Die Convex SPP1000 am RZ

Übersicht:

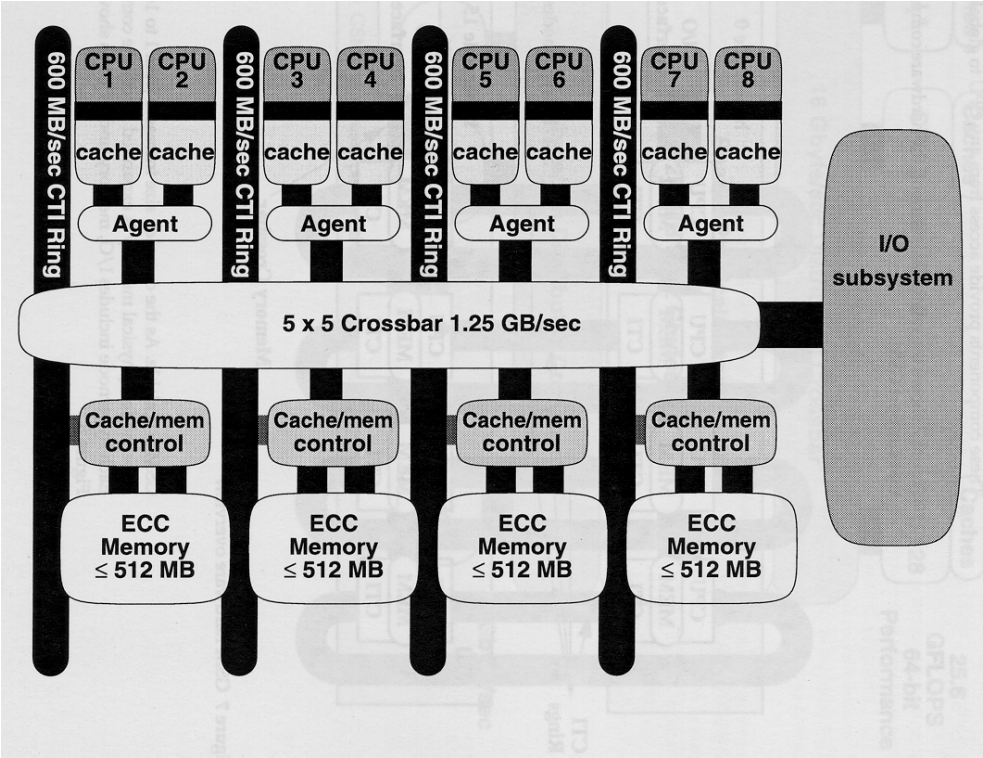
1. Architektur des HP/Convex-Parallelrechners SPP1000

2. Betriebsweise der SPP1000/16 (hydra) am RZ

3. Einige Benutzungs-Hinweise

4. Programmierung der SPP

1 Architektur des HP/Convex-Parallelrechners SPP1000



Technische Daten:

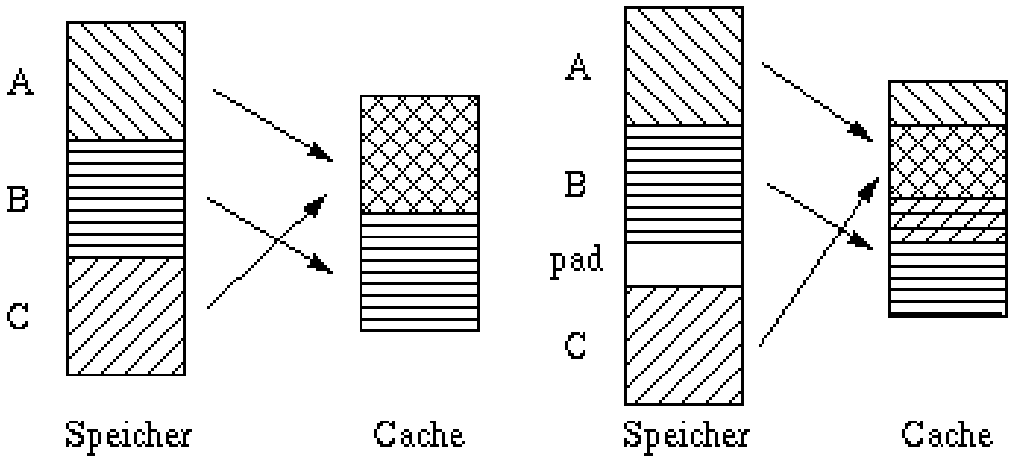
Prozessor	HP-PA 7100 / 100 MHz
Prozessor-Cache	1 MB Daten / 1 MB Instruktionen
5x5-Crossbar	5 x 250 MB/s
4fach CTI-Ring	4 x 600 MB/s
I/O-System	250 MB/s, 4xSBus

