Einführung in PVM

Christian Schaller

email: schaller@lrz-muenchen.de

LEIBNIZ - RECHENZENTRUM
DER BAYERISCHEN AKADEMIE DER WISSENSCHAFTEN

Übersicht

- Einführung in die Parallelverarbeitung
- Programmiermodelle
- Das PVM-Programmiermodell
- LRZ-Installation
- Gruppenfunktionen
- Programmiertechnik
- Portierung existierender Programme
- Paralleles Debugging
- Tools: Hence, XPVM
- Interna von PVM

Historie

- Sommer 1989: PVM 1.0
- 1991: HeNCE
- 1993: PVM 3, XPVM
- Unterstützt über 40 UNIX-Dialekte z.B.:

ALPHA	DEC Alpha/OSF-1
ALPHAMP	DEC Alpha multiprocessor/OSF
CRAY	Cray
HPPA	HP 9000 PA-Risc
I860	Intel RX Hypercube
KSR1	Kendall Square
LINUX	80[34]86 running Linux
NEXT	NeXt
PGON	Intel Paragon
RS6K	IBM/RS6000
SGI	Silicon Graphics IRIS
SUN4	Sun 4, 4c, sparc, etc.
SUNMP	Sun 4 multiprocessor

Literatur und weitere Informationen

- LRZ: `info lrzsw pvm`
- FTP-Server am LRZ:
  `ftp ftp.lrz-muenchen.de`
  `pub/comp/parallel/pvm3`
  Quellen der LRZ-Installation, weitere Doku.
  `pub/comp/parallel/pvm3/pvm-book.ps.Z`
  PVM-Buch (Postscript)
  `pub/comp/parallel/pvm3/ps/ug.ps.Z`
- WWW:
  - Kursunterlagen PVM-Kurs (LRZ)
    `http://www.lrz-muenchen.de/LRZ/PUBL/INFO/SW/PVM_KURS/homepage.html`
  - PVM-Homepage
    `http://hebe.nectar.cs.cmu.edu:8001`
  - PVM in Cornell
    `http://www.tc.cornell.edu/Edu/Education.and.Training.Materials/pvm.html`
- News-Gruppe `comp.parallel.pvm`, `lrz.parallel`
- aktuelle Quellen auf `elib.zib-berlin.de/netlib`

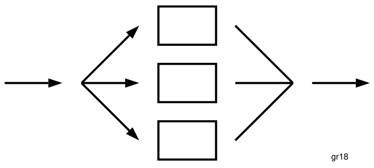
PVM als Standard ...

- Stärkere Verbreitung von WS als Arbeitsplatzrechner
- Wachsender Bedarf an Rechenleistung
- Einfache Installation auf WS-Netz
- Portabilität
- Entwicklung von Tools (XPVM, HeNCE,XAB)
- Schnellere Netze (besseres Komm./Rechenl.)
- Stärkeres Interesse an Parallelrechnern (grand challenges)
- Heterogene Netze (Vektorrechner, Parallelrechner, Graphik-WS)
- Skalierbarkeit
- Fehlertoleranz

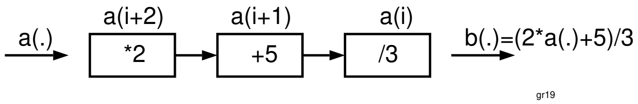
Begriffe zur Parallelverarbeitung

Formen der Parallelisierung

- Nebenläufigkeit



- Pipelining



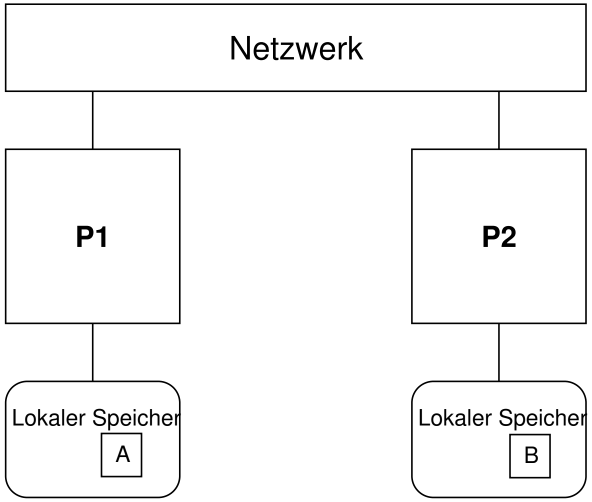
Granularität

- Programme
- Prozesse
- Gruppen von Maschinenbefehlen
- Teile von Maschinenbefehlen

