Concurrent Computing

Stand: 29.08.2016

Ostfalia Hochschule für ang. Wissenschaften				Fakultät Informatik	
Bachelor Studiengang: X Informatik; X Informatik i.P. X IT-Mangement X Wirtschaftsinformatik Master Studiengang: X Informatik					
Modul	---		Lehrveranstaltung	**Concurrent Computing**	
Semester	Dauer (Sem.)	Häufigkeit	Art	ECTS Punkte	Studentische Arbeitsbelastung
4./5. Sem.	1	1x pro Jahr wird angestrebt	Wahlpflicht X Qualifikation ☐ Überf. Komp	5	150h, davon ca. 40% Kontaktstudium ca. 60% Eigenstudium

Voraussetzung für die Teilnahme	Verwendbarkeit	Prüfungsform / Prüfungsdauer	Vorgesehene Lehr- und Lernmethoden / -formen	Verantwortlicher
Gute Programmierkenntnisse in Java	Keine Besonderheiten	50% Klausur/mündliche Prüfung 50% Erstellung und Dokumentation von Rechnerprogrammen	Vorlesung (2 SWS), Labor (2SWS)	Prof. I. Schiering

Kompetenzziele
Studierende • verstehen Begriffe zur Skalierung paralleler Implementierungen und können diese anwenden, • kennen Konzepte des Concurrent Computing für Shared Memory und Distributed Memory Architekturen, • realisieren Programme unter Verwendung der Konzepte
Lerninhalte
Die Studierenden lernen neben Grundlagen in einem allgemeinen Teil Ansätze für Shared Memory und Distributed Memory Architekturen kennen I. Allgemeiner Teil • Gründe für die Entwicklung von Multicore-CPUs • Flynn'sche Taxonomie sowie Klassifikation von MIMD-Systemen nach Johnson • Skalierung paralleler Implementierungen (Speedup) sowie parallele Effizienz • Grenzen der Skalierung unter Beachtung der Gesetze von Amdahl und Gustafson II. Beispiel für einen Shared Memory Ansatz ist die Java Concurrency API. • Lambda Expressions • Atomare Datenstrukturen • Concurrent Collections • Konzepte zur Synchronisierung (Semaphore, Monitore, Locks, etc.) und klassische Probleme (Wechselseitiger Ausschluss, Producer-Consumer, Reader Writer, Dining Philosphers) • Phasen-Modelle (Barrier, Phaser) • Verwaltung von Threads (ThreadPool, Fork-Join Pool) Alternativ wird eine funktionale Sprache (Clojure) verwendet: • Einführung in die funktionale Programmierung

• Datenstrukturen und Nutzung von Libraries
• Concurrency in Clojure: Atoms, Refs, Vars
• Library core.async

III. Beispiel für einen Distributed Memory Ansatz ist MPI:
• Ideen des Message Passing auf Basis der MPJ-Implementierung der mpiJava-Spezifikation
• Grundlegender Aufbau von MPJ-Anwendungen (Prozessbegriff, Kommunikatoren sowie das Versenden und Empfangen von Nachrichten)
• Fallstudie zum parallelen Sortieren sowie Bestimmung des Speedups auf Basis eines Modells
• Kollektive Kommunikationsfunktionen zur Synchronisation von Prozessen, Verteilen und Einsammeln von Ergebnissen mittels Broadcast, Scatter(v) und Gather(v), Zusammenführen von (Teil-)Ergebnissen mittels Reduce und Allreduce
• Verwendung von Wildcards beim Empfangen von Daten
• Implementierung eigener Reduce-Operationen sowie Erkennen von einzuhaltenden Rahmenbedingungen
• Erstellung eigener Kommunikatoren sowie Betrachtung möglicher Anwendungsgebiete
• Nicht-blockierendes Senden und Empfangen von Nachrichten

Alternativ wird eine funktionale Sprache (Erlang) verwendet:
• Einführung in die funktionale Programmierung
• Datenstrukturen und Nutzung von Libraries
• Concurrency in Erlang: Message Passing
• Fehlerbehandlung
• Concurrency Patterns

Literatur

Fernandez Javier, Java 7 Concurrency Cookbook (Quick Answers to Common Problems), Packt Publishing, 2012.
Daniel Higginbotham, Clojure for the Brave and True: Learn the Ultimate Language and Become a Better Programmer, No Starch Press, 2015. (http://www.braveclojure.com/)
Fred Hébert, Learn You Some Erlang for great good!, No Starch Press, 2013. (http://learnyousomeerlang.com/)
Marc Snir et al., MPI: The Complete Reference (Vol. 1: The MPI Core) – 2nd Edition, MIT Press, 1998.
William Gropp, Ewing Lusk, Anthony Skjellum, Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation) – 2nd Edition, 2000.
Ananth Grama et al., Introduction to Parallel Computing – 2nd Edition, Addison Wesley, 2003.