TECHNISCHE UNIVERSITÄT DRESDEN

Zentrum für Informationsdienste und Hochleistungsrechnen

Julia High Performance Programmiersprache für High Performance Computing?

Proseminar – Paul Gottschaldt

Zellescher Weg 12
Willers-Bau xxxx
Tel. +49 351 - 463 – xxxxx

Nöthnitzer Straße 46
Raum xxx
Tel. +49 351 - 463 – xxxxx

Vorname Name (vorname.name@tu-dresden.de)

ZIH
Zentrum für Informationsdienste und Hochleistungsrechnen

Julia – High Performance Programmiersprache?

„Forscher können sich nun auf die Lösung ihrer Probleme fokussieren und müssen sich nicht mehr mit dem Programmieren herumschlagen.“

- Viral Shah, CEO von Julia Computing

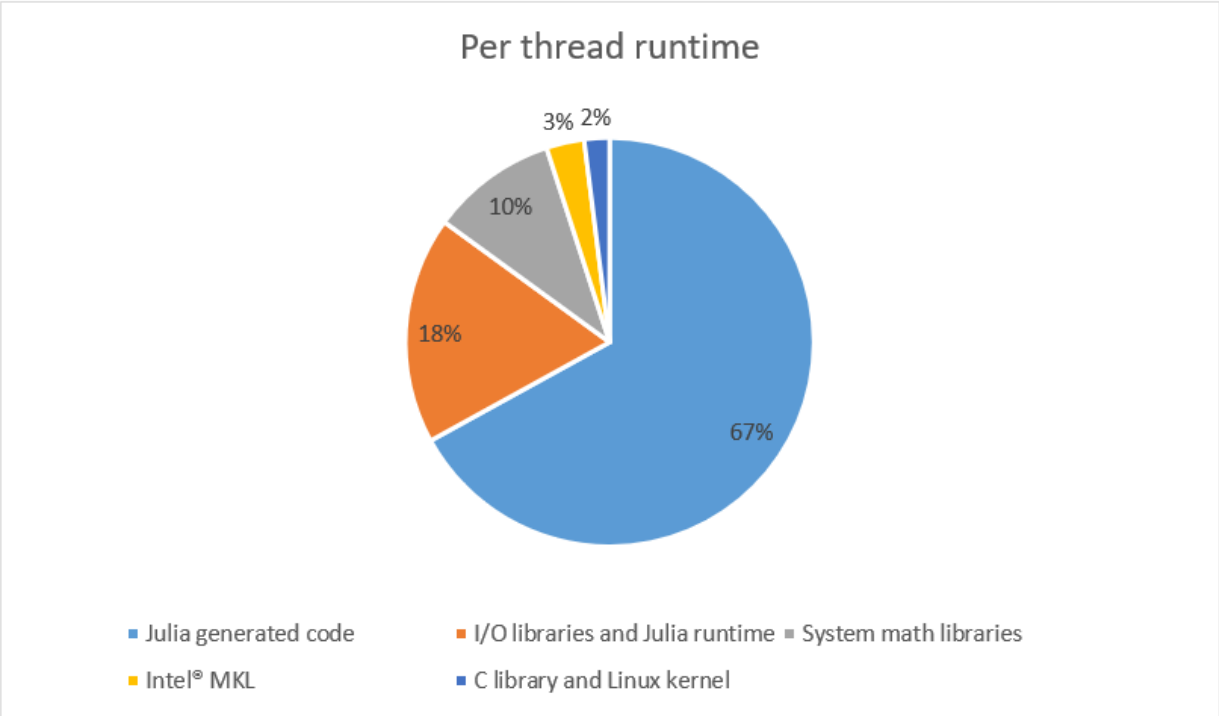


Figure 4: Per-thread runtime performance breakdown

https://www.nextplatform.com/2017/11/28/julia-language-delivers-petascale-hpc-performance