Programmiermodelle (1)

- Message Passing

Beispiel: `A=A+B` auf Prozessor **P1**



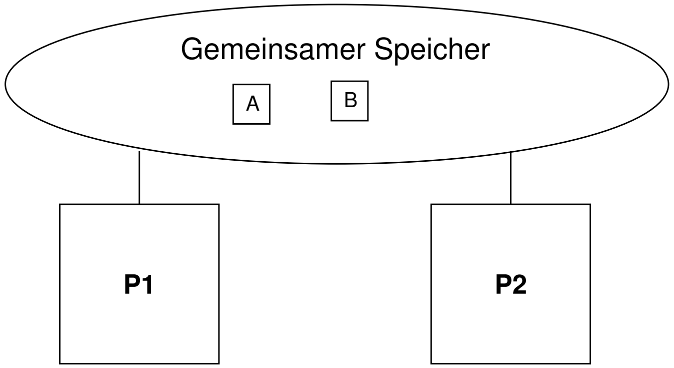

```
receive(B)from P2          send(B)to P1
A=A+B
```

gr01

Programmiermodelle (2)

- Shared Memory

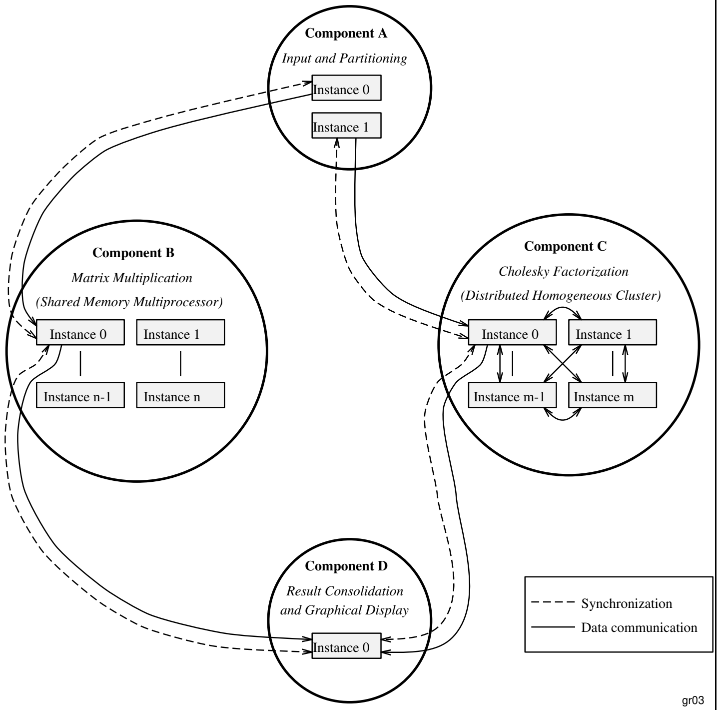
`A=A+B` auf Prozessor **P1**




```
A=A+B
```

gr02

PVM-Programmiermodell (1)



PVM-Programmiermodell (2)

- Parallelität auf Basis von UNIX-Prozessen
- Asynchrone Interaktion einzelner Prozesse (Message Passing)
- Programmierung in Fortran oder C mit gängigem Compiler
- Synchronisation der Prozesse durch Events/Messages
- PVM-Direktiven für
  - Prozeß-Verwaltung (Start,Beenden)
  - Kommunikation (send,receive,broadcast)
  - Synchronisation (barriers,rendevouz)
  - Dynamische Konfiguration, Signale
- Manuelle Partitionierung und Scheduling

Performance-Daten

Quelle: V. Sunderam, Emory Univ.

