



9. *Algorithmische Skelette*

- Abstrakte parallele Berechnungsschemata
- 



Skeletons

- In parallelen Algorithmen treten wiederkehrende Berechnungsmuster auf. Diese Muster bestehen aus
 - **Berechnungen** und
 - **Interaktionen** zwischen ihnen.

Von den Details der parallelen Berechnungen wird abstrahiert.

- Algorithmische Skelette bestehen aus
 - **einem Berechnungsschema (Funktion höherer Ordnung)**
 - **mit einer parallelen Auswertungsstrategie und**
 - **einem Kostenmodell zur Abschätzung der Ausführungszeit.**
- **Campbell's classification:**
 - **divide and conquer (rekursive Partitionierung)**
 - **task queue (work pool, master worker)**
 - **systolic (pipeline)**

Divide and Conquer

- Berechnungsschema als Funktion höherer Ordnung:

$d\&c :: (a \rightarrow Bool) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow [a]) \rightarrow ([b] \rightarrow b) \rightarrow a \rightarrow b$

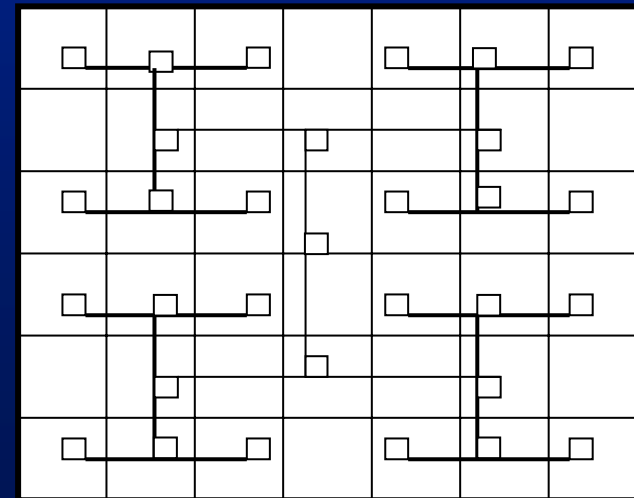
$d\&c \quad \text{trivial} \quad \text{solve} \quad \text{divide} \quad \text{conquer} \quad p$

$= \text{if (trivial } p) \text{ then solve } p$

$\quad \text{else conquer (map (d\&c trivial solve divide conquer) (divide } p))$

- parallele Implementierungsstrategien:

- idealisierte Implementierung auf Prozessorbaum
- H-Baum Implementierung binärer d&c auf dem Grid:



Task Queue - Farm - Master/Worker

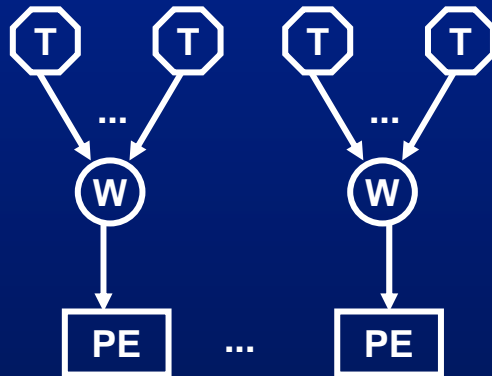
- Berechnungsschema:

farm $:: (a \rightarrow b \rightarrow c) \rightarrow a \rightarrow [b] \rightarrow [c]$

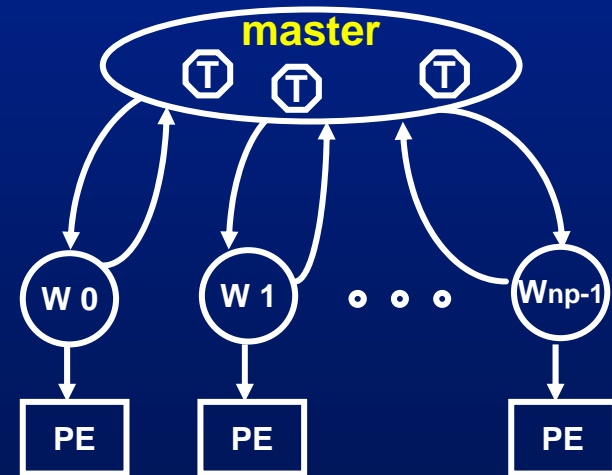
farm f env $= \text{map } (f \text{ env})$

- Implementierung:

- statische Lastverteilung:



- - dynamische Lastverteilung:



- Kostenmodell (statische Verteilung): $t_{\text{farm}} = t_{\text{setup}} + n/p (t_{\text{solve}} + 2t_{\text{comm}})$

Systolisches Schema - Pipelining

- Berechnungsschema:

pipe :: [[a] -> [a]] -> [a] -> [a]

pipe = foldr (.) id

- Implementierung:

lineare Pipeline von Prozessen



- Kostenmodell: $t_{\text{pipe}} = t_{\text{setup}} + (t_{\text{stage}} + t_{\text{comm}})(p + n-1)$



