X10

Seminar Multicore Programming
Uni Passau SS 2009

Alex von Rhein

Übersicht

- Hintergrund
- Entwurfs- Ziele
- Übersicht: Daten- & Kontrollstrukturen
- Grundlagen-Konzepte
- Aufbauende Konzepte
- Sicherheit in X10
- Existierende Implementierungen

Hintergrund: DARPA HPC Challenge

- DARPA: High Productivity Computing Systems Challenge
  - Leistungsteigerung durch Erhöhung der Taktrate nicht mehr möglich
  - Parallele Systeme nötig
  - Produktivität der Programme muss verbessert werden
    - +10x Produktivität bis 2010
  - Produktivität der Programmierer muss verbessert werden
    - „Normale“ Programmierer
    - Software Engineering Teams

Hintergrund: Teilnahme von IBM

- IBM PERCS
  - Productive, Easy-to-use, Reliable Computing System
  - Teilprojekte für Hardware und Software
- X10 als Programmiersprache des Projekts
  - Open Source (Eclipse Public License)
  - Ziele: Skalierbarkeit, Produktivität
  - Eclipse- Plugin
  - http://x10.codehaus.org/

Design Ziele von X10

- Sicherheit
  - Die Sprache durch Design sicher machen
  - Typfehler, ungesetzte Zeiger, Speicherüberläufe, Deadlocks
- Analysierbarkeit
  - Ermöglicht es relativ einfach Werkzeuge für die Sprache zu entwickeln (siehe Java)
- Skalierbarkeit
  - Unterstützt den Entwickler in der Entwicklung von effizienten Programmen
- Flexibilität
  - Auf NUCC Systemen sollen alle Beschleunigungen genutzt werden, die die Hardware bietet.

X10 Fact Sheet

- Ausgangspunkt: State-of-the-Art OOP
  - Java (& Scala)
  - Klassen, Einfachvererbung
  - Werkzeuge/ Entwicklungsumgebung
  - Automatische Speicherbereinigung
  - Alle Änderungen durch Ziele begründen
- Problematische Modelle ersetzen
  - Threads -> Activities (Asynchron)
  - Monitors -> Atomic Sections
  - Barriers -> Clocks
- Konzepte für Parallelität
  - Partitioned Global Address Space (PGAS)
  - Places
  - Distributions
  - Multi- Dimensionale Arrays
  - Aggregierte Operationen (Reduce, Scan)

Daten & Strukturen

Daten

- X10.lang.Double, …
- Wert Typen (Value Types)
  - Immutable
  - Können frei kopiert werden
- Referenz Typen
  - An einen festen Speicherbereich gebunden
- PGAS
  - Globale ID für jedes Objekt
  - Nur lokal zugreifbar
- Verteilte Arrays

Strukturen

- Sequentielle Kontrollstrukturen wie Java (& Scala)
- Strukturen um Parallelität auszudrücken
  - Für Array-Bearbeitung
  - Um entfernte Tasks zu erzeugen
- Modell für Exception Handling

Places & Distributions

- Place
  - (virtuelle) Recheneinheit
  - Gemeinsamer Heap
  - Sollte Prozessoren entsprechen
  - `x10.lang.place`
- Die (virtuellen) Places werden von der Deployment- Schicht auf die konkrete Architektur abgebildet

