Parallele Programmierung

- Was ist Parallelisierung?
- Paradigmen der parallelen Programmierung
- Werkzeuge zur Parallelisierung
- Algorithmische Aspekte
- Beispiele

Parallele Programmierung
Die zehn wichtigsten Fragen

- Was ist Parallelisierung eigentlich?
- Wie ist die allgemeine Vorgehensweise?
- Welche Paradigmen der Parallelisierung gibt es?
- Für welche Algorithmen sind diese Paradigmen jeweils gut geeignet?
- Mit welchen Werkzeugen wird parallelisiert?
- Welche Probleme gibt es bei der Parallelisierung durch den Compiler?
- Welche Klassen von Algorithmen können unterschieden werden?
- Wie parallelisiert man einfache numerische Verfahren?
- Wie parallelisiert man Anwendungen im Gebiet der Strömungsmechanik?
- Wie parallelisiert man Suchbaumverfahren?

Was ist Parallelisierung?

Aufgabe

- Finde impliziten Parallelismus im Algorithmus und mache ihn explizit
- Mittel: Verteile Programme und Daten auf die Ressourcen des Systems
- Wer? Was?
  - Programmierer und/oder Compiler und/oder Laufzeitsystem

Was ist Parallelisierung...

- Verteilung erzeugt neue Last (*overhead*)
  - Minimale Zusatzlast wenn nicht verteilt
- Verteilung nutzt Ressourcen optimal
  - Optimale Leistung wenn vollständig verteilt

Zielvorgabe

Nutze alle Ressourcen und minimiere die Zusatzlast

Anforderungen

- Zusätzlich zur sequentiellen Software
  - Aufteilung des Programms in kleine Teile (*partitioning*)
  - Hinzufügen von Koordination und Kommunikation
  - Abbildung der Teile auf die Komponenten des Computers (*mapping*)
- Probleme
  - Fehlersuche (neue Fehlertypen)
  - Leistungsanalyse (Programmbeeinflussung)
  - Lastausgleich (für optimale Leistung)

Paradigmen der Parallelisierung

Code-Aufteilung (auch: Macro-Pipelining)

Verteile den Programmcode über die Knoten

- Unterschiedlicher Code auf jedem Knoten
- Daten variieren gemäß dem Fluß der Berechnung
- Koordinator: erster/letzter Prozeß

Paradigmen der Parallelisierung...

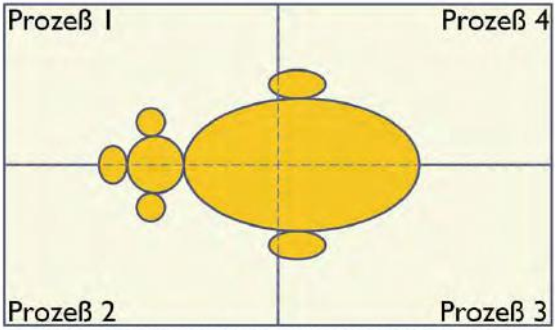
- **Vorteile** der Code-Aufteilung
  - Manchmal sehr einfach zu entwerfen
  - Passende Algorithmen existieren (z.B. FFT)
- **Nachteil** der Code-Aufteilung
  - Mehrere Quellcode-Dateien nötig
  - Schwierig an Zielmaschine anpaßbar
  - Komplexe Kommunikationsschemata
  - Schwierige Fehlersuche
  - Schwieriger Lastausgleich

Paradigmen der Parallelisierung...