Cholesky-Algorithmus zur Matrix-Faktorisierung

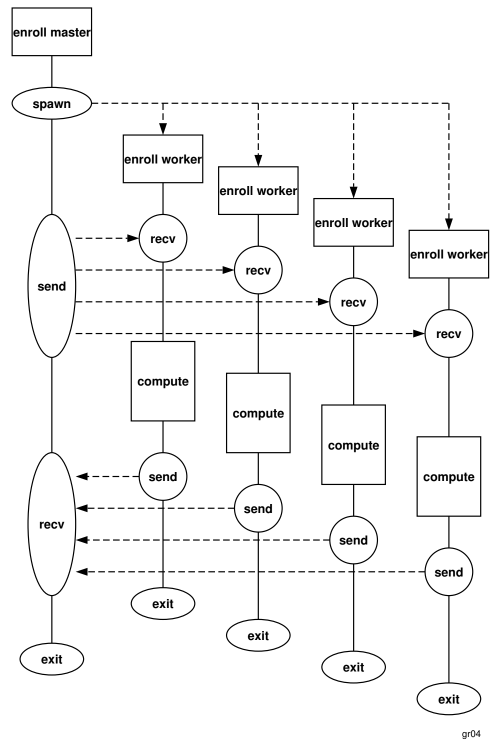
SUN Sparc im Vergleich zu iPSC/2.
Rechenleistung 3:1, Kommunikationsl. 1:3

Prozes-	Problemgröße (Ordnung der Matrix)							
soren	100		200		500		1000	
2	2	2	6	10	69	81	380	456
4	2	3	5	7	40	43	282	288
8	4	3	8	8	34	35	177	190
16	9	3	19	8	50	35	161	141

Preis / Performance (Wissenschaftl. Anwendung)

Rechner	Listenpreis	MFlops	MFl./$M
PVM (10 WS)	$ 252 K	207	822
Vektorr. (8 Proz.)	$ 20 M	2290	114

PVM-Anwendung (1)



PVM-Anwendung (2)

- Anwendung besteht aus mehreren Komponenten (*Prozesse, Tasks*)
- Interaktion durch Message Passing und Synchronisation
- Jeder Prozeß hat eine eigene `taskid`. Gruppen von Tasks haben zusätzlich `component name` und `instance number`
- Jeder Prozeß ist ein sequentielles Programm in C oder Fortran mit zusätzlichen PVM-Direktiven
- Mehrere Rechenmodelle sind möglich (Baum, SPMD, Master-Slave)
- Verteilung von Arbeit auf die Tasks manuell
- Scheduling der einzelnen Prozesse automatisch (unflexibel!) oder explizit. Granularität: Prozeß-Ebene!

Das PVM-System

- **Der Dämon**
  - Generiert den *virtuellen Parallelrechner*
  - Interaktiv: Kontrolle über die PVM-Konsole (Abfragen der Konfiguration, reset, ps) (Kommando `pvm3.setup`)
  - Im Hintergrund (ohne Kontrolle) (Kommando `pvm3.setup -c <hostf.>`)
- **PVM-Bibliotheken**

`libpvm.a`	C-Routinen
`libfpvm3.a`	Fortran-Routinen
`libgpvm3.a`	für Gruppenfkten

Fortran- und C-Schnittstelle

PVM-Direktiven im Programm

C:	`mytid = pvm_mytid();`
	(call by value)
Fortran:	`call pvmfmytid(mytid)`
	(call by reference)

Regel (C $\rightarrow$ Fortran):
Ersetze `pvm_` durch `pvmf` und füge den Rückgabeparameter in die Parameterliste ein.