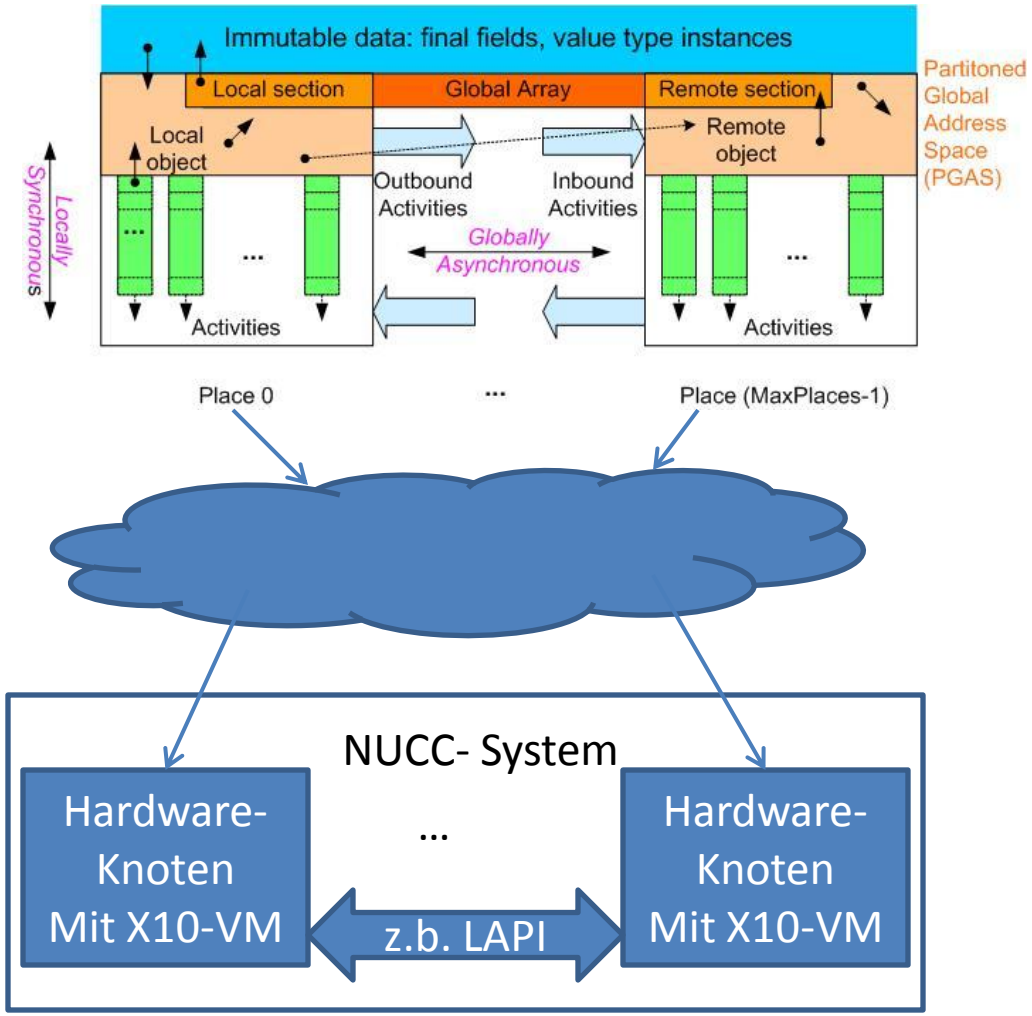
- Distribution
  - Ordnet jedem Element einer Menge einen Place zu
  - Können kombiniert werden
- Anwendung:
  - Erzeugen von vielen verteilten Tasks mit einem Befehl
  - Erzeugen/ Bearbeiten von verteilten Arrays
  - `x10.lang.dist`
  - `.factory.unique()`
  - `.union(dist d)`
  - `.intersection(d)`

Activities

- `async (P) {S}`
  - Neue Activity auf Place P
  - Führt den Block {S} aus
- Activities
  - Können nur auf lokale Variablen zugreifen
  - Können neue Activities erzeugen
  - Root-Activity wird von X10 gestartet (≈ main-Methode in Java)

- Die vier Activity- Operationen:
  - `async (P) {S}`
    - Führt Statement S aus
    - Nicht- Blockierend
  - `Finish (P) {S}`
    - Blockiert, bis alle Activities (transitiv) beendet sind
    - Sammelpunkt für Exceptions
  - `Future<int> f = Future (P) {E}`
    - Führt Ausdruck E aus
    - Nicht blockierend
  - `Force f`
    - Gibt das Ergebnis von E blockierend zurück
    - Wird in f gespeichert (keine wiederholte Auswertung)

Places sind virtuell!



Virtuelles Konzept

Deployment-Schicht

Hardware

Das waren die Grundlagen

1. Datentypen und Strukturen wie Java
2. **Places** -> Abbildung auf Rechenknoten
3. **Activities**
4. **Distributions** -> Ordnet jedem Punkt einer Menge einen Place zu

Arrays

- Verteilte Arrays werden mit Distributions erzeugt
- Bearbeitung mit speziellen Schleifen
  - Foreach
    - Erzeugt für jeden Point eine Activity auf dem lokalen Place
  - Ateach
    - Erzeugt für jeden Point eine Activity auf dem Place, auf den die Distribution verweist (wo die Daten liegen)

```
dist myDist = dist.factory.unique();
final int[.] a = new int[myDist] (point p) {return here.id;};
/* ateach (point p : a) {a[p] = here.id;} */
finish foreach (point p : a) {
    System.out.println(p + " Place " + here.id);
}
System.out.println();
ateach (point p : myDist) {
    System.out.println(a[p] + " Place " + here.id);
}
```

