Voraussetzungen für die Arbeit mit PVM und MPI

(Linux im PC-Pool *„Asgard“* bzw. auf dem *„case“*-Cluster)

Secure Shell

1. Die Rechner, die für PVM oder MPI genutzt werden sollen, müssen per `remote shell` bzw. `secure shell` erreichbar sein.

   So muss bspw. folgendes Kommando ohne Passwort-Abfrage funktionieren (und den angegeben Rechnernamen anzeigen):

   ```
   ssh <rechnername> hostname
   ```

2. Vorbereitungen für `ssh`:

   Im Homeverzeichnis muss ein Unterverzeichnis `.ssh` existieren, in dem einige wichtige Files stehen:

`id_xxx`	*Private* Key (nicht lesbar für andere),
`id_xxx.pub`	*Public* Key (lesbar, enthält nur eine, aber sehr lange Zeile!),
`authorized_keys`	Liste von Public-Keys, für die `ssh` ohne Password erlaubt wird.

   Wenn noch nichts davon existiert, sollte man das Kommando `ssh-keygen` aufrufen (Standarddateinamen bestätigen; „Passphrase“ kann man mit Enter wegdrücken).
   In der aktuellen `ssh`-Version wird `ssh-keygen` auf eine der folgenden Arten gerufen (oder auch mehrere nacheinander):

   ```
   ssh-keygen -t rsa          (Dateien: id_rsa   und   id_rsa.pub)
   ssh-keygen -t dsa          (Dateien: id_dsa   und   id_dsa.pub)
   ssh-keygen -t rsa1         (Dateien: identity und identity.pub)
   ```

   (Die letzte Zeile entspricht der älteren `ssh`-Version mit Protokoll 1.)

   Wenn die Datei `authorized_keys` (im Verzeichnis `~/.ssh`)[1] noch nicht existiert, dann einfach neu anlegen:

   ```
   cat id*.pub > authorized_keys
   ```

   Wenn `authorized_keys` schon existiert, aber die Zeile aus dem neu erzeugten *Public* Key noch nicht enthält, dann nur noch anfügen:

   ```
   cat id*.pub >> authorized_keys
   ```

   Bei Nutzern mit verschiedenen Homeverzeichnissen (lokales Filesystem und AFS) ist die Prozedur entsprechend für jedes Homeverzeichnis auf einem passenden Rechner zu wiederholen. In jedem `$HOME/.ssh/` sollte die Datei `authorized_keys` die public keys der Dateien `id*.pub` von allen Rechnern mit jeweils eigenem Homeverzeichnis enthalten, von denen aus der Zugang erlaubt sein soll. Dazu müssen diese Dateien evtl. mittels `scp` auf den jeweils anderen Rechner übertragen werden (temporär).

3. Da sich ein Teil der zu verwendenden Software unterhalb des Verzeichnisses `/afs/tu-chemnitz.de/project/sfb393/` befindet, verwenden wir zur Abkürzung eine Variable (etwa in `~/.cshrc` setzen):

   ```
   setenv SFB /afs/tu-chemnitz.de/project/sfb393
   ```

   bzw. (in `~/.bashrc`):

   ```
   export SFB=/afs/tu-chemnitz.de/project/sfb393
   ```

   Falls nicht bekannt, stellt man zunächst fest, welche Shell aktiv ist:

   ```
   echo $SHELL
   ```

[1] `authorized_keys` auf dem Zielrechner enthät public keys des Rechners, von dem man sich einloggt.

4. Das Hostfile (oder Machinefile) ist eine Datei, in der die Namen aller Rechner enthalten sind, mit denen man arbeiten möchte. Diese Datei wird ins eigene Verzeichnis kopiert:
   ```
   cp ${SFB}/pvm3/CASEHOSTS  ./
   ```
   und muss jeweils noch editiert werden, so dass der Rechner, auf dem man sich vom Arbeitsplatzrechner aus zuerst eingeloggt hat,
   - für PVM (oder MPICH) an letzter Stelle steht, bzw.
   - für Open-MPI an erster Stelle steht, sowie
   - die nicht benötigten Maschinen auskommentiert sind (mit `#` in Spalte 1).