Datenaufteilung

Verteile die Datenstrukturen auf die Knoten

- Identischer Code auf jedem Knoten
- Daten variieren von Knoten zu Knoten
- Durch ausgewählten Prozeß koordiniert



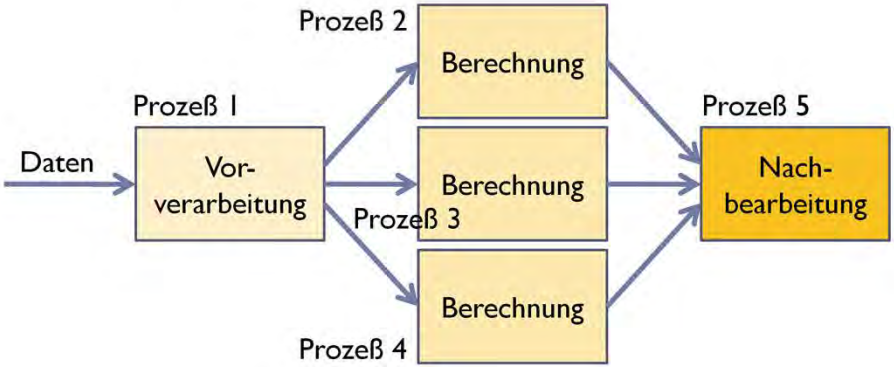
Paradigmen der Parallelisierung...

- **Vorteile** der Datenaufteilung
  - Leicht programmierbar: nur ein Quellcode
  - Einfach an Zielmaschinen anpaßbar
  - Sehr oft reguläre Datenaustauschschemata
  - Einfachere Fehlersuche
- **Nachteile** der Datenaufteilung
  - Manchmal dem Algorithmus nicht angemessen
- Datenaufteilung ist **Quasistandard**
  - Genannt **SPMD** (single program, multiple data)

Paradigmen der Parallelisierung...

Gemischte Code- und Datenaufteilung

- Dies ist unsere Zielvorstellung
  - Vereint Vorteile beider Ansätze
  - Benötigt Mehrprozeßbetrieb auf den Knoten



Werkzeuge zur Parallelisierung

- Automatisch parallelisierende Compiler
- Manuell parallelisierende Compiler
- Parallelisierung durch Laufzeitumgebung
- Parallele Spracherweiterungen
- Parallele Erweiterungen zu existierenden sequentiellen Sprachen
- Parallele Programmierbibliotheken für existierende sequentielle Sprachen

Automatisch parallelisierende Compiler

Parallelisierung auf Schleifenebene (Fortran)

- Compiler analysiert Datenabhängigkeiten
- Erkennt parallel ausführbare Schleifenindizes und verteilt sie
- Compiler-Pragmas notwendig (Steueranweisungen)
- Normalerweise schlechte Leistungsausbeute

Manuell parallelisierende Compiler

- Wichtiger neuer Ansatz: OpenMP
- Parallelisierung für Systeme mit gemeinsamem Speicher
- Übersetzung durch spezielle Kommentare kontrolliert
- Zusätzliche Verwendung von Bibliotheken

Parallelisierung durch Laufzeitumgebung

- Anwender übergibt dem System eine (höhere) Anzahl von Einzelaufträgen
- Laufzeitsystem schiebt mittels dynamischem Lastausgleich die Aufträge auf unbelastete Rechnerknoten
- Nur geeignet für grobgranulare Parallelisierung auf der Ebene der Aufträge

Parallele Sprachen und Spracherweiterungen

Ansatz: High Performance Fortran (HPF)

- Entworfen 1990+ durch ein Konsortium
- Schwindende Bedeutung für das Hochleistungsrechnen

Problem: Parallelisierungs-Qualität der Compiler

Parallele Programmierbibliotheken

Sind der Standard im Hochleistungsrechnen

- Konzept
  - Starten autonomer Prozesse (*spawning*)
  - Kooperation mittels Nachrichtenaustausch
- Beispiele
  - Parallel Virtual Machine (PVM) [veraltet]
  - Message Passing Interface (MPI)

Software/Hardware-Wechselspiel

Prinzipiell sind alle Programmierkonzepte auf allen Architekturen einsetzbar

In der Realität nutzen wir aus Effizienzgründen

- Bibliotheken zum Nachrichtenaustausch für Architekturen mit verteiltem Speicher
- Threads und automatische/manuelle Parallelisierung für Architekturen mit gemeinsamem Speicher

