Die Multicore-Herausforderung

M. Anton Ertl
TU Wien

Hardware-Entwicklung

- (Gordon) Moore's Gesetz: Doppelt soviele Transistoren alle 18/24 Monate
- Nicht-Moore's Gesetz: Prozessor-Geschwindigkeit wurden alle 2 Jahre doppelt so schnell
- Was tun mit den Transistoren?
  - Multi-Cores
  - Mehrere Threads pro Core (SMT, HT)
- Aber: Wie nutzt man Threads?

Automatische Parallelisierung sequentieller Programme

- 30 Jahre Forschung
- aufwändige und komplexe Compiler
- mäßiger Erfolg, besonders bei allgemeinen Programmen

Händische Parallelisierung

- Erfolgreich bei Scientific Computing
  Daten-Parallelismus (DOALL)

- Schwieriger bei allgemeinen Programmen
  Multi-Thread-Programmierung mit expliziter Synchronisation

Warum ist Multi-Thread-Programmierung schwierig?

- Neue Arten von Bugs
- Race Conditions, Deadlocks, Livelocks
- Schwieriger zu finden (nichtdeterministisch)

Warum ist Multi-Thread-Programmierung schwierig?

- Thread-Erzeugung ist sehr teuer
- Synchronisation ist teuer
- $\Rightarrow$ Parallelisierung nicht zu feinkörnig
- $\Rightarrow$ schlecht für Modularität, allgemeine Verwendbarkeit
- Lieber zuerst sequentielle Version optimieren

Pipes/Streams

- `cat */Punkte|sort -n|uniq -c`
- Von Programmierern angenommen
- Einzelnen Stufen sind wiederverwendbare Module
- Multi-Thread-Implementierung relativ billig
- Single-Thread-Implementierung: Coroutinen
  ähnlich einfachem Forth-Multitasker
- Umschalten zwischen Implementierungen nach Lage
- Grobkörniger Paralellismus, zur Laufzeit einstellbar
- StreamIt (MIT) für DSP-Programme

Forth

- Pipes anwendbar in vielen Sprachen
- In Forth zusätzlicher Nutzen:
  Mindert das Stack-Tiefen-Problem
  Weitere Form des Faktorisierens
- SeaForth

Beispiel ohne Pipes

```
: faxpy ( ra f-addr-x nstride-x f-addr-y nstride-y ucount -- )
    \ vy=ra*vx+vy
    >r swap 2swap swap r> 0 ?DO
        fdup dup f@ f* over + 2swap dup f@ f+ dup f!
        over + 2swap
    LOOP
    2drop 2drop fdrop ;
```

Beispiel mit Pipes

```
: v@ ( f-addr nstride ucount -- )
    0 ?do
        over f@ fput
        tuck + loop
    endput 2drop ;

: vf* ( ra -- )
    begin fget? while
            fover f* fput repeat
    fdrop ;

: v+! ( f-addr nstride -- )
    begin fget? while
            over f@ f+ over f!
            tuck + repeat
    2drop ;

: faxpy ( ra f-addr-x nstride-x f-addr-y nstride-y ucount -- )
    rot rot 2>r ['] v@ xxx|
    ['] vf* f|
    2r> v+! ;
```

Status

- Nur eine Idee, keine Implementierung

Offene Fragen

- Wie werden Pipe-Stufen angefangen?
  Parameter-Übergabe
  Syntax

- Wie werden sie beendet?
  Abbruch durch beliebige Stufe

- Statt linearer Pipes Bäume (Mergesort) oder DAGs

- Referenzen auf offene Pipes

Grenze für Multi-Cores

- Bandbreite zum System (v.a. RAM)
  8-16 cores/Sockel für PCs und Server

- Bandbreitensparend programmieren

- Kann Forths Sparsamkeit helfen?

- Zig MB (cache) on-chip