Nutzung von PVM

1. **Verzeichnisse für PVM anlegen:**
   Das erledigt z.B. ein Shell-Skript, das folgendermaßen aufzurufen ist:
   ```
   ${SFB}/pvm3/initpvm
   ```
   Was passiert dabei (nur beim erstmaligen Aufruf)?
   - im Home-Verzeichnis wird ein Unterverzeichnis `pvm3` angelegt
     ```
     mkdir ~/pvm3
     ```
   - im Unterverzeichnis `pvm3` werden einige symbolische Links erzeugt
     ```
     cd ~/pvm3
     ln -s ${SFB}/pvm3/lib  ${SFB}/pvm3/include  .
     ```
   - im Verzeichnis `pvm3` wird das Unterverzeichnis `bin` angelegt und darin mindestens ein weiteres Unterverzeichnis (für Linux-Rechner ist das: `pvm3/bin/LINUX` oder `pvm3/bin/LINUX64`)
     ```
     mkdir  bin  bin/LINUX  bin/LINUX64
     ```
   Später (wenn die Verzeichnisse bereits existieren) wird `initpvm` nicht mehr benötigt.
2. **Umgebungsvariable:**
   Beim Login sollte automatisch die Variable `PVM_ROOT` auf das oben genannte Verzeichnis `$HOME/pvm3` gesetzt werden, also z.B. in die Datei `~/.cshrc` eintragen
   ```
   setenv PVM_ROOT $HOME/pvm3
   ```
   oder in die Datei `~/.bashrc`:
   ```
   export PVM_ROOT=$HOME/pvm3
   ```
3. **Aufbauen bzw. Konfigurieren des „Virtuellen Parallelrechners“**:
   Starten des PVM-Daemons (am besten in einem eigenen Terminalfenster)
   ```
   ${PVM_ROOT}/lib/pvm [<hostfile>]
   ```
   (bzw. `xterm -sb -rv -e  ...  &`).
   Einige mögliche Kommandos in der PVM-Console (mit Prompt-Zeichen `pvm>`):

`add` *name*	–	Hinzufügen eines weiteren Rechners zur Virtuellen Maschine
`delete` *name*	–	Entfernen eines Rechners aus der VM
`conf`	–	Anzeige der verfügbaren Rechner (innerhalb der VM)
`reset`	–	alle Prozesse beenden (z.B. bei Verklemmung)
`ps -a`	–	alle laufenden Prozesse anzeigen
`quit`	–	Console verlassen, Daemon läuft aber weiter (**mögl. vermeiden**)
`halt`	–	PVM auf allen beteiligten Rechnern beenden, Console verlassen
`help`	–	verrät, was es sonst noch für Kommandos gibt

   Man beachte den feinen Unterschied zwischen `quit` und `halt`. PVM sollte immer mittels `halt` beendet werden, sonst gibt es beim nächsten Start einige Schwierigkeiten.

   Das manuelle Hinzufügen weiterer Rechner zur VM kann man sich ersparen, wenn das Hostfile als Argument des `pvm`-Kommandos angegeben wird:
   ```
   ${PVM_ROOT}/lib/pvm CASEHOSTS
   ```

4. **Temporäre Dateien:**
   PVM legt jeweils zwei temporäre Dateien an:
   - `/tmp/pvmd.<uid>` – daran sollte man sich nur im äußersten Notfall vergreifen, (wenn sich der Daemon nicht starten lässt, weil er zuvor einmal nicht korrekt beendet wurde).
   - `/tmp/pvml.<uid>` – enthält Meldungen des Daemons und Ausgaben des Anwenderprogramms von Prozessen mit höherer Prozessnummer (das sind z. B. die durch Prozess 0 nachträglich gestarteten weiteren Prozesse).

5. **Übersetzen und Linken eines Programms für PVM:**
   Wir wollen dasselbe Programm in Verbindung mit PVM und MPI testen. Deshalb benutzen wir zur Kommunikation die Routinen aus unserer eigenen Standardbibliothek `libCubecom.a`.

   Wichtige Parameter werden als globale Struktur gespeichert. Um darauf zuzugreifen, muss im Fortran- bzw. C-Quelltext die entsprechende Include-Anweisung enthalten sein:

   `include 'include/trnet.inc'` bzw. `#include "trnet.h"`

   Nach der Initialisierung sind dort die eigene Prozessnummer $\in \{0, \ldots, 2^n - 1\}$ als ganzzahlige Variable `ICH`, die Cube-Dimension als Variable `NCUBE` sowie die Gesamtzahl der Prozesse als Variable `NPROC` verfügbar.