Algorithmische Aspekte

Zweigeteilte Welt des Programmierers

- Numerische Algorithmen
  - Grand Challenge-Algorithmen: Wettervorhersage, Klimabestimmung, Proteindesign usw.
- Nichtnumerische Algorithmen
  - Suchverfahren: Theorembeweiser, Spieleprogramme usw.
  - Datenbank-Anwendungen

Numerische Algorithmen

- Strömungsmechanik (*Computational fluid dynamics, CFD*), numerische Berechnungen, Optimierungen, Simulationen usw.
  - Iterative Algorithmen
  - Beenden aufgrund einer globalen Bedingung
  - Reguläre Datenstrukturen (Vektoren, Felder)
  - Reguläre Kommunikationsstrukturen
  - Statische Prozeßstruktur

Nichtnumerische Algorithmen

- Datenbankanwendungen, künstliche Intelligenz
  - Suchbaumverfahren
  - Irreguläre Kommunikationsstrukturen
  - Irreguläre Datenstrukturen (dynamisch, *garbage collection*)
  - Dynamische Prozeß/Thread-Struktur

Eine erste Zusammenfassung

- Paradigmen der Parallelisierung
  - Datenaufteilung, Code-Aufteilung
- Werkzeuge zur parallelen Programmierung
  - Compiler und Bibliotheken
  - Das wichtigste: natürliche Intelligenz
- Zweigeteilte Welt
  - Numerische / nichtnumerische Algorithmen

Beispiele von Parallelisierungen

Drei Beispiele

- Numerische Anwendung
  - Diskussion bzgl. der Aufteilung und der Speicherarchitektur
- Strömungsmechanik (CFD)
  - Diskussion bzgl. der zu verteilenden Objekte
- Suchbaumverfahren
  - Diskussion bzgl. der Speicherarchitektur

Alle Beispiele manuell parallelisiert

Beispiel 1: Numerik (1/11)

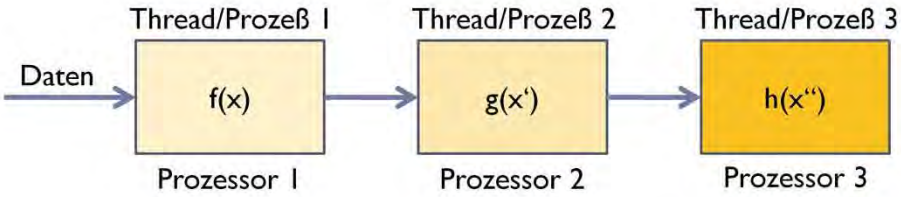
- Drei Funktionen f(), g() und h()
- Wende Funktionen auf eine Menge von Werten an und berechne Ergebnis h(g(f(x)))

- Wir betrachten vier Fälle:
  - Code-Aufteilung / Datenaufteilung
  - Gemeinsamer Speicher / verteilter Speicher

Beispiel 1: Numerik (2/11)

Code-Aufteilung

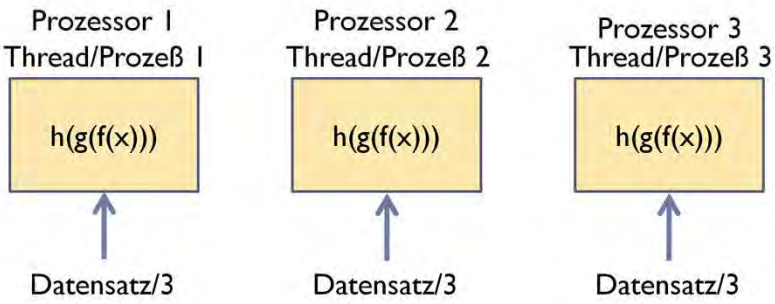
- Verteile drei Funktionen auf drei Knoten
- Arbeitet im Makro-Pipelining-Modus
- Wertemenge ist Eingabedatensatz
- Nur drei Prozessoren sinnvoll nutzbar



Beispiel 1: Numerik (3/11)