Ausgabe:

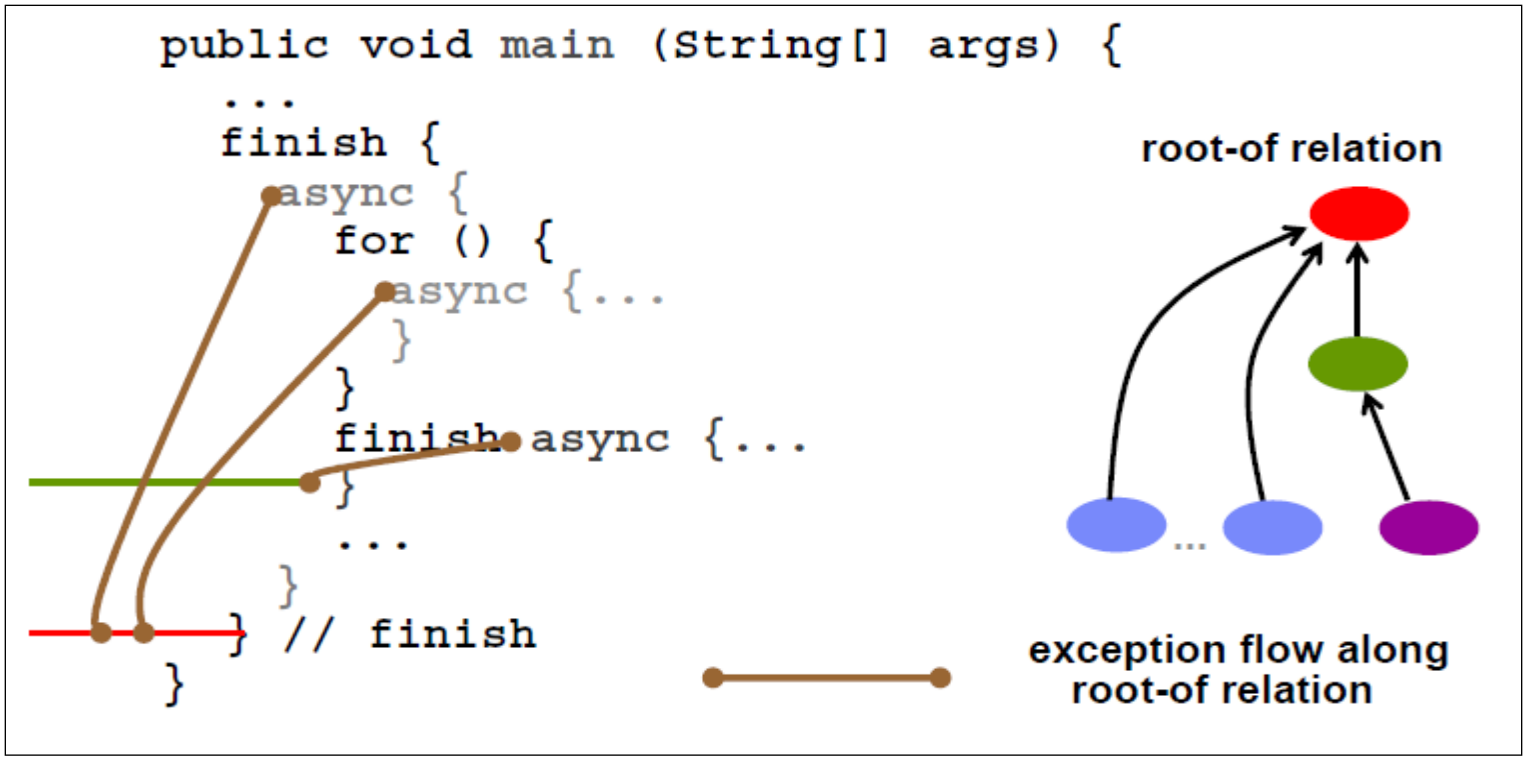
```
[1] Place 0
[0] Place 0
[2] Place 0
[3] Place 0

1 Place 1
0 Place 0
3 Place 3
2 Place 2
```

Rooted Exception Model (1/2)

