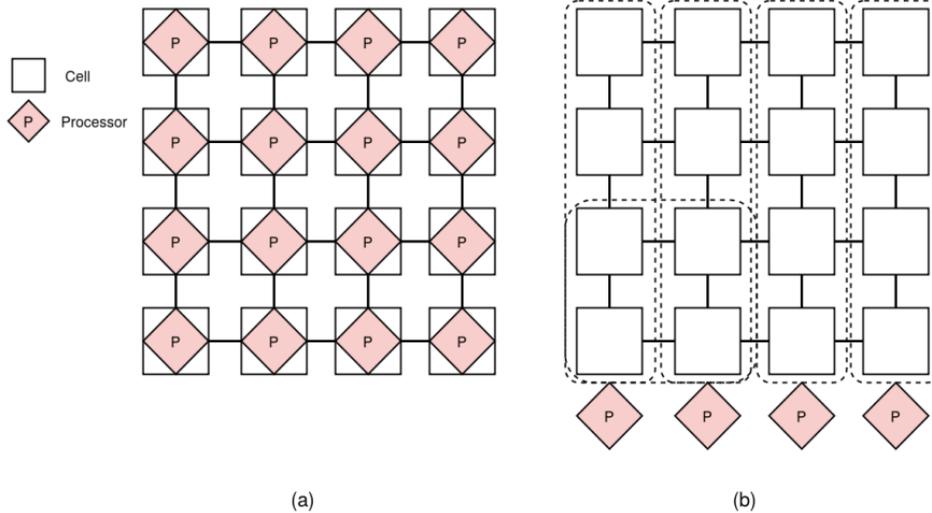


Abb. 1. (a) Das parallele (oder verteilte) 1:1 ZA Modell wo jede Zelle von einem physischen Prozessor ausgeführt wird → Verteiltes Speichermodell (b) Das partitionierte 1:N ZA Modell wo ein Bereich des ZA von einem Prozessor ausgeführt wird → Geteiltes und verteiltes Speichermodell

Das Partitionsmodell

- Partitionierung der Zellen des ZA in parallele Felder:
 - Jedes Feld besteht aus einer Gruppe aus Zellen $F_i = \{z \in Z\} \subset Z$ für sich: Ein P mit SM , Zellen eng gekoppelt
 - Alle Felder: Verteilter Zellenrechner, Multi- P , Felder lose gekoppelt über Verteilten und geteilten Speicher (DSM)
- Man unterscheidet:
 - Kernbereiche eines Feldes (Untergruppe von Zellen aus F) mit reinem SM Modell, und
 - Überlappende Randbereiche mit DM Modell
- Es gibt weiterhin einen gemeinsamen Takt (Synch.)

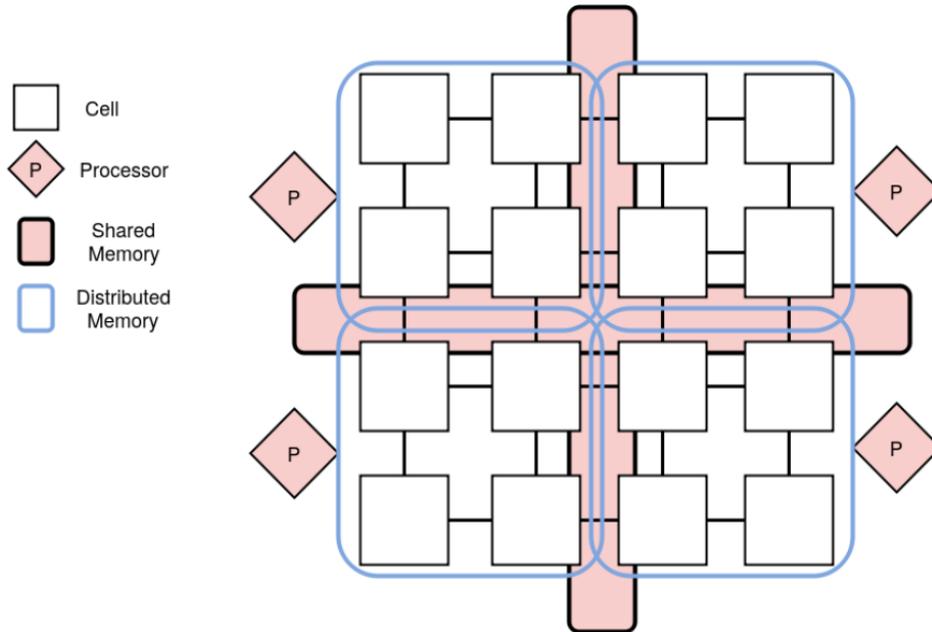


Abb. 2. Partitionierter ZA mit gemischten SM und DSM Modellen (Kern- und Randbereiche)

Parallele Programmierung des ZA

Da das Speichermodell der Kern- und Randzellen unterschiedlich ist müsste explizit bei der Programmierung zwischen Zellen im eigenen Feld und im Nachbarfeldern unterschieden werden!