   Quelltext (Fortran oder C):
   Das Grundgerüst eines Hauptprogramms sollte dazu etwa folgendes enthalten:

```
PROGRAM mytest
include 'include/trnet.inc'
call trinit(0)
write(*,*) 'Hello World from node ',ICH
...
call trclose
END
```

   bzw. (für C-Hauptprogramme alternativ `trinitc`)

```
#include <stdio.h>
#include "trnet.h"
int main(int argc, char **argv)
{
   int zero=0;
   trinitc(&argc,&argv,&zero)
   printf("Hello World from node %d\n",ICH);
   ...
   trclose();
   return 0;
}
```

   Welche Anweisungen erforderlich sind, um direkt mit PVM- oder mit MPI-Routinen zu arbeiten, kann man in den Beispiel-Quelltexten des Unterprogramms `trinit` sehen.

   Solange im Quelltext aber keine direkten Aufrufe von PVM- oder MPI-Funktionen enthalten sind, ist das Programm für beide Systeme nutzbar. Ob es dann PVM oder MPI als Kommunikationsbibliothek nutzt, wird erst beim Linken des Programms durch Angabe der entsprechenden Bibliothek festgelegt.

   In der PVM-Version fragt das Unterprogramm `trinit` den Nutzer nach der gewünschten Hypercube-Dimension n und versucht dann, das Programm entsprechend oft (noch $(2^n - 1)$-mal) zu starten.

Die zusätzlich gestarteten Programme rufen ebenfalls `trinit` und erkennen dort, dass sie sich dem zuerst gestarteten „unterzuordnen“ haben, d. h. das Programm `trinit` arbeitet dort entsprechend anders (mittels `if - then - else`).

Nach dem Aufruf von `trinit` kennt jeder Prozess seine eigene Nummer und die Gesamtzahl der laufenden Prozesse. Der Parameter bei `trinit(0)` ist nahezu bedeutungslos. Mit `trinit(-1)` wird ein Teil der Bildschirm-Ausgaben dieses Unterprogramms unterdrückt. Eine analoge Routine (gleiche Rufzeile) gibt es dann auch für MPI.

Include-Dateien und Bibliotheken:

Vorbemerkung:

Schon wegen der Länge der Kommandozeilen ist die Verwendung eines Makefiles sehr zu empfehlen (s. S. 5 bzw. 7)!

Wie anfangs beschrieben, wird nachfolgend angenommen, dass die Variable

`${SFB}` den Pfad `"/afs/tu-chemniz.de/project/sfb393"`

enthält. Dann sollten im Arbeitsverzeichnis die folgenden Einstellungen vorgenommen werden.

für Fortran:

```
ln -s ${SFB}/FEM/libs/include ./
```

und (nur bei Bedarf):

```
ln -s ${PVM_ROOT}/include/fpvm3.h ./
```

Durch den ersten symbolischen Link werden die Hypercube-spezifischen Include-Files leichter erreichbar (da mancher Fortran-Compiler die Angabe solcher Verzeichnisse mittels Option `-I ...` ignoriert).

Der zweite symbolische Link ist nur erforderlich, wenn der eigene Quelltext PVM-spezifische Aufrufe / Variablen verwendet und deshalb die folgende Include-Anweisung enthalten muss.

```
include  'fpvm3.h'
```

(Nicht erforderlich bei ausschließlicher Verwendung der „Hypercube-Bibliothek“)

Zum Übersetzen der eigenen Quelltexte wird `gfortran` benutzt:

```
gfortran -c [-O2 -march=x86-64]  *.f
```

Ebenso zum Linken unter Angabe der zusätzlich benötigten Bibliotheken

```
gfortran -o <executable> *.o -L${SFB}/FEM/libs/LINUXX64/ \
         -lCubecom -lTools -lvbasmod -lfpvm3 -lpvm3
```

Als `<executable>` ist der Name des zu erzeugenden ausführbaren Programms anzugeben, z.B. `mytest_pvm`.

für C:

Man benötigt keine zusätzlichen symbolischen Links, sondern verwendet die Include-Option beim Compilieren:

```
gcc -c [-O2 -march=x86-64] -DUNDER -I${SFB}/FEM/libs/headers  *.c
```

Bei expliziter Verwendung von PVM-Funktionen im eigenen Quelltext ist zusätzlich die Compiler-Option `-I $PVM_ROOT/include` notwendig.