Julia High Performance - Gliederung

1. Grundlagen zu Julia
2. Parallele Programmierung in Julia
3. Bewertung der Performanz von Julia
4. Resümee

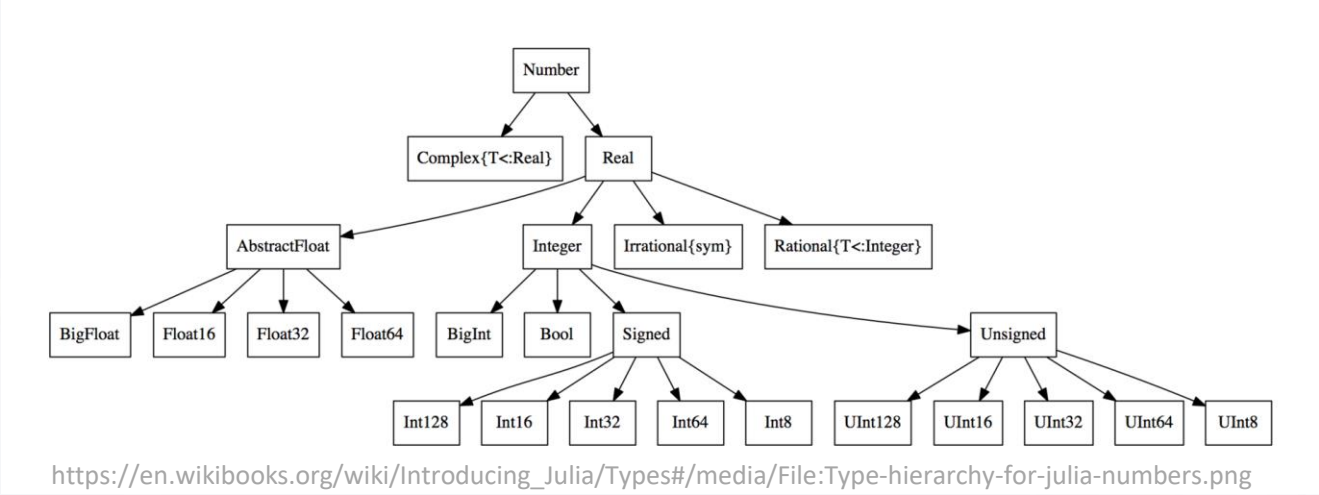
Julia High Performance – Grundlagen zu Julia

- Junge flexible höhere Programmiersprache
- Ansporn: Hohe Produktivität von Python mit C-Performanz kombinieren
- Syntax stark an MATLAB und Python orientiert
- Dynamisch aber stark typisiert
- Funktional, objektorientiert und imperativ
- Fast komplett in Julia geschrieben
- Packetmanager der an Git anlehnt

Julia High Performance – Grundlagen zu Julia

- Daten eines Objekts explizit von dessen Methoden getrennt
- Abstrakte und konkrete Datentypen bilden Typenbaum
- Funktionen generische Objekte und Methoden einzelne Implementierungen dieser für bestimmte Argumenttypen
  - meth(obj::Object,arg::Number) statt obj.meth(Number arg)

Vererbung bei der je nach Eingabetypen einer Funktion Verhalten weiterreicht, Struktur aber ignoriert



https://en.wikibooks.org/wiki/Introducing_Julia/Types#/media/File:Type-hierarchy-for-julia-numbers.png

Julia High Performance – Grundlagen zu Julia

- Metaprogrammierung - an Lisp angelehntes Feature
- Eigener Code als Datenstruktur in sich selbst repräsentiert
- Manipulierbar -> ermöglicht Code transformieren und generieren
- @Macros Funktionen, welche Ausdrücke als Argumente nehmen, diese manipulieren und Ausdruck zurück geben
- Werden ausgeführt, wenn der Code übersetzt wird

Julia High Performance – Grundlagen zu Julia

- vierstufiger Compilierungsprozess
  - Code_lowered, code_typed, Code_llvm und Code_native
- Zur Laufzeit nativer Maschinencode mittels LLVM JIT Compiler
- Maschinencode wird zwischengespeichert und erneut ausgeführt bei Aufruf der Funktion mit gleichen Argumenttypen
- Funktionen ccall() und pycall() lassen Code direkt in Julia einbinden
  - Werden dank LLVM zu gleichen Maschinencode wie normalerweise compiliert