Beim Linken

C:	`libpvm.a`
Fortran:	`libpvm.a` und `libfpvm3.a`
	(`libfpvm3.a` enthält nur Wrapper)
optional:	`libgpvm3.a` für Gruppenfkten

Prozeß-Verwaltung (C)

- `tid pvm_mytid():`
  Prozeß meldet sich beim Dämon an
- `info pvm_exit():`
  Prozeß meldet sich beim Dämon ab
- `info pvm_halt():`
  Shutdown von PVM (Dämon)
- `info pvm_kill(tid):`
  terminiert den Prozeß `tid`
- `numt pvm_spawn(*task, **argv, flag, *where, ntask, *tids):`
  startet neue PVM-Prozesse
- `tid pvm_parent():`
  ergibt die `tid` des Prozesses, der den aufrufenden Prozeß gestartet hat

Prozeß-Verwaltung (Fortran)

- `call pvmfmytid(tid)`:
  Prozeß meldet sich beim Dämon an
- `call pvmfexit(info)`:
  Prozeß meldet sich beim Dämon ab
- `call pvmfhalt(info)`:
  Shutdown von PVM (Dämon)
- `call pvmfkill(tid, info)`:
  terminiert den Prozeß `tid`
- `call pvmfspawn(task, flag, where, ntask, tids, numt)`:
  startet neue PVM-Prozesse
- `call pvmfparent(tid)`:
  ergibt die `tid` des Prozesses, der den aufrufenden Prozeß gestartet hat

pvm_spawn

- C: `int numt = pvm_spawn(char *task, char **argv,int flag,char *where, int ntask, int *tids)`
- Fortran: `call pvm_spawn( task, flag, where, ntask, tids)`

Parameter:

- `task` Name des ausführbaren Programms
- `argv` Array von Argumenten für `task`
- `flag` Optionen: Wahl der Maschine, debug
- `where` Spez. Maschine od. Architektur
- `ntask` Anzahl der Tasks
- `tids` Array mit `tids` der Tasks
- `numt` Anzahl der gestarteten Tasks

Beispiel: Prozeß-Verwaltung (C)

```
#include "pvm3.h"

main()                  /* spmd ! */
{
  int mytid;            /* my task id */
  int tids[NPROC];      /* array of task id */
  int num;              /* num of spawned procs */

  /* enroll in pvm */
  mytid = pvm_mytid();
  /* find out if I am parent or child */
  tids[0] = pvm_parent();
  if (tids[0] < 0){    /* I am the parent */
    tids[0] = mytid;
    num = pvm_spawn("spmdc", (char**)0, 0,
                    "", NPROC-1, &tids[1]);
    printf("parent spawned %d processes\n",num);
  }
  else{
    printf("child: my tid is: %d\n",mytid);
  }
  pvm_exit();
  exit(0);
}
```


Beispiel: Prozeß-Verwaltung (Fortran)

```
      program spmdf
      include "/sw/share/pvm3/include/fpvm3.h"
      PARAMETER( NPROC=4 )
      integer mytid, num, info
      integer tids(0:NPROC)

      call pvmfmytid( mytid )
      call pvmfparent( tids(0) )
      if ( tids(0) .lt. 0 ) then
c        ---------------
c        I am the parent
c        ---------------
         tids(0) = mytid
         call pvmfspawn('spmdf',PVMDEFAULT,'*',
     &                  NPROC-1,tids(1),num)
         print*,'parent spawned ',num,' processes'
      else
         print*,'child: my tid is: ',mytid
      end if
      call pvmfexit(info)
      stop
      end
```

Shell1: PVM-Konsole

Aufruf durch `pvm3.setup` auf allen unterstützten LRZ-Plattformen
(SUN, HP, CRAY, SGI, IBM)

```
sun1-103:  pvm3.setup
pvm> add sun2
1 successful
                    HOST     DTID
                    sun2    80000
pvm> conf
2 hosts, 1 data format
            HOST     DTID      ARCH   SPEED
            sun1    40000      SUN4    1000
            sun2    80000      SUN4    1000
pvm> ps
            HOST      TID       FLAG COMMAND
pvm> spawn spmdc
1 successful
t4000a
pvm> quit

pvmd still running.
sun1-104:
```


