Einführung in PVM

Übungen zur Vorlesung
„Parallele und verteilte Algorithmen“

Philipps-Universität Marburg
Sommersemester 2002

Das PVM System

- PVM steht für
  Parallele Virtuelle Maschine
- Netzwerk von Unix-Workstations als „virtuelle“ parallele Maschine
- dazu:
  - Dämon-Prozess auf jeder Workstation
  - Bibliotheken mit C- bzw. Fortran-Routinen zur Message-Passing Programmierung in allgemeiner MIMD-Form

PVM-Anwendungen

- mehrere Komponenten (Prozesse/Tasks)
- Ein Prozess ist ein sequentielles C- oder Fortran-Programm mit Aufrufen von PVM-Routinen.
- Jeder Prozess hat eine eigene taskid.
- Interaktion durch Nachrichtenaustausch
- manuelle Arbeitsverteilung auf Prozesse
- automatische oder explizite Platzierung von Prozessen

Beispielprogramm: hello.c

```
#include <stdio.h>
#include "pvm3.h"
main()
{    int cc, tid;
     char buf[100];
     printf("i'm t%x\n", pvm_mytid());
     cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);
     if (cc == 1) {
          cc = pvm_recv(-1, -1);
          pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
          pvm_upkstr(buf);
          printf("from t%x: %s\n", tid, buf);
     } else
          printf("can't start hello_other\n");
     pvm_exit();  exit(0);
}
```

header file laden

Taskid abfragen/ Prozess anmelden

Nach-richten-empfang (Details später)

Ein Programm „hello_other“ auf anderem Rechner starten

Abmelden

Beispielprogramm: hello_other.c

```
#include "pvm3.h"

main()
{
    int ptid;
    char buf[100];
    ptid = pvm_parent();
    strcpy(buf, "hello, world from ");
    gethostname(buf + strlen(buf), 64);
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, 1);
    pvm_exit();
    exit(0);
}
```

Taskid des Elternprozesses abfragen/ Prozess anmelden

Nach-richten-versand (Details später)

Abmelden

Prozessverwaltung

- tid pvm_mytid()
  - Abfrage der Taskid
  - falls dies erster PVM-Routinenaufruf, meldet sich der Prozess 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 mit Taskid tid
- tid pvm_parent()
  - Abfrage der Taskid des Elternprozesses, d.h. des Prozesses, der den aufrufenden Prozess gestartet hat

Starten neuer Prozesse

numt pvm_spawn(*task, **argv, flag, *where, ntask, *tids)

- startet ntask neue PVM-Prozesse, die jeweils das Programm task mit Argumenten argv ausführen
- Das Feld tid enthält die Taskids der neuen Prozesse
- flag und where bestimmen, wo die Prozesse ausgeführt werden; einige Optionen:

flag		Bedeutung
PvmTaskDefault	0	don‘t care where
PvmTaskHost	1	„where“ gibt Zielrechner an
PvmTaskArch	2	„where“ gibt Zielarchitektur an

- Der Rückgabewert numt liefert die Anzahl der tatsächlich gestarteten Prozesse.

Kommunikation: Senden

- Initialisierung des Sendepuffers und Festlegung der Nachrichtenkodierung (meist: PvmDataDefault)
  - bufid pvm_initsend(PvmDataDefault)
- Packen der zu versendenden Daten in den Sendepuffer
  - pvm_pack() Familie von Routinen, etwa:
    - info pvm_packf (printf-like format)
    - info pvm_pkbyte (*cp, count, stride)
    - info pvm_pkint (*np, count, stride)
    - ... (siehe Manual)
    - info pvm_pkstr (*cp)
- Senden der Daten
  - info pvm_send (tid, messagetag)
  - info pvm_mcast (*tids, ntask, messagetag)
- Packen und Senden in einem
  - info pvm_psend(tid, messagetag, *vp, count, type)

Kommunikation: Empfangen

- Empfang einer Nachricht
  - blockierend: bufid pvm_recv(tid, msgtag)
  - nicht-blockierend: bufid pvm_nrecv(tid, msgtag)
  - mit Timeout: bufid pvm_trecv(tid, msgtag, & tmout)
- Analyse einer empfangenen Nachricht
  - info pvm_bufinfo(bufid, *bytes, *msgtag, *tid)
- Entpacken einer Nachricht
  - mit Integer-Werten: info pvm_upkint(&pointer, number, stride)
  - mit Double-Werten: info pvm_upkdouble(&pointer, number, stride)
  - mit einem String: info pvm_upkstr(&pointer)
  - … (siehe Manual)
- Test auf Nachrichteneingang (nicht blockierend, kein Nachrichtenempfang)
  - bufid pvm_probe(tid, msgtag)
- Empfangen und Entpacken in einem
  - info pvm_precv(tid, messagetag, *vp, count, type, *rtid, *rtag, *rlen)

tid bzw. msgtag gleich –1 erlaubt beliebige Werte beim Empfang.

PVM am Fachbereich

- Grundeinstellungen (in .tcshrc hinzufügen):

```
setenv PVM_ROOT /app/lang/parallel/pvm3
setenv PVM_ARCH `$PVM_ROOT/lib/pvmgetarch`
setenv PVM_DPATH $PVM_ROOT/lib/pvmd
setenv PATH ${PATH}:$PVM_ROOT/lib
alias pvm $PVM_ROOT/lib/pvm
```

Kompilieren und Starten von Programmen

- Lokale Kopie des Header-Files erstellen:
  - cp $PVM_ROOT/include/pvm3.h .
- Aufruf des C-Compilers:
  - cc –L$PVM_ROOT/lib/LINUX hello.c –lpvm3 –o hello oder
  - cc –o hello hello.c $PVM_ROOT/lib/LINUX/libpvm3.a

  Für jedes Programm/jeden Prozess muss eine Kompilation erfolgen.
  Im Beispiel müssen hello.c und hello_other.c kompiliert werden.

- Die ausführbaren Dateien müssen in das Verzeichnis
  $PVM_ROOT/bin/LINUX
  kopiert werden.

- Vor dem Aufruf der Programme muss zunächst der PVM-Dämon mit pvm oder pvmd gestartet werden.

Starten von PVM

- Starten des Dämons mit Rechnerliste
  - pvmd hostlist
  - Beispiel für eine hostlist:

```
tripoli
lome
kampala
tunis
lusaka
rabat
asmara
algiers
bangui
brazzaville
```

- Starten der PVM-Konsole
  - pvm

  Falls noch nicht geschehen, wird der Dämon gestartet;
  die Angabe einer Hostlist ist ebenfalls möglich

Die PVM-Konsole ermöglicht eine Kontrolle aller Knoten der PVM sowie aller PVM-Prozesse.

PVM Konsole

- Aufruf durch pvm
- wichtigste Kommandos:
  - help [command] - Hilfe zu Kommandos
  - quit - Verlassen der Konsole (PVM Dämon bleibt aktiv)
  - halt - Alle PVM-Dämonen werden gestoppt und die Konsole verlassen
  - conf - Auflistung der Rechner in virtueller Maschine
  - add host(s) - Hinzunahme von Rechnern zu virt. Maschine
  - delete host(s) - Entfernen von Rechnern
  - ps [-a] - Auflistung aller aktiven PVM-Prozesse
  - spawn file - Start eines Prozesses
  - kill tid - Terminierung eines Prozesses
  - reset - Alle Prozesse löschen und PVM zurücksetzen