Verteilte und Parallele Programmierung

Mit Virtuellen Maschinen

Prof. Dr. Stefan Bosse

Universität Koblenz - FB Informatik - FG Praktiksche Informatik

Überblick

Topics

1. Formales: Taxononmien und Metriken von parallelen sowie verteilten Systemen
2. Kommunikationsmodelle:
   - Speicherbasierte Kommunikation (geteilter Speicher)
   - Nachrichtenbasierte Kommunikation (Netzwerke)
3. Ablaufmodelle und Architeturen:
   - Sequenzieller Prozess und Zustandsautomaten
   - Parallele Prozesse, Threads, Koroutinen
   - Zelluläre Automaten
   - Prozessalgebra
   - Asynchrone Ein- und Ausgabe (Vordergrund-Hintergrundsystem)
4. Synchronisation, Interprozesskommunikation
5. Virtuelle Maschinen
6. **Programmierung mit Lua**

Programmierung

Die Lua Programmiersprache

- Dynamisch typisierte Skriptsprache
- Einfach zu erlernen, funktional, minimalistisch, universell in der Anwendung
- Sequenzielle Ausführung, aber Verschachtelung mit Koroutinen (jedoch nicht präemptiv)
- Serialisierbar (Daten und Funktionen)
- Es gibt verschiedene Virtuelle Maschinen
  - Lua-C
  - Lua-JS (Fengari)
  - LuaJit
  - ...
- VM ist leichtgewichtig, Multihtreading einfach möglich, geringe Startzeit
- Einfache Erweiterbarkeit (C-API, JS-API)

Programmierung

- Diverse Softwaretools
  - lua (Core)
  - lvm (LuaJit)
  - paluvm (Lua-C)
  - VNetOS (Lua-JS)
  - LuaWeb (Lua-JS)
- Diverse (eingebaute) Bibliotheken
  - parallel → Parallel Map&Reduce
  - csp → Communicating Sequential Processes
  - rpc → Remote Procedure Call
  - luv → Asynchronous IO
  - caluap → Parallel Cellular Automata
  - tuple → generative musterbasierte Kommunikation

Architekturen und Implementierung

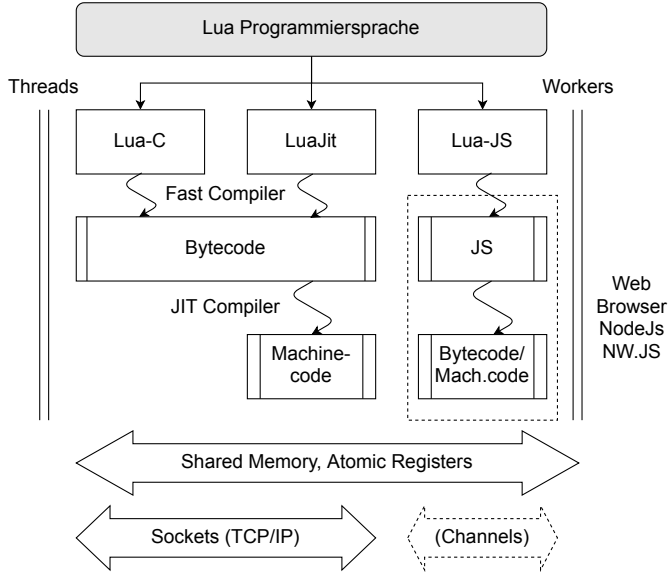
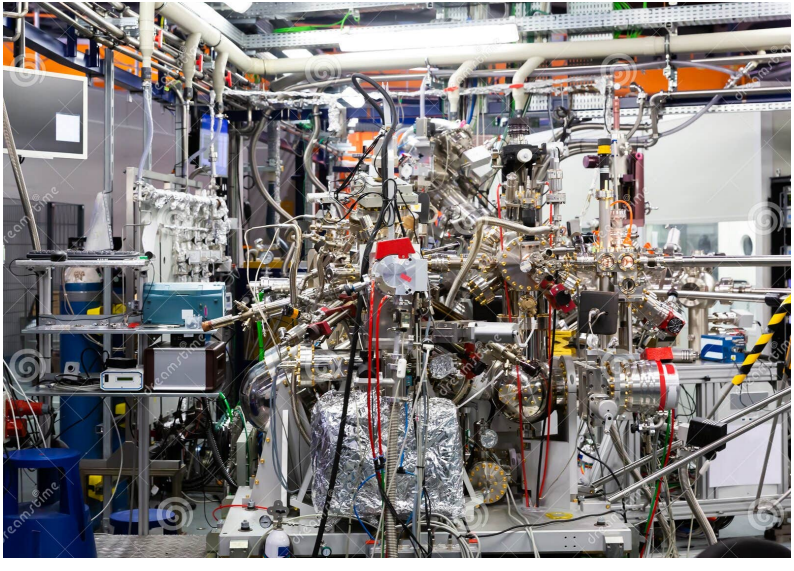


Abb. 1. Verschiedene Lua Implementierungen die in diesem Kurs zum Einsatz kommen

Lua als Experimentallabor



Schwerpunkte in diesem Kurs

Parallelisierung primär auf Kontrollpfadebene!