Skelettprogrammierung

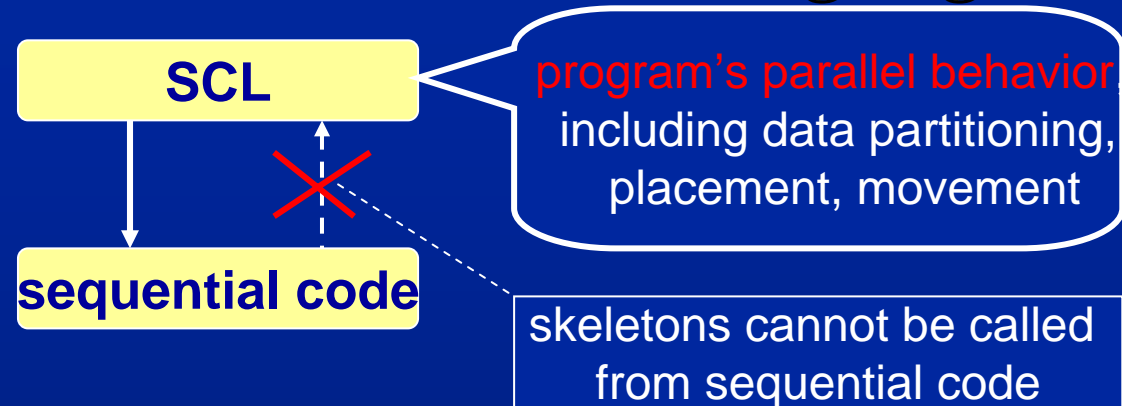
- feste Anzahl von Skeletten höherer Ordnung
(algorithmische Skelette)
- verschiedene hoch optimierte Implementierungen für verschiedene Zielarchitekturen
(Architekturskelette)
- **Programmiermethodologie:**
 - wähle ein passendes Skelett
 - verwende diese in einem Programm
 - schätze die zu erwartende Leistung => Kostenmodell
 - revidiere Design, falls notwendig => Transformationsregeln

Entwicklung

- **Ursprung:**
 - **PhD thesis von M. Cole (Univ. of Edinburgh, 1988)**
 - wenige komplexe Skelette:
 - FDDC (fixed degree divide & conquer)
 - IC (iterative combination)
 - TQ (task queue)
- **erste Systeme:**
 - P³L Pisa Parallel Programming Language [Pelagatti et al. 1995]
basierend auf imperativer Berechnungssprache
 - SCL Structured Coordination Language [Darlington et al. 1995]
basierend auf funktionaler Berechnungssprache

SCL - Structured Coordination Language

layered skeletal approach:



- provides lower level of detailed control through **explicitly distributed arrays ParArray** with skeletons:

partition :: Partition_pattern -> SeqArray index a

-> ParArray index (SeqArray index a)

gather :: Partition_pattern -> ParArray index (SeqArray index a)

-> SeqArray index a

align :: ParArray index a -> ParArray index b -> ParArray index (a,b)

- higher level skeletons:

- elementary / computational / communication skeletons



SCL Skeletons

- **elementary skeletons** (data parallel operations over distributed arrays)
 - `map` $:: (a \rightarrow b) \rightarrow \text{ParArray index } a \rightarrow \text{ParArray index } b$
 - `imap` $:: (\text{index} \rightarrow a \rightarrow b) \rightarrow \text{ParArray index } a \rightarrow \text{ParArray index } b$
 - `fold` $:: (a \rightarrow a \rightarrow a) \rightarrow \text{ParArray index } a \rightarrow a$
 - `scan` $:: (a \rightarrow a \rightarrow a) \rightarrow \text{ParArray index } a \rightarrow \text{ParArray index } a$
- **computational skeletons** (parallel control flow)
 - `farm` $:: (a \rightarrow b \rightarrow c) \rightarrow a \rightarrow \text{ParArray index } b \rightarrow \text{ParArray index } c$
`farm f e = map (f e)`
 - `spmd` $:: [(\text{ParArray index } a \rightarrow \text{ParArray index } a, \text{index} \rightarrow a \rightarrow a)] \rightarrow \text{ParArray index } a \rightarrow \text{ParArray index } a$
`spmd [] = id`
`spmd ((gf, lf) : fs) = (spmd fs) . gf . (imap lf)`
 - ...
- **communication**
 - `rotate` $:: \text{Int} \rightarrow \text{ParArray index } a \rightarrow \text{ParArray index } a$
 - ...

Standardisierung - Skelettbibliotheken für MPI -

- H. Kuchen (Univ. Münster, 2002)
 - Skelette als **C++ Bibliothek auf der Basis von MPI**
 - Polymorphie über Templates simulieren
 - Funktionen höherer Ordnung mittels überladener Operatoren
 - Unterscheidung
 - **datenparallele Skelette**
 - Manipulation verteilter Datenstrukturen
 - Berechnungsskelette: map, fold, zip, scan ...
 - Kommunikationsskelette: rotate, broadcast, permute, gather ...
 - **kontrollparallele Skelette**
 - pipe, farm, d&c, search
 - 1. Erzeuge Prozesstopologie
 - 2. starte Tasks (parallele Prozesse)





Standardisierung

- Skelettbibliotheken für MPI II -

- M. Cole, A. Benoit (Univ. of Edinburgh, 2004)
 - **eSkel** - Edinburgh Skeleton Library

Ziel: Erweiterung von kollektiven Operationen (einfache Skelette)
in MPI um algorithmische Skelette

Zur Zeit: 5 Skelette

1. pipeline
 2. farm
 3. deal (wie farm, ohne farmer mit zyklischer Taskverteilung)
 4. haloswap -> iterative Approximation
 - Schleife mit
 - local update
 - check for termination
 5. butterfly -> divide & conquer in hypercube
-
-

Pipeline Skelett

```
void Pipeline (int ns, Amode_t amode[],  
              eSkel_molecule_t * (*stages[])(eSkel_molecule_t ), int col,  
              Dmode t dmode, spread t spr[], MPI_Datatype ty[],  
              void *in, int inlen, int inmul, void *out, int outlen,  
              int *outmul, int outbuffsz, MPI_Comm comm)
```

=> 15 Parameter:

- 3 Parameter für Pipeline Eingaben
- 4 Parameter für Pipeline Ausgaben
- 3 Parameter für Pipeline-Stufen-Spezifikation
- 4 Parameter für Schnittstellen & Modi
- 1 Parameter für Kommunikator