Konfiguration des SPP-Systems am RZ (hydra)

2 Hypernodes / 16 CPUs
2 GB Hauptspeicher
24 GB Plattenspeicher
DAT-Tape
FDDI
Teststation HP 9000/715

- CPU HP-PA 7100/ 100 MHz
  pro Takt gleichzeitig 1 Mult und 1 Add
  200 MFLOPS Peak

- Cache 1MB Daten / 1 MB Instruktionen
  direct mapped
  Problem: cache thrashing



$$C = A + B$$

Speicher-Hierarchie:

Speichertyp	Zugriff	Latenzzeit
prozessor cache	CPU	10 ns
node private	Hypernode	500 ns
CTI cache	Hypernode	500 ns
global	alle	2000 ns

- SPP-UX

  Mach Microkernel, 1 Kernel pro Node

  div. Server, 0-1 pro Node
    - Process-Management
    - File-System
    - Subcomplex
    - Network

  HP-UX ABI
    - Systemcalls
    - Signale
    - Shared Libraries
    - System-Dateien
    - Hilfsprogramme
    - ...

- Scheduling

  SPP-Prozeß $\rightarrow$ Mach-Tasks (1 pro Node)
  $\rightarrow$ Threads

  Thread bleibt immer auf seiner Node
  Thread bleibt möglichst auf seiner CPU (Cache!)

2 Betriebsweise der SPP1000/16 (hydra) am RZ

2.1 Zugang

erreichbar unter `hydra.rz.tu-harburg.de`

Aufgaben:

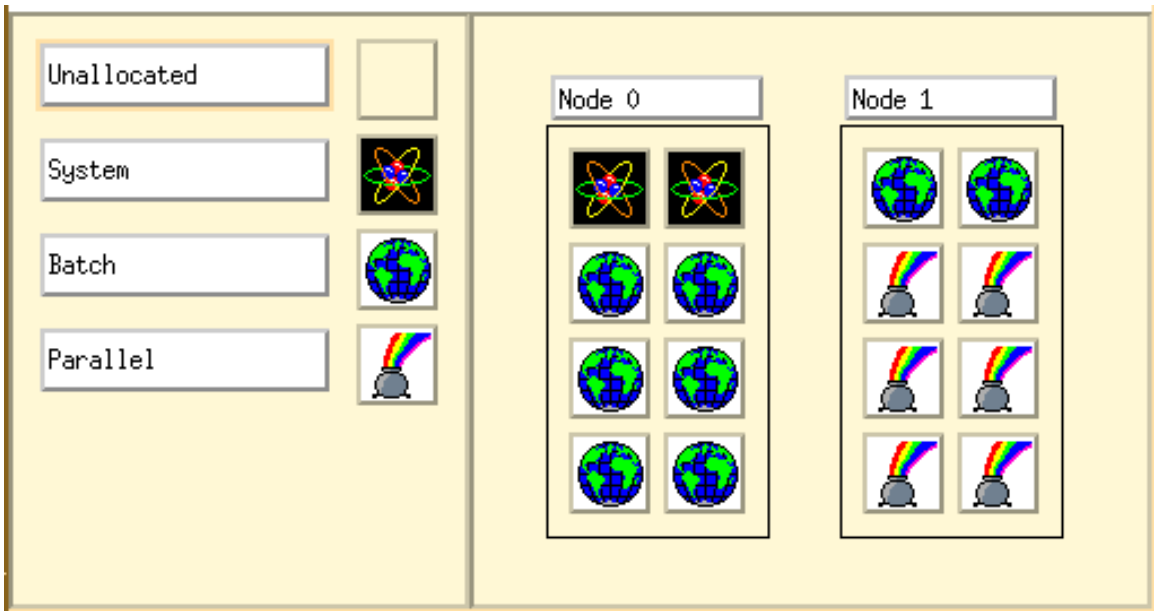
Ablösung der C3
skalare Jobs mit großem Hauptspeicherbedarf
(> 100 MB)
Parallele Programme
Entwicklung und Produktion