- Grundlagen von parallelen und verteilten Systemen
- Konzepte der parallelen und verteilten Programmierung
- Praktische Relevanz und Anwendung
- Plattformen und Technologien, Virtualisierung
- Netzwerke, Nachrichten, und Protokolle

Parallelisierung sekundär auf Datenpfadebene!

- Vektoroperationen

Begleitet von Übungen um obige Techniken konkret anzuwenden

Vorlesung

2 SWS mit Grundlagen und Live Programming
→ On-site und On-line
→ Tutorial Videos

Übung

2 SWS mit Programmierung und angewandter Vertiefung → Digitale Notebooks mit Online Hilfe Funktion und Einreichungssystem

Voraussetzungen

Grundlegende Programmierfähigkeiten, Grundkenntnisse in Rechnerarchitektur und Netzwerken

Materialien

1. Die Vorlesungsinhalte (Skript, Folien) werden auf **http://edu-9.de** unter der Rubrik Lehre zusammengestellt und angeboten. Achtung: Kürzel **vpp4k** beachten!

2. Weitere Materialien (Tutorials, Übungen, Software) werden ebenfalls auf **http://edu-9.de** bereitgestellt

3. Es gibt einige Videos und sind über **http://edu-9.de** erreichbar

4. Es gibt eine Mailing Liste (ZfN) über die aktuelle Informationen versendet werden (Updates, Ankündigungen). Eine Anmeldung ist nicht erforderlich.

Leistungen

Folgende Möglichkeiten einer Prüfungsleistung stehen zur Auswahl:

1. Mündliche Prüfung
2. Schriftliche Ausarbeitung zu einer Fragestellung zu dem Thema (Review/Survey)
3. Die Bearbeitung einer experimentellen Arbeit (Lua) mit kleiner schriftlicher Arbeit (Dokumentation)

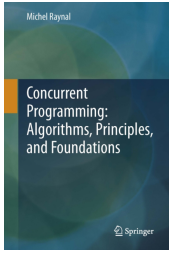
Literatur

Vorlesungsskript und Folien

Die Inhalte der Vorlesung werden sukzessive bereitgestellt (HTML, PDF, EPUB Format)

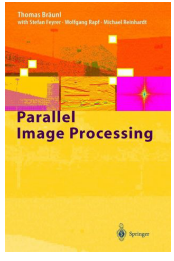
Concurrent Programming: Algorithms, Principles, and Foundations

Michel Raynal, Springer 2013, ISBN 978-3-642-320626-2



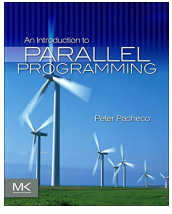
Parallel Image Processing

T. Bräunl, Springer Berlin, 2001



Literatur

An introduction to Parallel Programming
P. S. Pacheco, Elsevier, MK, 2011.



Lua Scripting Language
Tutorial Points, K. K. Panigrahi, 2016.

Software

Lua-C

- Reine C Implementierung der Lua VM (Bytecode) inklusive REPL und Compiler
- Weit verbreitet
- Einfach zu integrieren (Bibliothekskonzept)
- Mittlere Performanz im Vergleich zu Google V8/nodejs, höhere Performanz im Vergleich zu Python

PaLu VM

- Aufbauend auf Lua-C (5.4)
- Parallele Ausführung in Threads möglich durch Kapselung (keine globalen Variablen)
- Eingebaute Erweiterungen für die parallele und verteilte Programmierung (csp.lua, parallel.lua, rpc.lua)
- Asynchrone Ein- und Ausgabe mittels luv
- Mittlere Performanz im Vergleich zu Google V8/nodejs, höhere Performanz im Vergleich zu Python

Software

LuaJit

- LuaJit Virtual Machine ⇒ Kernelkomponente von Michael Pall (luajit.org)
- Integrierte Bytecode und native code just-in-time Compiler
- Parallele Ausführung in Threads möglich durch Kapselung (keine globalen Variablen)
- Mittlere Performanz im Vergleich zu Google V8/nodejs, hohe Performanz im Vergleich zu Python