Julia High Performance – Parallele Programmierung

- Eigene Implementierung einer Nachrichtenübertragung zur Kommunikation zwischen Prozessoren
- Jeder Prozessor eindeutige Kennzeichnung
- Master-Worker-Prinzip
- Weitere Prozessoren einbindbar
  - entweder beim Starten via julia -p n
  - Oder mit der Funktion addprocs(n), wobei Remoteprozessoren via SSH Verbunden sein müssen
- Workers() und nworkers() geben derzeitige Anzahl an eingebundenen Arbeiterprozessoren an
- Rmprocs(id) entfernt bestimmten Prozessor

Julia High Performance – Parallele Programmierung

- Zwei primitive Objekte (Remote-Referenz und Remote-Aufruf)
- Remote-Referenz:
  - Instanz
  - Kann von jedem Prozessor aus genutzt werden
  - Zeigt auf gespeichertes Objekt auf einem bestimmten Prozessor
  - Zwei Ausprägungen von Remote-Referenzen (Future und RemoteChannel)
- Remote-Aufruf
  - Antrag, bestimmte Funktion mit bestimmten Argumenten auf Prozessor x auszuführen
  - Rückgabewert eines Aufrufs, der sofort zurückgegeben wird und später via fetch() geholt werden kann
  - remotecall(f,id,args…) oder remotecall_fetch(f,id,args..) direkter Remoteaufruf

Julia High Performance – Parallele Programmierung

- Bietet auch zwei Macros für Remote-Aufruf
- @Spawnat(pid,expr)
- @spawn(expr)
  - Sucht selbstständig nach freiem Prozessor für Ausdruck expr
  - Verhält sich dabei intelligent (versucht Datentransport zu vermeiden)
- Funktionen und Pakete müssen allen Prozessoren bei Remote-Aufruf bekannt sein
- Globale Variablen nicht synchronisiert
- Lokale Variablen direkt auf rechnenden Prozessor definieren

Julia High Performance – Parallele Programmierung

- Zwei Möglichkeiten um Programme schnell und einfach zu parallelisieren
- @parallel
  - Parallele For–Schleife
  - Besonders gut für Probleme mit vielen unabhängigen Iterationen aber kleinen Aufgaben in diesen
  - Auch Reduktionsfunktion (tensor-rank-reducing)
  - nheads = @parallel (+) for i = 1:M;
    Int(rand(Bool));
    end
- Pmap()
  - Besonders gut für große Funktionen, die einzeln auf Elemente einer Menge angewandt werden müssen
  - pmap(i-> log(i), 1:M, batch_size)

Julia High Performance – Parallele Programmierung

- Shared Arrays
  - Jeder teilnehmende Prozess besitzt eigenen Zugang zu gesamten Feld
  - Für gemeinsames Rechnen geeignet (Kommunikationsaufwand verringt)
  - Sdata() -> gibt Array zurück
- Distributed Array
  - Jeder teilnehmende Prozess besitzt lokalen Zugang zu eigenem Teil (chunk) des gesamten Arrays
  - Besonders Geeignet für sehr große Felder, welche zu groß für einen Prozessor sind
  - Convert() oder localpart() -> gibt Array bzw den lokalen Teil als Array zurück
- Beide nutzen gemeinsam genutzten Speicher im System um gleiches Array über viele Prozessoren hinweg zu verlinken

Julia High Performance – Parallele Programmierung

- Nutzbarkeit einer Sprache hängt von ihren Hilfswerkzeugen ab
- BenchmarkTools
  - Paket, das umfangreiche Funktionen bietet um Laufzeit, Allokationen, Speicher im best/worst oder mittelwert
- Julias hauseigene Macros um erzeugten Code zu inspizieren in den jeweiligen Stufen des Compilers
  - @code_native, @code_typed, @code_llvm
  - @code_warntype hebt ungenaue Typspezifikationen farblich hervor, da diese für den Compiler deutlichen Zeitverlust bedeuten
- Profiling
  - Werkzeug, das ermittelt wieviel Zeit pro Zeile benötigt worden ist
- Linter
  - Hinweise für einfachere oder performantere Implementierungen

Julia High Performance – Bewertung der Performanz

- Performanz ist ganz toll!

Julia High Performance – Resümee

- Performanz ist ganz toll!

Pro	Contra