Durch `-DUNDER` werden die Namens-Konventionen zwischen C und Fortran angepasst. Dazu gehört in jedem C-Programm das Header-File `"names.Cube.h"`, welches durch

```
#include "trnet.h"
```

schon mit geladen wird. Damit wird bspw. gesichert, dass ein in Fortran geschriebenes Unterprogramm `"tree_down"` im C-Quelltext unter diesem Namen verwendet werden kann, obwohl es eigentlich `"tree_down_"` heißt (und bei anderen Fortran-Compilern auch mal `"tree_down__"`).

Zum Linken des C-Programms mit der (z.T. in Fortran geschriebenen) Hypercube-Bibliothek sollte wie oben `gfortran` benutzt werden (nicht `gcc`), also

```
gfortran -o  ...  *.o  ...
```

6. **Starten des Programms:**

   Der PVM-Daemon muss auf einer gewünschten Anzahl von Maschinen gestartet sein wie in Punkt 3 beschrieben.

   Das ausführbare Programm muss zum Starten zunächst ins PVM-Arbeitsverzeichnis kopiert

   ```
   cp mytest_pvm ${PVM_ROOT}/bin/LINUX64/
   ```

   oder als symbolischer Link dort angelegt werden:

   ```
   cd ${PVM_ROOT}/bin/LINUX64/
   ln -s $HOME/arbeitsverzeichnis/mytest_pvm ./
   ```

   Das Programm wird wie ein beliebiges Unix-Programm in einem Terminalfenster (**nicht** in der PVM-Console) gestartet:

   ```
   cd ${PVM_ROOT}/bin/LINUX64/
   ./mytest_pvm
   ```

   Dieses ist dann Prozess Nr. 0 und initiiert nach dem Start beim Aufruf des Unterprogramms `trinit` die gewünschte Zahl weiterer paralleler Prozesse mittels `pvmfspawn`. Diese Anzahl wird durch `trinit` von Prozess Nr. 0 durch Abfrage der Hypercubedimension bestimmt. Sie kann diese Information aber auch aus der Kommandozeile entnehmen, wenn das Programm mit Kommandozeilen-Argument gestartet wird:

   ```
   ./mytest_pvm -NCUBE=3
   ```

   (hier also auf $2^3 = 8$ Prozessoren).

7. **Empfehlung für ein (primitives) Makefile:**

```
SFB = /afs/tu-chemnitz.de/project/sfb393
FC  = gfortran
CC  = gcc
PROGRAM = mytest_pvm
LIBDIR  = $(SFB)/FEM/libs/LINUXX64
PARLIBS = -L$(LIBDIR) -lCubecom -lTools -lvbasmod -lfpvm3 -lpvm3
HEADRS  = $(SFB)/FEM/libs/headers
FFLAGS  = -O2 -march=x86-64 -ffast-math
CFLAGS  = -O2 -march=x86-64 -ffast-math -I$(HEADRS) -I.
CPPFLAGS= -DUNDER
OBJECTS = mytest.o mysubprog.o

$(PROGRAM): $(OBJECTS)
          $(FC) -o $@ $(OBJECTS) $(PARLIBS)

.f.o:
          $(FC) -c $(FFLAGS) $*.f
.c.o:
          $(CC) -c $(CPPFLAGS) $(CFLAGS) $*.o
clean:
          rm *.o
```

Für die Nutzung von (Open-)MPI sind nur wenige Änderungen erforderlich (s.S. 7).

Nutzung von OpenMPI

1. **Verzeichnisse für MPI** ...

   ... beschränken sich auf ein von allen Rechnern erreichbares Arbeitsverzeichnis. Wenn dazu ein **AFS-Verzeichnis** verwendet wird, sollte man daran denken, dass die anderen Maschinen nicht immer automatisch ein AFS-Token beziehen, um entsprechende Zugriffsrechte auf das Verzeichnis zu erhalten, also besser weitgehende Leserechte setzen:

   `fs sa -dir . -acl chemnitz rl`

   oder noch offener

   `fs sa -dir . -acl urz:anyuser rl`.

   Die erforderlichen MPI-Kommandos sollten im Suchpfad gefunden werden (siehe unten: Umgebungsvariablen, Punkt 2).

2. **Umgebungsvariable:**

   Die zum Compilieren bzw. Starten der Programme benötigten Befehle (u.a. `mpif77`, `mpicc`, `mpirun`) dürfen nicht mit gleichnamigen Befehlen anderer installierter MPI-Versionen verwechselt werden.