Fengari VM

- Direkte JavaScript Implementierung von Lua (C) in JavaScript
- Kann mit jedem Web Browser und mit Node Webkit (nw.js) ausgeführt werden
- Parallele Ausführung in Web Workern möglich, ggfs. mit Shared Memory, aber limitierte Interprozesskommunikation
- Niedrige Performanz (aber vergleichbar mit Python VM)

Software

lvm

edu-9.de

- LuaJit VM mit Multithreading
- Ausführung von der Kommandozeile
- Wird auch für Live Programming und mit Digitalen Notebooks genutzt
- Einsatz auf verschiedenen Hostplattformen : PC, Smartphone, Embedded PC, Server, ..
- Einsatz auf Betriebssystemen: Windows, Linux, Solaris, MacOS, Android

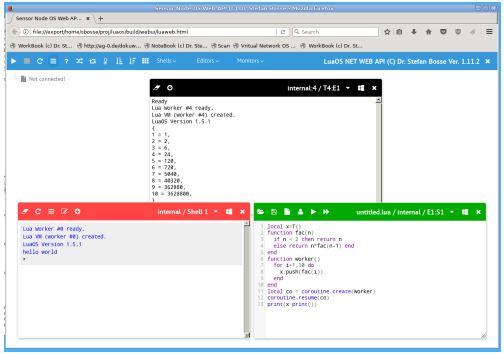
```
> lvm parfib.lua
Thread [fe5af458:4] released
Thread [fe5afa00:5] released
{
  1 = 9227465,
  2 = 24157817,
  3 = 63245986,
  4 = 165580141,
  5 = 14930352,
  6 = 39088169,
  7 = 102334155,
  8 = 267914296,
}
686478381
Time 6235 ms
```

Ausführung Kommandozeile

Software (Optional)

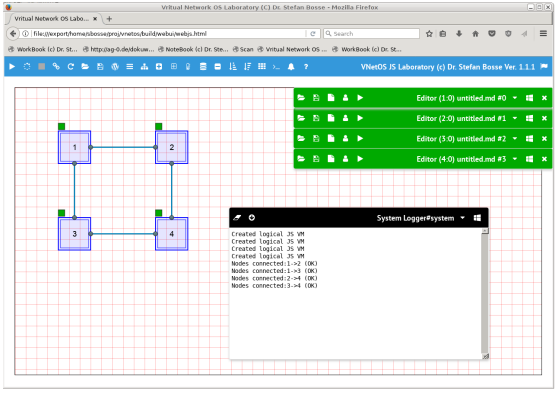
LuaOS Web IDE

- Beinhaltet ein kleines LuaOS und die Fengari Lua VM
- Multithreading durch Verwendung von Web Workern
- Schnittstelle zu externen *lvm* Instanzen via weblvm REST API



VNetOS Web IDE

- Beinhaltet das LuaOS und die Fengari Lua VM
- Aufbau von virtuellen Netzwerken
- Einbindung von externen *lvm* Instanzen möglich



Software (Bilbiotheken)

parallel.lua

(In *lvm* bereits integriert)

edu-9.de

- Programmierung in Lua
- Bibliothek für parallele und verteilte Systeme
- Einfach zu Erlernen
- Benötigt die LuaJit VM *lvm*

```
require('parallel')
local function worker (id,set)
  local results = T{}
  for i = 1,#set do
    results:push(fib(set[i]))
  end
  return results
local data = {34,35,36,37,38,39,40,41}
local p = Parallel:new(data,options)
p:time():
  map(worker):
  reduce(sum):
  apply(print):
  time()
```

Programmcode

Software (Bilbiotheken)

csp.lua

(In *lvm* bereits integriert)

edu-9.de

- Programmierung in Lua
- Bibliothek für Konkurrierende Parallele Programmierung mit Shared Memory
- API entspricht erweiterten CCSP Modell
- Einfach zu Erlernen
- Benötigt die Lua VM *lvm* (integriert)

```
require('Csp')
Par({
  function (pid)
    local x=ch1.read();
    ch2.write(fib(x))
  end,
  function (pid)
    Seq({
      function () .. end,
      function () .. end
    })
  end
  function (pid)  Seq({..}) end
},{ ch1:Channel(1), ch2:Channel(1))
```