Shell2: Entwicklung

Shell Nr.2

```
sun1-133: pvm3.setup -x
no execution
pvm3 $ aimk
making in SUN4/ for SUN4
cc -g -I/sw/share/pvm3/include \
   -o /home/a/a283208/pvm3/bin/SUN4/spmdc \
   ../spmd.c \
   /sw/share/pvm3/lib/SUN4/libpvm3.a
pvm3 $ spmdc
parent spawned 3 processes
pvm3 $ ls -al /tmp/pvm*
-rw------- 1 a283208   18 <dat> /tmp/pvmd.11382
-rw------- 1 a283208  190 <dat> /tmp/pvml.11382
pvm3 $ cat  /tmp/pvml.11382
[t80040000] ready 3.3.7 Wed Jan 11 10:50:25 1995
[t80040000] [t40004] child: my tid is: 262148
[t80040000] [t40005] child: my tid is: 262149
[t80040000] [t40006] child: my tid is: 262150
pvm3 $ exit
sun1-134:
```

Message Passing: Send (C)

- `bufid pvm_initsend (encoding)`:
  initialisiert Send-Puffer, Kodierung XDR, Raw oder PvmDataInPlace (>3.3)
- `info pvm_pkint(&pointer, number, stride)`
  `info pvm_pkdouble(&pointer, number, stride)`
  `info pvm_pkstr(&pointer)`
  kopiert Daten in den aktiven Puffer
- `info pvm_send(tid, msgtag)`
  Sendet die Daten aus dem aktuellen Send-Puffer an `tid`.
- `info pvm_mcast(*tids, ntask, msgtag)`
  Broadcast an mehrere `*tids`
- `info pvm_psend(tid, msgtag, *buf, len, datatype)`
  Packen und Senden an `tid` in einem Aufruf, Msg-Reihenfolge bleibt erhalten (MPP!)

Message Passing: Send (Fortran)

- `call pvmfinitsend(encoding, bufid)`
  initialisiert Send-Puffer, Kodierung XDR, Raw oder PvmDataInPlace (>3.3)
- `call pvmfpack(DATYP, name, number, stride, info)`
  kopiert Daten in den aktiven Puffer
  `DATYP` = `REAL8`, `INTEGER4`, `STRING` etc.
- `call pvmfsend(tid, msgtag, info)`
  Sendet die Daten aus dem aktuellen Send-Puffer an `tid`.
- `call pvmfmcast(ntask, tids, msgtag, info)`
  Broadcast an mehrere `*tids`
- `call pvmfpsend(tid, msgtag, buf, len, datatype, info)`
  Packen und Senden an `tid` in einem Aufruf
  Msg-Reihenfolge bleibt erhalten (MPP!)

Message Passing: Receive (C)

- `bufid pvm_probe(tid, msgtag)`
  überprüft, ob eine Message eingetroffen ist
- `bufid pvm_recv(tid, msgtag)`
  Empfang einer Message (blocking)
- `bufid pvm_nrecv(tid, msgtag)`
  Empfang einer Message (non-blocking)
- `bufid pvm_trecv(tid, msgtag,`
  `& tmout)`
  Empfang einer Message (mit Timeout)
- `info pvm_upkint(&pointer,`
  `number, stride)`
  `info pvm_upkdouble(&pointer,`
  `number, stride)`
  `info pvm_upkstr(&pointer)`
  Entpacken einer Message aus dem aktiven Puffer

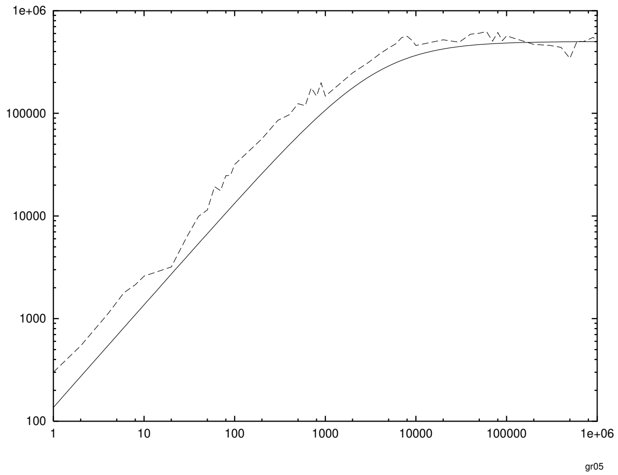
Message Passing: Receive (Fortran)