   Um sie nicht stets mit voller Pfadangabe aufrufen zu müssen, sollte der Suchpfad entsprechend angepasst werden:

   in `~/.cshrc`:

   ```
   if ( -d /usr/local/openmpi/bin ) then
      setenv MPIHOME /usr/local/openmpi
      set path = ( ${MPIHOME}/bin ${path} )
      setenv LD_LIBRARY_PATH ${MPIHOME}/lib:${LD_LIBRARY_PATH} )
   endif
   ```

   oder

   in `~/.bashrc`:

   ```
   if [ -d /usr/local/openmpi/bin ]
   then
      export MPIHOME=/usr/local/openmpi
      export PATH=${MPIHOME}/bin:${PATH}
      export LD_LIBRARY_PATH=${MPIHOME}/lib:${LD_LIBRARY_PATH}
   fi
   ```

   Beachte: Zuvor sollte man prüfen, ob die PATH-Variable etwa (durch Administrator-Vorgabe) das richtige Verzeichnis bereits enthält:

   `which mpirun`, bzw. `echo $LD_LIBRARY_PATH`.

3. **Übersetzen und Linken eines Programms für MPI:**

   Hier gibt es kaum Unterschiede zu den in Punkt 5 (S. 3) gegebenen Hinweisen.

   Durch Verwendung von `mpif77` (auch `mpif90`) bzw. `mpicc` als Compiler werden (fast) alle MPI-spezifischen Optionen und Standardbibliotheken automatisch benutzt.

   Im Makefile gibt es einige wenige Unterschiede gegegenüber der PVM-Variante (siehe S. 7).

4. **Starten des Programms:**

   Auf dem Rechner, der im Hostfile (Datei `CASEHOSTS`) als erster eingetragen ist, wird das Programm mit dem Kommando `mpirun` gestartet, wobei die Anzahl der Prozessoren in der Kommandozeile vorgegeben wird:

   ```
   mpirun -np 4 -hostfile CASEHOSTS mytest_mpi
   ```

   oder mit vollständigem Pfad für `mpirun`:

   ```
   ${MPIHOME}/bin/mpirun -np 4 -hostfile CASEHOSTS mytest_mpi
   ```

   Wird das Argument `-hostfile ...` (bzw. `-machinefile ...`) weggelassen, laufen alle Prozesse auf der aktuellen Maschine.

5. **Makefile für OpenMPI**

```
SFB = /afs/tu-chemnitz.de/project/sfb393
MPIHOME = /usr/local/openmpi
FC  = $(MPIHOME)/bin/mpif77
CC  = $(MPIHOME)/bin/mpicc
PROGRAM = mytest_mpi
LIBDIR  = $(SFB)/FEM/libs/OMPI64
PARLIBS = -L$(LIBDIR) -lMPIcubecom -lTools -lvbasmod
HEADRS  = $(SFB)/FEM/libs/headers
FFLAGS  = -O2
CFLAGS  = -O2 -I$(MPIHOME)/include -I$(HEADRS) -I.
CPPFLAGS= -DUNDER
OBJECTS = mytest.o mysubprog.o

$(PROGRAM): $(OBJECTS)
          $(FC) -o $@ $(OBJECTS) $(PARLIBS)

.f.o:
          $(FC) -c $(FFLAGS) $*.f
.c.o:
          $(CC) -c $(CPPFLAGS) $(CFLAGS) $*.o
clean:
          rm *.o
```

Weitere Hinweise

Eine ausführlichere Dokumentation der Unterprogramme der Bibliothek `libCubecom.a` bzw. `libMPIcubecom.a` sowie `libvbasmod.a` (UP für Vektoroperationen) gibt es unter
`http://archiv.tu-chemnitz.de/pub/2006/0042/data/sfb02-01.pdf`