Programmcode

Software

rpc.lua

edu-9.de

- Programmierung in Lua
- Bibliothek für Remote Procedure Call Kommunikation
- Aufbau von verteilten Systemen
- Einfach zu Erlernen
- Benötigt die Lua VM *lvm*

```
local rpc = Rpc({verbose=2})
rpc:getreq('127.0.0.1',12345,function (req)
  print(req)
  return {
   result=req.x+req.y,
   stat='OK'
  }
end)
local stat,reply =
  rpc:trans('127.0.0.1',12345,
            {x=1,y=2})
print(reply)
```

Programmcode

Konzept

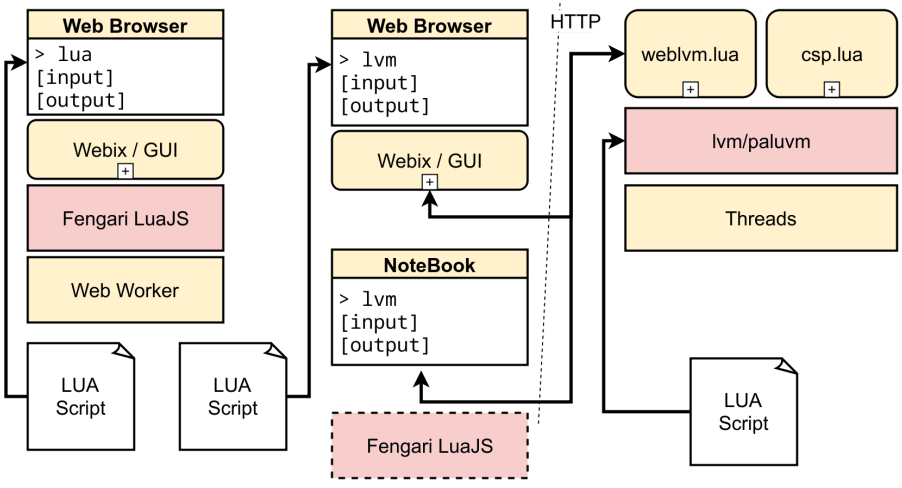


Abb. 2. Verschiedene Konzepte Lua VMs in diesem Kurs einzusetzen und zu integrieren

Ziele

1. Verständnis der Grundprinzipien und Architekturen verteilter (VS) und paralleler Systeme (PS) und Fähigkeit zum Transfer auf technische Systeme;
2. Verständnis und Fähigkeit der programmatischen Anwendung von Synchronisation und Kommunikation in VS und PS;
3. Verständnis der Probleme und dem Betrieb von parallelen Systemen im Vergleich zu sequenziellen Systemen bezüglich Effizienz, Blockierung, Skalierung, Ressourcenbedarf sowie Fehler wie Verklemmung (Deadlocks);
4. Praktische Kenntnisse der Programmierung von PS und VS anhand von Programmierübungen mit Lua und (p)lvm (LuaJit);
5. Erkenntnisse der Grenzen und Möglichkeiten der Parallelisierung und Verteilung und die Fähigkeit effiziente Systeme zu entwickeln → Virtuelle Maschinen!
6. Vorbereitung für Methoden und zukunftige Trends im Cloud Computing und Internet der Dinge (praktisch: Raspberry PI).

Inhalte

A. Parallele und Verteilte Systeme, Zelluläre Automaten
B. Sequenzielle und Parallele Datenverarbeitung
C. Funktionale, Sequenzielle, und Parallele Komposition
D. Prozessmodelle, Kontroll- und Datenpfade, Petri-Netze
E. Kommunizierende Prozesse, Synchronisation und Kommunikation (Primitiven)
F. Virtuelle Maschinen: Architekturen, Programmverarbeitung, Speichermanagement
G. Praktische Parallele Programmierung mit *Lua* und *csp.lua/parallel.lua*
H. Praktische Verteilte Programmierung mit *Lua* und *rpc.lua/parallel.lua*
I. Verteilung und Parallelisierung: Methoden und Algorithmen
J. Netzwerke und Nachrichrenaustausch
K. Plattformen und Architekturen: Multiprozessor, Cluster, Cloud, IoT, GPU
L. Gruppenkommunikation, Konsens, Distributed Shared Memory...