Daten-Aufteilung

- Repliziere die Funktionen auf den Knoten
- Verteile die Wertemenge auf die Knoten



Beispiel 1: Numerik (4/11)

Code-Aufteilung mit gemeinsamem Speicher

- Basis-Implementierung
  - Werte in Vektor gespeichert
  - Drei Threads, jeder auf eigenem Prozessor
  - Jeder Thread berechnet eine Funktion f, g, h
  - Vektoreinträge werden durch Berechnungsergebnisse ersetzt
  - Variable kennzeichnet den aktiven Thread
- Nachteil
  - Dies ist ***kein*** paralleles Programm (offensichtlich)!

Beispiel 1: Numerik (5/11)

- Verbesserung der Basis-Implementierung
  - Zähler für Thread [i], der seine Position anzeigt
  - Thread [i] darf bis zum Zählerstand von Thread [i-1] -1 vorrücken
- Vorteil
  - Gute parallele Implementierung
- Nachteil
  - Schlechtes Koordinations/Berechnungs-Verhältnis: zu häufige Inspektion der Zählervariablen

Beispiel 1: Numerik (6/11)

- Zweite Verbesserung
  - Erhöhe Granularität der Berechnung
  - Erhöhe Zähler jeweils um 1000, nicht um 1
- Vorteil
  - Gute parallele Implementierung
  - Besseres Koordinations/Berechnungs-Verhältnis
- Nachteil
  - Längere Phasen zum Füllen und Entleeren der Pipeline

Beispiel 1: Numerik (7/11)

Code-Aufteilung mit verteiltem Speicher

- Basis-Implementierung
  - Werte in Vektor abgespeichert
  - Drei Prozesse, jeder läuft auf eigenem Prozessor
  - Jeder Prozeß berechnet eine der Funktionen f, g, h
  - Prozeß [i] berechnet alle Zwischenergebnisse und sendet Vektor an Prozeß [i+1]

- Nachteil
  - Dies ist wieder ***kein*** paralleles Programm!

Beispiel 1: Numerik (8/11)

- Verbesserung der Basis-Implementierung
  - Prozeß [i] sendet berechnete Werte sofort zu Prozeß [i+1]
- Vorteil
  - Gute parallele Implementierung
- Nachteil
  - Schlechtes Kommunikations/Berechnungs-Verhältnis: häufiges Senden von Werten

Beispiel 1: Numerik (9/11)

- Zweite Verbesserung
  - Erhöhe die Granularität
  - Sende Werte in Blöcken zu 1000
- Vorteil
  - Gute parallele Implementierung
  - Besseres Kommunikations/Berechnungs-Verhältnis
- Nachteil
  - Längere Phase zum Füllen und Leeren der Pipeline

Beispiel 1: Numerik (10/11)

Datenaufteilung mit gemeinsamem Speicher

- Basis-Implementierung
  - Werte auf drei Blöcke aufgeteilt
  - Drei Threads berechnen je h(g(f(x))) pro Block
  - Eine Variable pro Block signalisiert das Ende der Berechnung
  - Ausgewählter Thread organisiert Ausgabe der Ergebnisse
- Vorteile
  - Gute parallele Implementierung
  - Keine Zugriffskonflikte bei den einzelnen Werten
- Nachteil (gering)
  - Potentieller Zugriffskonflikt auf Binärcode der Threads

Beispiel 1: Numerik (11/11)

Datenaufteilung mit verteiltem Speicher

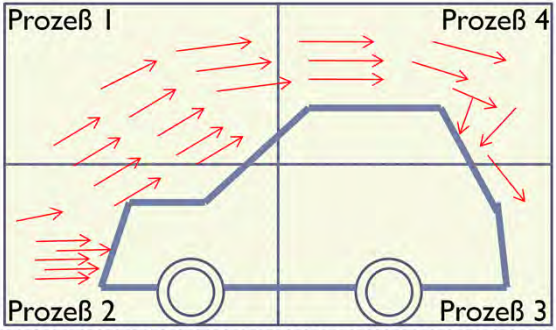
- Basis-Implementierung
  - Werte sind auf die Knoten verteilt
  - Drei Prozesse auf drei Knoten berechnen h(g(f(x)))
  - Ergebnisse werden einzeln an Prozeß 0 gesendet
- Vorteile
  - Gute parallele Implementierung
  - Gutes Kommunikations/Berechnungs-Verhältnis
- Nachteil
  - Programmierung der Datenverteilung