Die PVM-Version von trinit (fast) im Original

```
C ===== trinit.f ---------------------------------------- 27.12.93 ====
C Variante fuer PVM Version 3.4 - Workstation-Cluster       M.Pester
C ================================================     - 26.01.01 -
C Modifikation vom 18.7.2000:
C   bei bis zu MAX_Procs Prozessoren wird versucht die Prozesse
C   reihum gleichmaessig auf die Maschinen zu verteilen, sonst
C   wie bisher alles auf einmal und nach Gutduenken von PVMD
C Falls jedoch Maschinen mit unterschiedlichen Datenformaten
C benutzt werden, hat die Auswahl der passenden Architektur
C Vorrang (Mischen von Pentium und HPPA geht hier nicht)
C ================================================
      SUBROUTINE TrInit(dummy)
C def. in trnet_m.inc :       PARAMETER (MAX_Procs=1024)
C Aufgabe: Belegung der Variablen NCUBE, ICH
C          Aufbau der logischen Verbindungen zu Nachbarn
C Modifikation vom 30.11.00:
C   Kommandozeilenargument -NPROC=##  oder  -NCUBE=##  zugelassen

      INCLUDE 'include/trnet.inc'
      INCLUDE 'trnet_m.inc'
      INCLUDE 'fpvm3.h'
      INTEGER  dummy
      CHARACTER HOST*16,ARCH*8,REQARCH*8,KEY*1,ARG1*16
      CHARACTER*16 hostnames(MAX_Procs)
C -- Programmname aus arg[0]:
      call getarg    ( 0, MyName )
      ARG1=' '
      call BaseName  ( MyName, L_Name )    ! ohne Pfad
C -- Pruefen ohne PVM-Fehlermeldung, ob pvmd aktiv ist:
      call pvmfsetopt(PvmAutoErr,0,Info)
      call pvmfmytid ( mytid )
      IF (mytid .LT. 0) THEN    ! PVM-Daemon nicht gefunden
        pvm_tids(0)=mytid
        write(*,90) MyName(1:L_Name)
        read (*,99) KEY
        IF (KEY .EQ. 'n' .OR. KEY .EQ. 'N') STOP
        ICH=0
        NPROC=1
        goto 20
      ELSE
C -- Nun PVM-Fehlermeldungen wieder aktivieren:
        call pvmfsetopt(PvmAutoErr,1,Info)
      ENDIF
90    Format(/' PVM scheint nicht aktiv zu sein. Soll das Programm ',
     +      A,' nur als Ein-Prozessor-Variante laufen ? - (j)/n - ',$)
99    Format(A1)
C ------------------------
C     Get cube information
C ------------------------
      call pvmfparent(id_par)            ! wie wurde ich gestartet?
      IF (id_par .LT. 0) then            ! durch Nutzer
        pvm_tids(0)=mytid
        write(*,100)
        call pvmfconfig(nhost,narch,idtid,HOST,REQARCH,ispeed,info)
        call pvmfsetopt(PvmRoute,PvmRouteDirect,Info)
        write(*,101) nhost,narch
        write(*,102) REQARCH,HOST,idtid,ispeed
        IF (nhost .gt. 1) THEN
          IF (nhost .GT. MAX_Procs .OR. nhost .EQ. 1) THEN
            DO i=2,nhost
              call pvmfconfig(nhost,narch,idtid,HOST,ARCH,ispeed,info)
              write(*,102) ARCH,HOST,idtid,ispeed
            ENDDO
          ELSE   ! Maschinennamen merken, um einzeln zu starten
            DO i=1,nhost
            call pvmfconfig(nhost,narch,idtid,hostnames(i),ARCH,
```

```
     +                 ispeed,info)
           if (i .LT. nhost)
     +           write(*,102) ARCH,hostnames(i),idtid,ispeed
           ENDDO
         ENDIF
       ENDIF
       IF (narch .GT. 1) write(*,109)
       if (L_Name .EQ. 0) then
         write(*,120)
         call flush(6)
         read(*,'(A)') MyName
         L_Name=Len_Trim(MyName)
120      Format(' Program name cannot be detected automatically.'/
     +          ' (may be you have no Fortran main program?) '/
     +          ' Please type in the name of your executable '/
     +          ' (without path): ',$)
       endif
109    Format(' ',24('*'),' WARNING ',24('*')/
     +        '     "Cubecom" does not support different data formats!'/
     +        '     I will do my very best !'/
     +        ' ',57('*'))
100    Format(' Hypercube init for PVM.')
101    Format(' PVM daemon found on',I3,' host(s) of',i3,
     +        ' architecture types(s).')
102    Format(4x,A8,' on host ',A,' - pvmd =',z8,', speed:',I8)
103    Format(' Launching program: ',A )
107    Format(' Enter cube dimension: ',$)
       N_arg=IARGC()
       I_arg=1
       NCUBE=-11
       write(*,103) MyName(1:L_Name)
       IF (N_arg .gt. 0) then
         DO WHILE (I_arg .LE. N_arg .AND. NCUBE .LT. 0)
200        CONTINUE
           call getarg ( I_arg, ARG1 )
           L_arg1 = Len_Trim(ARG1)
           I_arg=I_arg+1
           if (L_arg1 .GT. 7) then
             call UpCase(ARG1)
             if (ARG1(1:7) .EQ. '-NCUBE=') then
               read(ARG1(8:L_arg1),*,ERR=200) NCUBE
             else if (ARG1(1:7) .EQ. '-NPROC=') then
               read(ARG1(8:L_arg1),*,ERR=200) NPROC
               NCUBE=0
               DO WHILE (NPROC .GT. 1)
                 NPROC=NPROC/2
                 NCUBE=NCUBE+1
               ENDDO
             else
               goto 200