Virtuelle Maschinen

- Die Verwendung von virtuellen Maschinen nimmt in der Datenverarbeitung immer mehr zu (**Skriptverarbeitung und Abstraktion**)
- Ein Schwerpunkt liegt in der Parallelisierung in und mit virtuellen Maschinen
- Vor allem Parallelität auf Kontrollpfadebene und kommunizierende Systeme sollen betrachtet werden!

```
local s = {Semaphore(1),Semaphore(1),Semaphore(1)}
local b = Barrier(3)
Par({
  function () b:await(); for i = 1,10 do
    s[1]:down(); s[2].down(); eat(); s[2]:up(); s[1]:up(); think()
  end end,
  function () b:await(); for i = 1,10 do
    s[2]:down(); s[3]:down(); eat(); s[3]:up(); s[2]:up(); think()
  end end,
  function () b:await(); for i = 1,10 do
    s[3]:down(); s[1]:down(); eat(); s[1]:up(); s[3]:up(); think()
  end end
})
print('Done.')
```

Virtuelle Maschinen und Interpreter

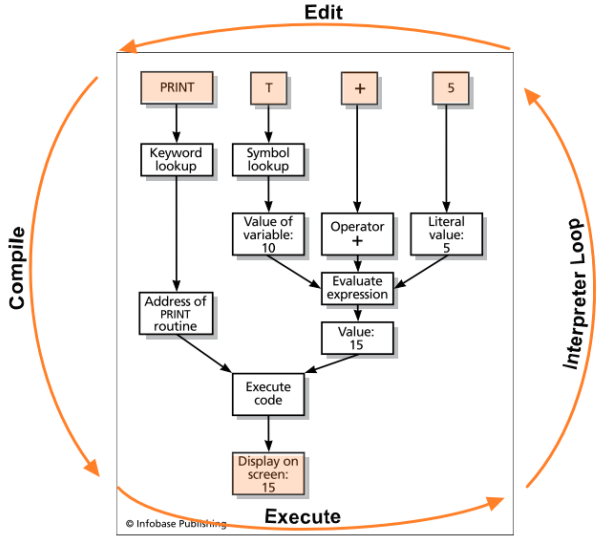


Abb. 3. Interpreter Zyklus: Editieren → Übersetzen → Ausführen

Just-in-Time Compiler

Interpreter können im wesentlichen auf drei Arten (Architekturklassen) implementiert werden:

1. Direkte Ausführung des Quelltextes (die Nutzereingabe und bereits geschriebene Skripte) (*Parse* → *Execute*)

2. Virtuelle Maschine und Übersetzung des Quelltextes in eine Zwischenrepräsentation die von einer virtuellen Maschine ausgeführt werden kann → Bytecode

3. Virtuelle Maschine mit Bytecode Übersetzung, Ausführung des Bytecodes, und ausgewählter Übersetzung des Bytecodes in nativen Maschinencode → JIT Compiler!

> Virtuelle Maschine: Virtualisierung ≡ Abstraktion und Automatisches Speichermanagement

Beispiele

- **BASIC**: Klasse 1
- **Python**: Klasse 2 (Bytecode)
- **JavaScript**: Klasse 2 (Spidermonkey, WEB Browser) und Klasse 3 (Google Chrome/V8, nodejs)
- **OCaML**: Klasse 2 (und native Codeerzeugung mit Compiler)
- **Lua**: Klasse 2 (Lua) und Klasse 3 (LuaJit/lvm)

Parallelisierung der Datenverarbeitung von VM schwierig, Verteilung hingegen ist prinzipiell moglich.

Abstraktion

Was kann durch eine VM abstrahiert oder virtualisert werden?

Programmiersprachen

Welche Programmiersprachen werden häufig verwendet?

Welche parallelen Programmiersprachen sind bekannt?

Parallele und Verteilte Programmierung mit Lua

- In diesem Kurs soll die Programmierung mit der Skriptsprache Lua erfolgen und mit der virtuellen Maschine *lvm* ausgeführt werden
- Der Lua Quelltext wird durch einen Übersetzer in Bytecode übersetzt der von *lvm*/*paluvm* direkt ausgeführt wird.
  - Besonderheit: Der Bytecode wird direkt während des Parservorgangs erzeugt (kein AST-IR)
- Die LuaJit (*lvm*) oder PaLu VM (*paluvm*) unterstützen parallele Datenverarbeitung und das Konzept der Prozessblockierung
  - Prozesse
  - Threads
  - Koroutinen (Fibers)

- Kontrollpfadparallelität benötigt i.A. Kommunikation und das Konzept der Blockierung des Prozessflusses!
- Formales Ausführungsmodell: **Communicating Sequential Processes** (CSP)

CSP ist eine von Tony Hoare an der Universität Oxford entwickelte Prozessalgebra zur Beschreibung von Interaktion zwischen kommunizierenden Prozessen

- **Programmfluss = Kontrollfluss + Datenfluss**
- Parallele und Verteilte Datenverarbeitung: *Übergang vom Shared Memory (SM) zum Distributed (Shared) Memory (DSM) Modell!*

Parallele und Verteilte Programmierung mit Lua