Beispiel 2: Strömungsmechanik (1/3)

(Das Beispiel illustriert die Probleme in der Praxis)

Simulation eines Windkanals

- Iterative Berechnung mit Zeitschritt t
  - Mikroskopischer Ansatz: Berechne Teilchen
    Molekulardynamik vs. Monte Carlo
  - Makroskopischer Ansatz: Berechne Verteilung von Druck, Temperatur usw.



Beispiel 2: Strömungsmechanik (2/3)

Mikroskopischer Ansatz:

Erste Variante: Verteile Teilchen

- Jeder Prozeß (Prozessor) verwaltet und berechnet einen Teil der Teilchen
- Nachteil
  - Physikalisch benachbarte Teilchen nur schwer bestimmbar
- Vorteil
  - Gleichverteilung der Anzahl der Teilchen ergibt meist guten Lastausgleich

Beispiel 2: Strömungsmechanik (3/3)

Mikroskopischer Ansatz:

Zweite Variante: Verteile Volumenanteile

- Jeder Prozessor berechnet einen Teil des Volumens
- Nachteil
  - Wechselnde Anzahl von Teilchen führt meist zu schlechter Lastbalance
- Vorteil
  - Benachbarte Teilchen leicht bestimmbar

Beispiel 2a: Klimasimulation

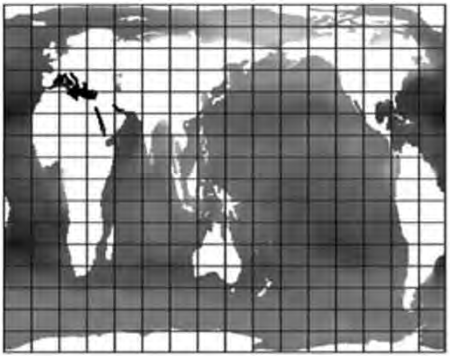
Parallelisierung und paralleler Ablauf durch folgende Schritte

1. Zerlegung des Gebiets in eine sehr große Anzahl von Zellen oder Gitterpunkten (Partitioning)
   - Komplexes Problem der Klimasimulation, der numerischen Modellierung und der Informatik,
2. Bestimmung der zu verwendenden Prozessoranzahl (zum Zeitpunkt des Jobversendens)
3. Zusammenfassung von Zellen oder Gitterpunkten zu Gruppen, damit es genau zur Prozessoranzahl passt (Mapping)
   - Komplexes Problem der Informatik

Beispiel 2a: Klimasimulation...

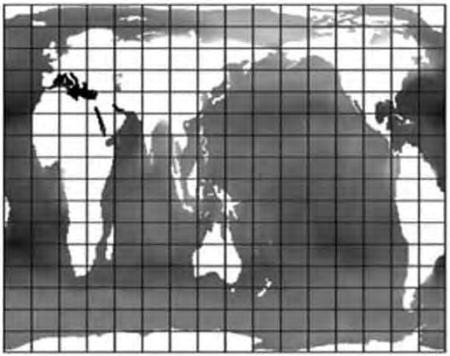
Beispiel:

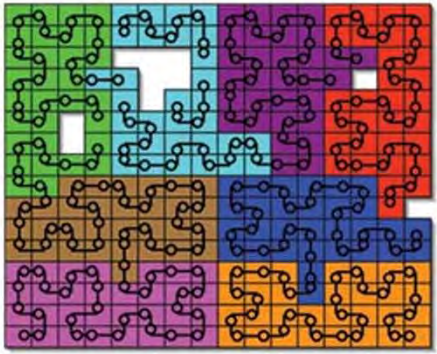
- Gittererstellung für ein zu modellierendes Gebiet eines Ozeanmodells



Beispiel 2a: Klimasimulation...