             endif
           endif
         ENDDO
       ENDIF
       IF (NCUBE .EQ. -11) then
         write(*,107)
         read(*,*) NCUBE
       ENDIF
       NPROC=ISHFT(1,NCUBE)
       IF (NCUBE .GT. 0) THEN
         if (narch .EQ. 1) then     ! belieb. Hosttyp mit gleicher Arithm.
           IF (nhost .GT. MAX_Procs .or. nhost .LE. 2) THEN
C            ! Verteilung der Prozesse durch PVM
             call pvmfspawn(MyName(1:L_Name),PVMDEFAULT,'*',NPROC-1,
     +                      pvm_tids(1),info)
           ELSE  ! Verteilung der Prozesse selbst durchfuehren (einzeln)
             k=0
             mist=0
```

```
              ngot=1
              DO i=1,NPROC-1
111             k=k+1
                if (k .gt. nhost) k=1
                call pvmfspawn(MyName(1:L_Name),PVMHOST,hostnames(k),1,
     +                         pvm_tids(i),info)
                if (info .eq. 1) then
                  ngot=ngot+1
                  info=ngot
                  write(*,205) i,hostnames(k)
205               Format(' -> ICH=',I4,'  on host: ',A)
                else
                  mist=mist+1
                  write(*,105) hostnames(k)
105               Format(' spawn failed on host ',A)
                  info=ngot
                  if (mist .GE. NPROC) goto 130
                  goto 111
                endif
              ENDDO
            ENDIF
          else                  ! gleiche Architektur erzwingen
            call pvmfspawn(MyName(1:L_Name),PVMARCH,REQARCH,NPROC-1,
     +                   pvm_tids(1),info)
          endif
130       NP=info+1
          IF (NP .GT. 1) write(*,106) NPROC
106       Format(' Running: ',I5,' processes.')
          IF (NP .LT. NPROC) THEN
            write(*,104) NPROC-NP,NPROC ! (pvm_tids(i),i=0,NPROC-1)
104         Format(' Fehler bei PVM_SPAWN:',I4,' von ',
     +         i4,' Proz. nicht erfolgreich gestartet.')
            NCUBE=0
            DO WHILE (ISHFT(1,NCUBE+1) .LE. NP)
              NCUBE=NCUBE+1
            ENDDO
            NP1=ISHFT(1,NCUBE)
            k=0
            DO i=0,NPROC-1
              if (pvm_tids(i) .gt. 0) then
                k=k+1
                if (k .gt. NP1) then
                  write(*,'(2i3,z8)') k,i,pvm_tids(i)
                  call pvmfkill(pvm_tids(i),info)
                else
                  pvm_tids(k-1)=pvm_tids(i)
                endif
              endif
            ENDDO
            NPROC=ISHFT(1,NCUBE)
            write (*,201) NCUBE,NPROC
201         Format(' Fortsetzen mit NCUBE=',I3,', NPROC=',I3,
     +           ' ? - (j)/n ',$)
            read(*,99) KEY
            IF (KEY .EQ. 'n' .OR. KEY .EQ. 'N') then
              DO i=1,NPROC-1
                call pvmfkill(pvm_tids(i),info)
              ENDDO
              call TrClose
              STOP
            ENDIF
          ENDIF      ! (Fehlerbehandlung)
C
          call pvmfinitsend( PVMDEFAULT, info )
          call pvmfpack( INTEGER4, NCUBE, 1, 1, info )
          call pvmfpack( INTEGER4, NPROC, 1, 1, info )
          call pvmfpack( INTEGER4, pvm_tids, NPROC, 1, info )
          call pvmfmcast( NPROC-1, pvm_tids(1), 0, info )
```

```
          ICH=0
        ELSE IF (NCUBE .LT. 0) THEN
          call TrClose
          STOP
        ENDIF
      else
        call pvmfrecv( id_par, 0, info )      ! recv from parent
        call pvmfunpack( INTEGER4, NCUBE, 1, 1, info )
        call pvmfunpack( INTEGER4, NPROC, 1, 1, info )
          call pvmfunpack( INTEGER4, pvm_tids, NPROC, 1, info )
          do 10 i=1, NPROC-1
           if( mytid .eq. pvm_tids(i) ) ICH = i
10      continue
        endif
20      NCUBE_limit=NCUBE
      call Cube_Dim(NCUBE)

      if (ICH .EQ. 0) call ListPVMOPTS
      return
      end



      Subroutine BaseName(Name,l)
      Character*(*) Name
      l=Len(Name)
      lo=l
      DO WHILE(Name(lo:lo) .EQ. ' ' .AND. lo .GT. 0)
        lo=lo-1
      ENDDO
      lu=lo
      DO WHILE(Name(lu:lu) .NE. '/' .AND. lu .GT. 0)
        lu=lu-1
      ENDDO
      l=lo-lu
      lu=lu+1
      Name=Name(lu:lo)
      return
      END



      subroutine ListPVMopts
      include 'fpvm3.h'
      include 'trnet_m.inc'

      write(*,*) 'PVM options listing:'
      write(*,90)
      IF (pvm_tids(0) .LT. 0) THEN
        write(*,*) ' PVM is not running.'
        goto 100
      ENDIF
10    Format(' Routing policy: ',T30,A)
      call pvmfgetopt(PvmRoute,id)
      IF (id .EQ. PvmDontRoute) then
        write(*,10) 'DontRoute'
      ELSE IF(id .EQ. PvmAllowDirect) THEN
        write(*,10) 'AllowDirect'
      ELSE IF(id .EQ. PvmRouteDirect) THEN
        write(*,10) 'RouteDirect'
      ENDIF

      ...

      return
      end
```