- `call pvmfprobe(tid, msgtag,`
  `bufid)`
  überprüft, ob eine Nachricht eingetroffen ist
- `call pvmfrecv(tid, msgg, bufid)`
  Empfang einer Message (blocking)
- `call pvmfnrecv(tid, msgtag,`
  `bufid)`
  Empfang einer Message (non-blocking)
- `call pvmftrecv(tid, msgtag,`
  `sec, usec, bufid)`
  Empfang einer Message (mit Timeout)
- `call pvmfunpack(DATYP, name,`
  `number, stride, info)`
  Entpacken einer Message aus dem aktiven Puffer
  `DATYP` = `REAL8`, `INTEGER4`, `STRING` etc.

Performance: Ping-Pong

Hardware: Zwei LRZ-Suns, durch Ethernet verbunden.
Software: Programm comms1 aus GENESIS

- Messages unterschiedl. Länge werden an den Nachbarn und wieder zurück geschickt.
- gemessen wird Durchsatz: Datenmenge/Zeit
- Modell für Übertragungszeit: $t_n = (n + n_{\frac{1}{2}})/r_{inf}$
  r_{inf}: asympt. Rate, $r(n_{\frac{1}{2}}) = r_{inf}/2$



Bsp: Master (Fortran) Teil 1

```
      program masterf
      include "/sw/share/pvm3/include/fpvm3.h"
      integer i, info, nproc, msgtype
      integer mytid, tids(32), who
      double precision result(32), data(100)
      character*18 nodename
      character*8 arch
c   ----- starting up all the tasks ----------
      call pvmfmytid( mytid )
      nproc = 5
      call pvmfspawn( 'slavef', PVMDEFAULT, '*',
     &                nproc, tids, numt )
c   initiate data array
      do 20 i=1,n
         data(i) = 1
20    continue
c   broadcast data to all node programs
      call pvmfinitsend( PVMDEFAULT,, info )
      call pvmfpack( integer4, nproc, 1, 1, info )
      call pvmfpack( integer4, n, 1, 1, info )
      call pvmfpack( real8,    data, n, 1, info )
      msgtype  = 1
      call pvmfmcast( nproc, tids, msgtype, info )
```

Bsp: Master (Fortran) Teil 2

```
c  wait for results from nodes
   msgtype  = 2
   do 30 i=1,nproc
      call pvmfrecv( -1, msgtype, info )
      call pvmfunpack( integer4, who, 1, 1,
  &                    info )
      call pvmfunpack( real8, result(who+1),
  &                      1, 1, info )
      print *, 'i got',result(who+1),
  &            '  from', who
30 continue
c ------ end user program --------
   call pvmfexit(info)
   stop
   end
```


Bsp: Slave (Fortran) Teil 1

```
   program slavef
   include "/sw/share/pvm3/include/fpvm3.h"
   integer  info, mytid, mtid, msgtype, me
   integer  tids(0:32)
   real*8 result, data(100)
   real*8 work

c  enroll this program in pvm
   call pvmfmytid( mytid )
c  get the master's task id
   call pvmfparent( mtid )
c ---- begin user program --------
c  receive data from host
   msgtype  = 1
   call pvmfrecv( mtid, msgtype, info )
   call pvmfunpack( integer4, nproc, 1, 1, in-
fo )
   call pvmfunpack( integer4, n, 1, 1, info )
   call pvmfunpack( real8,    data, n, 1, info )
c  determine which slave i am (0 -- nproc-1)
c  do calculations with data
   result = work( me, n, data, nproc )
```