2.2 Anwendungsprogramme

Flow-3d
Ansys (auch in einer parallelen Version!)
Marc
Matlab
Maple
Uniras

2.3 Subcomplex-Konfiguration

Name	CPUs	Speicher	Aufgabe
System	2	16 MB	System-Prozesse, interaktiv
Batch	8 (6)	2x16 MB	skalare Batchjobs
Parallel	6 (8)	48 MB	parallele Batchjobs Tests paralleler Anwendungen (interaktiv: < 10 min)



2.4 Batchbetrieb

Name	CPU [h]	Speicher MB	Subcomplex	Run
short	0.5	200	Batch	3
medium	4.0	300	Batch	3
long	24.0	700	Batch	2
parallel	24.0	700	Parallel	1

beliebig viele Jobs pro Benutzer
Änderung der Abarbeitungsreihenfolge
nach Runden (vgl. C3)

Kettenjobs möglich mit

```
qsub -q <queue> -r <name> -C <script>
```

2.5 Plattenplatz

permant: auf dem Fileserver
- Quoten
- regelmäßiges Backup

temporär: lokale Platten (schnell!)
- keine Quoten
- kein Backup
- 2 Bereiche:
  - usertemp festes Verzeichnis
    - nach 7 Tagen gelöscht
  - scratch sehr schnell
    - nur in Batchjobs
    - erreichbar über $TMPDIR
    - bei Job-Ende gelöscht
    - garantiert (ziemlich) leer

3 Einige Benutzungs-Hinweise

3.1 Programm-Attribute

Ändern von Programm-Attributen mit `mpa`

Beispiele:

`mpa -max <N> <prog> <options>`
prog läuft auf max. N Prozessoren

```
mpa -specific node_private \
  -stacktype node_private \
  -private node_private \
  -threadstack node_private \
  <prog> <options>
```
prog benutzt lokalen Speicher statt globalem

`mpa -data 600000000 <prog> <options>`
prog darf 600 MB Datenbereich anlegen

3.2 System-Beobachtung

syspic X-Window-Tool
CPU-Auslastung, Speicher, Paging, ...
sehr CPU-intensiv

pot Informationen zu laufenden Prozessen
viele Konfigurationsparameter
sehr CPU-intensiv

ps System-V-Variante, etwa

ps -ef	alle Prozesse
ps -f -u et9qq	Prozesse von et9qq

cnx_ps Optionen wie ps
-T für Thread-Informationen

3.3 Dokumentation

man-Kommando

gedruckte Manuals im RZ

RZ-WWW (Betriebsorganisation/SPP1000)

Newsgruppe tuhh.hydra

RZINFO-Artikel (auch im WWW)

4 Programmierung der SPP

4.1 Allgemeines

fc, cc, wie auf C3
fort77 (HP-Fortran-Compiler)

Debugger CXdb
Profiler CXpa
Application-Profiler (ipo)

Veclib, Lapack, teilweise parallel
NAG (skalar)

4.2 Shared-Memory-Programmierung

automatische Parallelisierung bei -O3

- Aufteilung von Schleifen auf mehrere Threads
- soviele Threads wie Prozessoren (zur Laufzeit)
- alle Daten im globalen Speicher (default)

Compiler-Direktiven

- Schleifen zum Parallelisieren auswählen
- Daten lokalisieren
- Tasks definieren
- Threads synchronisieren

CPSlib

- Compiler Support Library: benutzen die Compiler
- explizites Thread-Management

Speicher-Modell (aus Programmierer-Sicht):

thread private	lokale Daten für einen Thread
node private	lokale Daten in einem Node
near shared	global, aber im anfordernden Node abgelegt
far shared	global, in 4kB-Blöcken durch alle Nodes verteilt
block shared	global, ein großer Block pro Node

4.3 Message-Passing-Programmierung

PVM:

Version 3.3.8, von Convex angepaßt
interne Kommunikation über Shared Memory
nur ein PVM-Dämon (pro Subcomplex)
Debuggen mit CXdb
Profiling mit CXpa oder Tracing mit xpvm
Skript zum Aufräumen:

/usr/convex/pvm/lib/pvm_clean

Umgebungsvariable:

PVM_ARCH = CSPP
PVM_ROOT = /usr/convex/pvm

MPI:

Version 1.0.11 von mpich, von Convex angepaßt
momentan langsamer als PVM
schnelle Version in Vorbereitung
die Standard-Message-Passing-Bibliothek