- Die Kommunikation bei Zellen innerhalb eines Feldes ist ein direkter Speicherzugriff,
 - bei verteilten Feldern (physisch getrennte Rechner) ein Nachrichtenversand (DSM),
 - und bei SM auf einem Parallelrechner (Multi-core Rechner) wieder ein direkter Speicherzugriff, aber bei Verwendung einer VM auf ein spezielles geteiltes Speicherobjekt!

Monitore

- Um heterogene und nicht einheitliche Programmierung (Siehe Entwurfsprinzipien von Verteilten Systemen!) können **Objektmonitore** eingesetzt werden
 - Sogenannte getter/setter Funktionen überwachen den Zugriff auf Feldern von Objekten (Proxy)
 - Bei Nachbarzellen innerhalb eines Feldes wird direkt auf die Objektvariablen zugegriffen
 - Bei Nachbarzellen außerhalb des Feldes wird entweder Nachrichtenkommunikation verwendet (Achtung: ausführungsblokkierend!) oder bei SM auf ein spezielles geteiltes Speicherobjekt zugegriffen

```
function C:initialize (...)
  for k,v in pairs(parameter.state) do
    if shared then
      self.__state[k]=v
    else
      self[k]=v;
    end
  end
end
function C:__newindex( index, value )
  if index ~= '__state' and self.__state ~= nil and self.__state[index] ~= nil then
    if shared then model.shared[index]:write(value,self.y,self.x)
    else self.__state[index] = value end
  else
    rawset( self, index, value )
  end
end
function C:__index( index )
  if index ~= '__state' and self.__state ~= nil and self.__state[index] ~= nil then
    if shared and then return model.shared[index]:read(self.y,self.x)
    else return self.__state[index] end
  else
    local value = rawget( self, index )
    if value == nil then return rawget( getmetatable(self), index)
    else return value end
  end
end
end
```

Code 1. Lua Objektmonitor mit Objektgetter und Settergateways. Bei geteilten Zellen werden die Zellzustandsvariablen (*state*) in einem Hintergrundobjekt verwaltet.

- Bei einem Zugriff auf die Zustandsvariablen einer Zelle, also z.B. `self.open`, werden die Setter und Getter Funktionen aufgerufen
- Handelt es sich um eine geteilte Randzelle und bei dem Objektzugriff um eine Zustandsvariable (aus der *state* Definition der Zelle), wird ein von allen Randzellen geteilter Speicher verwendet, ansonsten die lokale private Speicher einer Zelle
 - Anstelle des geteilten Speicherzugriffs könnte auch nachrichtenbasierte Kommunikation verwendet werden (*read/write* Remote Procedure Call)

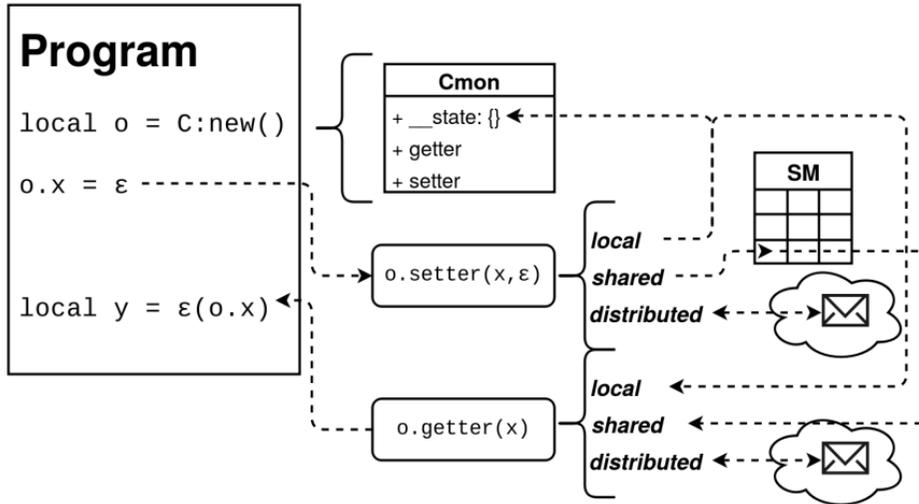


Abb. 3. Objektmonitore implementieren lokalen, geteilten, und verteilten Speicherzugriff: Virtualisierung des Speicherzugriffs durch getter und setter Methoden

- Aber: Sowohl der Zugriff auf geteilten (und geschützten) als auch vor allem auf verteilten Speicher verursacht erhöhte Laufzeitkosten!
- Da die Zelloperationen i.A. geringe Rechenkomplexität aufweisen kann das parallele ZA Modell schnell ineffizient werden (Skalierungsproblem)