- Verwendung von 8 Prozessoren
- Aufsammeln der Zellen im Gitter durch raumfüllende Kurven





Quelle: http://www.cisl.ucar.edu/research/2006/numerical.jsp

Beispiel 2a: Icosaeder-Gitter (20-Flächer)

ICON (Icosahedral non-hydrostatic) general circulation model

- Max-Planck-Institut für Meteorologie Hamburg
- Deutscher Wetterdienst Offenbach

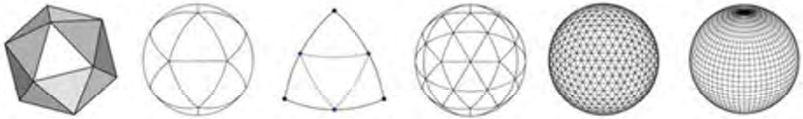


Figure 1 describes the construction of the grids. The icosahedron (a) is projected onto a sphere (b).
The edges of each triangle are bisected into equal halves or more generally into n equal sections. The new edge points are connected by great circle arcs to yield 4 (or more generally n^2) spherical triangles within the original triangle (c).
After refining the first spherical icosahedrons (d), this method can be extended in the same way. After two more steps the grid is reached (e). Such grids avoid polar singularities of latitude-longitude grids and allow a high uniformity in resolution over the whole sphere (f).

Beispiel 2a: Icosaeder-Gitter (20-Flächer)

Icosaeder-Gitter mit regionalen Verfeinerungen

- blau: global
- grün: nördl. Hemisphäre
- rot: Europa

Beispiel: Konzepte der Parallelisierung

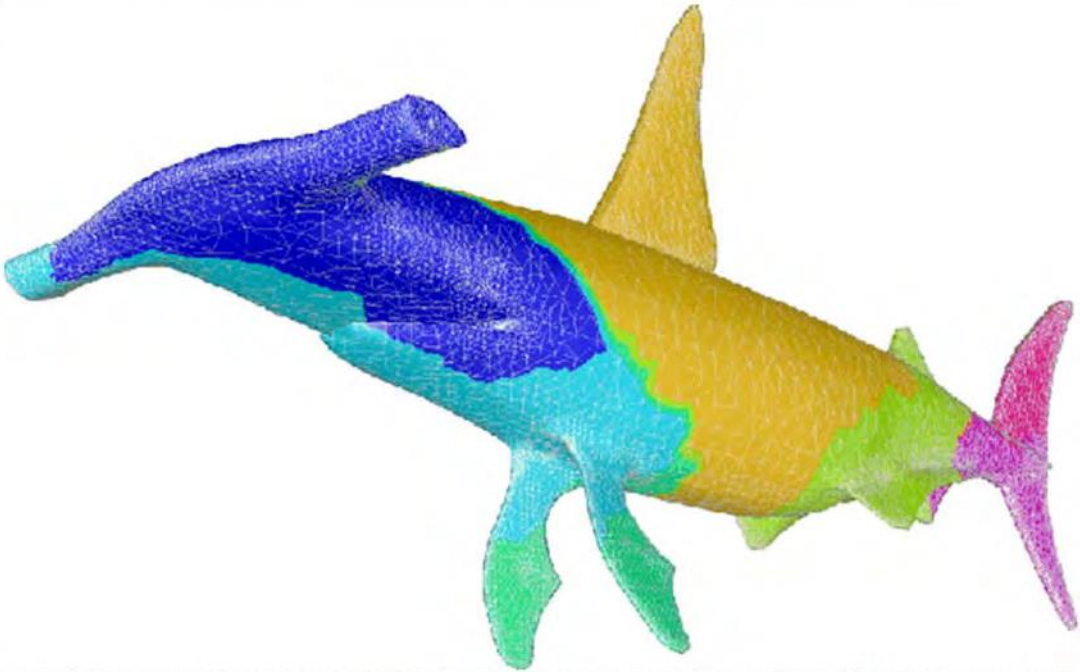
Bei Klima häufig: **Multiple** Program, Multiple Data (MPMD)