Die MPI-Version von trinit (fast) im Original

```
C ===== trinit.f ---------------------------------------- 07.05.97 ====
C Variante f. MPI
C ======================================
        SUBROUTINE TrInit(noprint)
C Aufgabe: Belegung der Variablen NCUBE, ICH
C          Aufbau der logischen Verbindungen zu Nachbarn
C Common-Block (in 'trnet_m.inc') enthaelt Character-Variable MyName,

        INTEGER  MSGTYPE
        INCLUDE 'include/trnet.inc'
        INCLUDE 'trnet_m.inc'
        INCLUDE 'mpif.h'
        character*40 mpihostname
        DATA MSGTYPE /9999/

        CALL mpi_init (ierr)
        CALL mpi_comm_rank (MPI_COMM_WORLD, ICH, ierr)
        CALL getarg    ( 0, MyName )
        CALL BaseName  ( MyName, L_Name )
#ifdef LAMMPI
C Bei LAM-MPI "invalid argument":  Originalroutine ist fehlerhaft!
          CALL mp_get_hostname(mpihostname,lnghost,ierr)
#else
          CALL mpi_get_processor_name(mpihostname,lnghost,ierr)
#endif
        IF (noprint .EQ. 0) THEN
          write(*,1000) MyName(1:L_Name), ICH, mpihostname(1:lnghost)
1000      Format(' Running ... ',A,' ICH=',I3,' on host ',A)
        ENDIF
        CALL mpi_barrier(MPI_COMM_WORLD,IERR)
        CALL mpi_comm_size (MPI_COMM_WORLD, NCPUS, ierr)
        IF (ICH .EQ. 0) write(*,*) 'Prozessoren:',NCPUS

        NCUBE=int(log(float(ncpus))/log(2.0)+0.1)
        NPROC=NCPUS
        NCUBE_Limit=NCUBE

        IF (ICH .EQ. 0) THEN
          NTYPES=1
          write(*,100) NCPUS,MyName(1:L_Name),ncube
100          format(' MPI verfuegt ueber',I3,' Prozessoren, '/
     +                 ' Programm ',a,' gestartet, ',
     +                 ' Cube-Dimension : ',I3)
          IF (NCUBE .LT. 0) STOP
        ENDIF
        CALL Cube_Dim(NCUBE)        ! Lforw, Lback werden belegt.

        RETURN
        END
```