- Problem:
  - X10 Activities werden nicht synchron beendet
  - Die Activity A kann beendet werden, bevor die von ihr mit async aufgerufene Activity B beendet ist
  - Wer fängt dann Exceptions, die von B geworfen werden?
- Lösung
  - *X10 Rooted Exception Model*

Rooted Exception Model (2/2)

- Finish-Aufrufe sind „Sammelpunkte“ für Exceptions
- Jede Activity hat eine „root“ Activity
- Ein X10-Programm wird mit `finish main(…)` gestartet



Atomic Blocks

Unconditional Atomic Blocks

- Eingeleitet mit Schlüsselwort `atomic{}`
- Nur „einfache“ Berechnungen
  - Nur lokaler Speicher
  - Keine blockierenden Aufrufe oder `async`
  - Alle Speicherzugriffe müssen statisch zu bestimmen sein
  - Exakter Algorithmus zur Konfliktauflösung steht noch nicht fest

Conditional Atomic Blocks

- (mindestens) ein Guard (nur Feldzugriff oder Konstante)
- Überprüfung der Guard und Ausführung des Blocks werden unteilbar durchgeführt
- Deadlocks möglich!

```
class Semaphore {
  private boolean taken;

  void p() {                          acquire semantics
      when (!taken)
         taken = true;
  }

  atomic void v() {                   release semantics
      taken = false;
  }
}
```

Clocks

```
async clocked (c1, c2,…) {…}
```

- Synchronisation von Activities
  - Activities können bei einer (oder mehreren) Clocks (x10.lang.clock) registriert sein
  - Registieren nur bei Erzeugung der Activity
  - Weiterschalten mit `c.next()` (blockierend) oder `c.resume()` (nicht- blockierend)
  - Aufruf von `next()` endet erst, wenn alle registrierten Activities weitergeschaltet haben
- Activities können sich auch abmelden (`c.drop()`)
- Ersetzen MPI-Barriers
- Flexibler als Barriers, die immer auf einer festen Untermenge der Prozessoren bestehen
- `Next()` schaltet **alle** Clocks, auf denen die Activity registriert ist
- Deadlock-Freiheit wird garantiert (mit Beweis!)

Deadlock- Freiheit (für Clocks)

- Beweisidee:
  - Annahme: ein Programm steckt fest (Deadlock)
  - Also haben alle Activities auf (verschiedenen) Clocks `next()` aufgerufen
  - Darum sind auch alle Clocks freigeschaltet
    - verwendet das next() alle Clocks einer Activity schaltet
  - Also können alle Activities weiterrechnen

→Kein Deadlock

Abkürzungen, implizite Statements

```
await (c) => when (c) {;}
at (p) e  => (future (p) e).force;
```

- Erlauben kürzere Programme
- Bessere Lesbarkeit/Wartbarkeit ??
- Das hängt von der Menge der neuen Schlüsselwörter ab
- Automatisches Einfügen von `at(p)e` bei Zugriff auf Objekte auf anderen Places?

Abschluss

- Sicherheit in X10
- Existierende Implementierungen, Versionen

Sicherheit in X10

- Man kann in X10 Fehler erzeugen
  - Deadlocks, Null Pointer Exception, …
- Wenn man unbedingt will
  - Beispiel: Nullable <T> hat alle Werte von T und null

- X10 versucht nicht dem Programmierer zu verbieten Fehler zu machen
- X10 versucht zu helfen die Fehler zu vermeiden durch
  - Übersichtliche Sprache
  - Vorgaben, welche Konstrukte wie zu benutzen sind
  - Werkzeugunterstützung für Entwicklung und Debugging

Implementierungen

- Version 1.5
  - Im wesentlichen Java-Syntax, Diese Präsentation
  - X10DT-Plugin für Eclipse
- Version 1.73 (24. März 09)
  - Scala-Syntax eingeführt
- Version 1.75 (26. Juni 09)
  - Implementierungen für Java, C, C++ für Linux
  - X10DT Plugin aktualisiert

Überblick

Ziele der Sprache

Motivation

Places

Distributions

Activities

Arrays

Rooted Exception Model

Clocks

Atomic Blocks

Sicherheit

Implementierungen

„Diese Frage ist zu gut, um sie mit einer Antwort zu verderben.“
Robert Koch

„Über Fragen, die ich nicht beantworten kann, zerbreche ich mir nicht den Kopf.“
Konrad Zuse