Z.B. MPI-ESM (MPI-M Earth System Model)

- Atmosphärenmodell: ECHAM (192 Prozesse)
  - Alleinelaufend als MPI/OpenMP-Programm
- Ozeanmodell: MPI-OM (63 Prozesse)
- Modellkoppler: OASIS3 (1 Prozess)

Wird abgebildet auf 256 Prozessorkerne

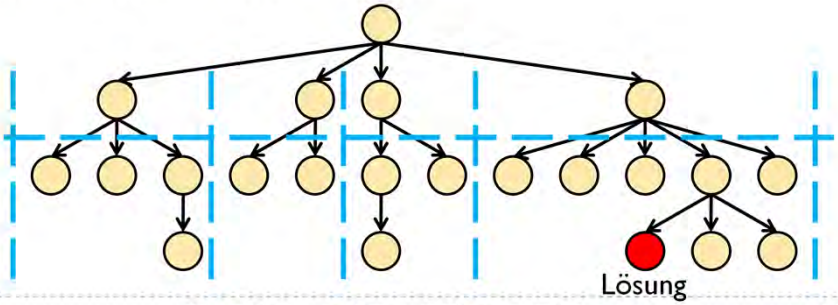
Zurück zur Strömungsmechanik ☺



Beispiel 3: Suchbaumverfahren (1/3)

(Das Beispiel illustriert eine wichtige Problemklasse)

- An jeder Position mehrere Fortsetzungen möglich
- Probleme
  - Ebene der Lösung(en) unbekannt
  - Lastausgleich zwischen den Prozessoren
  - Feststellung des Programmendes



Beispiel 3: Suchbaumverfahren (2/3)

Baumsuche mit **gemeinsamem** Speicher

- Thread berechnet Baum bis zur Ebene j und gibt Beschreibung in eine Warteschlange
- Verfügbare Threads entnehmen Beschreibung aus der Warteschlange und berechnen den verbleibenden Baum

- Gute parallele Implementierung
  - Lastausgleich ist kein Problem
  - Feststellung des Berechnungsendes: setze Ende-Bit; andere Prozesse prüfen regelmäßig

Beispiel 3: Suchbaumverfahren (3/3)

Baumsuche mit **verteiltem** Speicher

- Prozeß i berechnet bis zum Level j und gibt Beschreibung in eine lokale Warteschlange
- Untätige Prozesse kontaktieren Prozeß i, bekommen Elemente der Warteschlange als Nachricht und berechnen den restlichen Baum

Gute parallele Implementierung

- Lastausgleich kein Problem, aber mehr Aufwand
- Feststellen des Berechnungsendes: sende Ende-Nachricht an alle anderen Prozesse; diese prüfen regelmäßig

Zusammenfassung zu den Beispielen

- Es gibt verschiedenste Varianten, Code zu parallelisieren
- **Die konkrete Variante beeinflusst die maximal erzielbare Leistung des Verfahrens**
- ***Normalerweise kann die Effizienz der parallelen Programmvariante nicht am sequentiellen Programm bestimmt werden***
- **Datenaufteilung ist in den meisten Fällen einfacher zu programmieren**
- Suchbaumverfahren sind oft trivial zu parallelisieren

Parallele Programmierung
Zusammenfassung

- Parallelisierung von Programmen ist eine komplexe Tätigkeit
- Man nutzt Code-Aufteilung und Datenaufteilung
- Datenaufteilung meist einfach und effizient
- Werkzeuge: Programmierbibliotheken, Erfahrung
- Deutliche Unterschiede zwischen numerischen und nichtnumerischen Algorithmen
- Effizienz der Parallelisierung selten vorhersagbar