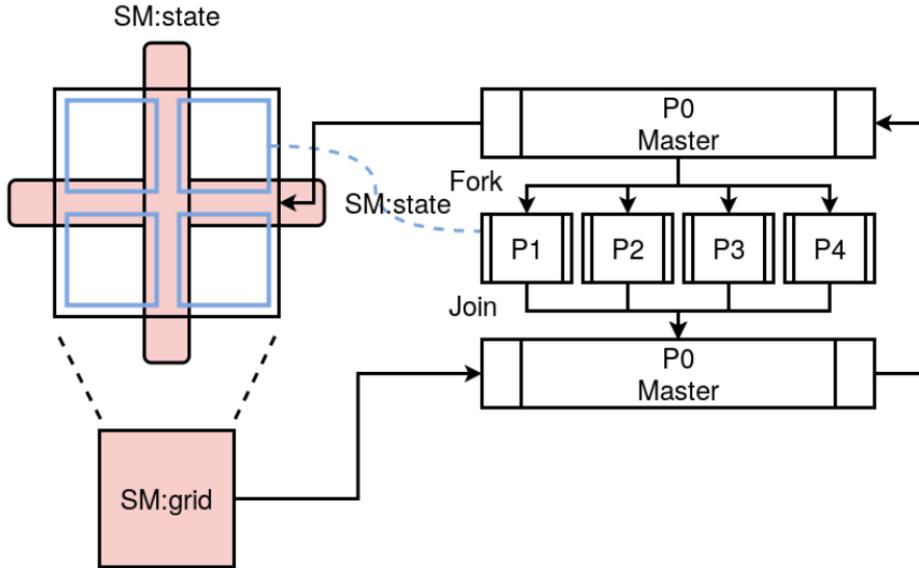


Abb. 4. (Links) Partitionierter ZA mit lokalen und geteilten Speicher in den Randbereichen (Rechts) Prozessablaufmodell: Fork & Join Phasen (jeweils einzeln für *before, activity, after*)

Multi-Computer Parallelrechner: GreenArrays GA144

[www.greenarraychips.com]

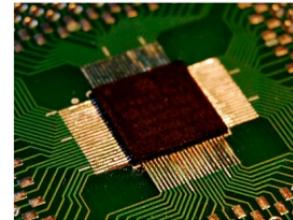
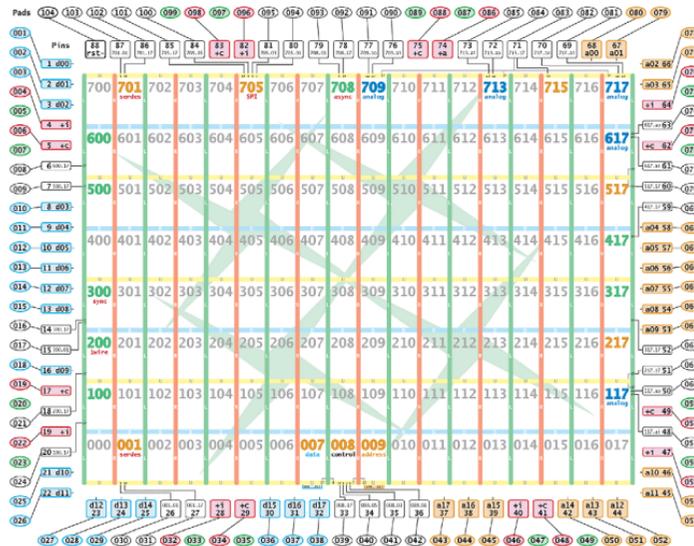


Abb. 5. GA144 Array Rechner (ein Mikrochip!)

Der GA144 Parallelrechner bietet:

- 144 F18A FORTH Prozessoren (FORTH: Interpretersprache, stackbasiert)
 - Anordnung der einzelnen Prozessoren in einem 2D Gitter mit Nachbarknotenkommunikation
 - Pro F18A Knoten: 9k Word RAM
- 96GIPS!
- $14\mu\text{W}$ - 650mW Leistungsaufnahme
- Kein zentraler Takt! Asynchrone Digitallogik

Zusammenfassung

Parallelisierung und Verteilung von Zellulären Automaten erfordert ein hybrides Speichermodell (lokal privat, geteilt, und verteilt)

Durch die kurzreichweitige Datenabhängigkeit der Zellen kann durch Feldpartitionierung der Kommunikationsaufwand deutlich reduziert werden (nur Randzellen benötigen geteiltes Speichermodell mit direkten oder nachrichtenbasierten Zugriff)

Objektmonitore können ein einheitliches Programmierkonzept erzielen und Virtualisierung von dem darunterliegenden Speichermodell erreichen