



## Lösungen zum Übungszettel 10

### Aufgabe 1:

a) PCAM Design-Methode:

1. (Spezifikation)
2. Partitionierung  
Problemabhängige Zerlegung der Berechnung und der Daten in Teile (Tasks). Ohne Berücksichtigung der Zielarchitektur. Fokus auf Such nach möglichen parallelisierbaren Stellen.
3. Kommunikation  
Analyse der Datenabhängigkeit. Festlegung der Kommunikationsanforderungen.
4. Bündelung  
Zusammenfassung stark zusammenhängender Teile zu größeren Tasks. Ziel: Effizienzsteigerung durch Kostenminimierung (Kommunikation).
5. Abbildung  
Abbildung der resultierenden Struktur auf konkrete Zielarchitektur. Aufgaben werden den Prozessoren zugeordnet. Ziel: Maximierung der Prozessorauslastung (statische oder dynamische Lastenverteilung)

b) Mögliche Gebietszerlegung:

- a) Eingabematrizen/-vektoren zu Tasks
  - b) Ausgabevektoren zu Tasks
- => Datenparallelität

Funktionszerlegung:

- a) Multiplikationen
  - b) Additionen
- => Kontrollparallelität

### Aufgabe 2:

a) Vergleich der Prioritäten, polish-notation, Syntax-Tabelle, Distributivmethode

b) 1. Durchlauf:

$$T1 * d + T2 + T3$$

$$T1 = (b * c)$$

$$T2 = (a + e)$$

$$T3 = (f + g)$$



---

2. Durchlauf:

$T4 + T5$

$T4 = (T1 * d)$

$T5 = (T2 + T3)$

3. Durchlauf:

$T0 = T4 + T5$

**Ausgabe:**  $((b * c) * d) + ((a + e) + (f + g))$

Die Ausführungszeit wird von  $TA=6$  auf  $TA=3$  reduziert!

### Aufgabe 3:

- a) Idee: Compiler extrahiert automatisch die Parallelität aus einem sequentiellen Programm und generiert eine MPI-basierte oder Thread-basierte parallele Implementierung.  
Ansatz: In den meisten HPC-Anwendungen wird die meiste Rechenzeit für (geschachtelte) Schleifen benötigt.  
Analyse von Datenabhängigkeiten in Schleifen zur Erkennung von sicher parallelisierbaren Schleifen. Generierung von parallelem Code für einfache Schleifen mit Barrieren-Synchronisation am Schleifenende. Einige einfache Code-Transformationen werden zur Erhöhung der Anzahl parallelisierbarer Schleifen durchgeführt.
- b) 1) Schleifen mit Index-Abhängigkeiten zwischen Iterationen nicht automatisch parallelisierbar.  
2) Einfache Schleife (d.h. ohne Index-Abhängigkeiten zwischen Iterationen), gut automatisch parallelisierbar.  
3) Schleifen mit mehreren Ausgängen nicht automatisch parallelisierbar.
- c) Probleme:
  - a. Loop Carrier Dependencies
  - b. Schleifen mit Index-Abhängigkeiten zwischen Iterationen
  - c. Schleifen mit Funktionsaufrufen
  - d. Zeigerbasierte Schleifen
  - e. Schleifen mit (evtl. versteckten) Reduktionen
- d) Code-Transformationen können die Probleme lösen:
  - a. Skalare Variablen durch Feldvariablen ersetzen
  - b. Variablen in Schleifen umbenennen
  - c. Anweisungen oder Funktionen einsetzen
  - d. Reduktionen von Vektoren automatisch erkennen und durch Aufrufe von parallelen Reduktionsfunktionen ersetzen