Bsp: Slave (Fortran) Teil 2

```
c  send result to host
   call pvmfinitsend( pvmdefault, info )
   call pvmfpack( integer4, me, 1, 1, info )
   call pvmfpack( real8,    result, 1, 1, info )
   msgtype  = 2
   call pvmfsend( mtid, msgtype, info )
c ------- end user program --------
c  program finished. leave pvm before exiting
   call pvmfexit(info)
   stop
   end

   real*8 function work( me, n, data, nproc )
c  --------------------------------------
   return
   end
```


PVM am LRZ (1)- Verzeichnistruktur

- PVM-System: (SUN,HP,CRAY,IBM,SGI)
  - `/sw/share/pvm3`
  - `../doc`
  - `../examples`
  - `../man`
  - `../include`
  - `../lib`
  - `../lib/ARCH` enthält `libpvm.a`, `libfpvm.a`, `libgpvm.a`,
  - `../src`

  YEL, KSR: `/usr/local/sys/pvm3`
- Anwender-Umgebung:
  - `$HOME/pvm3`
  - Ausführbare Programme: `$HOME/pvm3/bin/$ARCH`
  - Für Verbindungsaufbau ohne Passwort: `$HOME/.rhosts`
- Header-Files:
  - C: `#include <pvm3.h>`
  - Fortran `INCLUDE '/sw/share/pvm3/include/fpvm3.h'`

PVM am LRZ (2) - Der Dämon

- `pvm3.setup`
  startet den Dämon mit Konsole
  - Interaktive Kontrolle des Dämonen
  - Dynamische Konfiguration des virtuellen Parallelrechners
  - Kommandos z.B. `help`, `add`, `delete`, `halt`, `quit`, `ps`, `reset`
- `pvm3.setup -c <hostf.>`
  startet den Dämon im Hintergrund mit Hostfile:

```
sun1
# Kommentar
sun2
yel dx=/usr/local/sys/pvm3/lib/pvmd
```

  Bei Login-Shell `sh` wird die Option `dx=..` benötigt!
- `pvm3.setup -p`
  Konfiguration mit expliziter Passwort-Validierung (Details in `info lrzsw pvm`)

PVM am LRZ (3) - aimk

`aimk` ist PVM-spezifisches, von der Architektur unabhängiges make
(Pfad-Einstellung z.B. durch `pvm3.setup -x`)
Beispiel für ein Makefile:

```
PVMLIB = $(PVM_ROOT)/lib/$(PVM_ARCH)/libpvm3.a
SDIR = ..
BDIR = $(HOME)/pvm3/bin
XDIR = $(BDIR)/$(PVM_ARCH)
CC = cc
CFLAGS = -g -I$(PVM_ROOT)/include
LIBS = $(PVMLIB) $(ARCHLIB)

default: $(XDIR)/masterc $(XDIR)/slavec

clean:
        rm -f *.o masterc masterf slavec slavef

$(XDIR)/masterc: $(SDIR)/master.c
        $(CC) $(CFLAGS) -o $(XDIR)/masterc /
              $(SDIR)/master.c $(LIBS)

$(XDIR)/slavec: $(SDIR)/slave.c
        $(CC) $(CFLAGS) -o $(XDIR)/slavec /
              $(SDIR)/slave.c $(LIBS)
```

Hilfe bei Problemen

- Master-Pvmd läßt sich nicht starten:
  (z.B.: `libpvm [pid2845]:`
  `mksocs() connect: Connection refused`)
  alte Socket-Verbindungen löschen!
  `rm /tmp/pvmd.`*uid*

- `can't start pvmd` (Slave):
  - Ausgabe (`echo`) in `.cshrc`?
  - Ist `PVM_ROOT` nicht gesetzt (Bourne-Shell)?
  - Testaufruf
    `rsh <host> $PVM_ROOT/lib/pvmd -s`
    Wenn alles in Ordnung ist:
    `[pvmd pid18718] slave_config: bad args`
    `[pvmd pid18718] pvmbailout(0)`

- `Duplicate Host`:
  Zweite PVM-Instanz löschen

- `failed to start group server`:
  `ln -s $PVM_ROOT/lib/pvmgs`
  `$HOME/pvm3/bin/$ARCH`