- Paralleles Rechnen II:
- PVM-Programmierung
-

- Markus Dahms
- (dahms@fh-brandenburg.de)
- 27.10.2004

Agenda

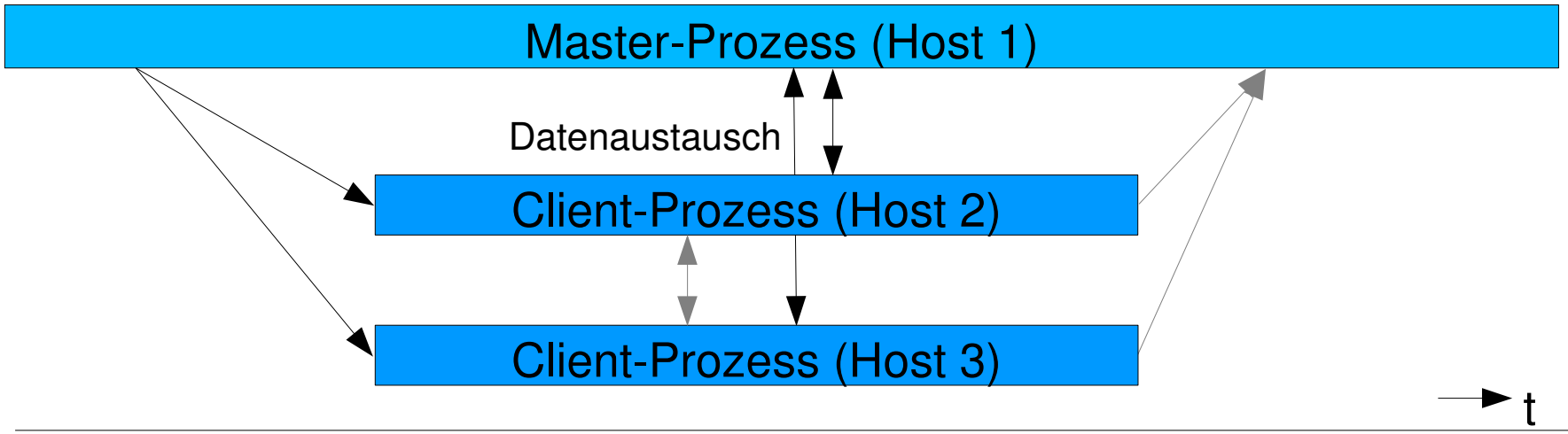
- PVM allgemein
- Grundstruktur paralleles Programm
- Einführung PVM-API
  - Initialisierung & Fehlerbehandlung
  - Prozesse
  - Datenaustausch
  - Erstellen und Ausführen des Programms
- Weitere Informationen

PVM allgemein

- PVM: Parallel Virtual Machine
- Funktionsbibliothek, Daemon und Konsole
- Programmiersprachen
  - Standarddistribution: C und Fortran
  - externe Erweiterungen für Perl, Python, C++, TCL, Java, Common Lisp, IDL, S-Lang, Matlab und R ...
- Alternativen: MPI und andere..

Struktur paralleles Programm

- Master startet Clients (“spawn”)
- Rechenarbeit & Datenaustausch
- Clients beenden sich, Master hat Ergebnis



PVM-API: Initialisierung

- Einbinden der Header-Datei

```
#include <pvm3.h>
```

- eigene TID & Parent-TID (Task IDs)

```
int mytid, parent_tid;

mytid = pvm_mytid();
parent_tid = pvm_parent();
```

- Master oder Client?

```
if ((parent_tid == PvmNoParent) ||
    (parent_tid == PvmParentNotSet)) { /* Master */ }
else { /* Client */ }
```

PVM-API: Fehlerbehandlung

- Rückgabewert < 0 bedeutet Fehler
- `pvm_perror()` gibt Fehlermeldung aus

```
int  info;

info  =  pvm_exit();  /* oder fast jede andere
                         PVM-Funktion  */

if(info <  0)
{
   pvm_perror("Fehler:  ");
   pvm_exit();
   return  EXIT_FAILURE;
}
```

PVM-API: Prozesse

- Master: Erstellung eines neuen Prozesses

```
int info, child_tid;

info = pvm_spawn(argv[0], NULL, PvmTaskDefault,
NULL,
                 1, &child_tid);
```

- Master: Beenden eines Child-Prozesses

```
info = pvm_kill(child_tid);
```

- Client beenden: Programm beenden

- Aufräumen

```
info = pvm_exit();
```

PVM-API: Benachrichtigungen

- Master fordert Benachrichtigungen über Zustandsänderungen des Clients an
  - ein Client hat sich beendet: `PvmTaskExit`
  - ein Host wurde gelöscht: `PvmHostDelete`

```
#define  MSGTAG_EXITED  0x0001

info = pvm_notify (PvmTaskExit, MSGTAG_EXITED, 1,
                   &child_tid );
```

- Master erhält Nachricht (siehe `pvm_recv`)

PVM-API: Datenaustausch

- Senden von Daten

```
#define MSGTAG MYDATA 0x0123

int bufid, info, i = 12345;
char *str;

str = "Dies ist ein Text";

/* Fehlerbehandlung nicht vergessen ;)
   (Rueckgabewerte der PVM-Funktionen auswerten)
*/

bufid = pvm_initsend(PvmDataDefault);
info = pvm_pkint(&i, 1, 1);
info = pvm_pkstr(str);
info = pvm_send(pvm_parent(), MSGTAG_MYDATA);
```

PVM-API: Datenaustausch

- Empfangen von Daten
  - nichtblockierende Version `pvm_nrecv()`

```
int  bufi d, in fo, i ;
cha r  str [2048 ];

/*  Fehle rbeha ndlun g  nic ht  ve rgess en  ;)
    (Ruec kgabe werte  der  PVM-F unkti onen  auswe rten)
*/

buf id =  pvm_r ecv (-1 , MS GTAG_ MYDAT A);
inf o =  pvm_up kint (&i,  1,  1);
inf o =  pvm_up kstr (str);

pri ntf( "%d: % s\n", i, s tr);
```

Erstellen und Ausführen

- Erstellung eines lauffähigen Programms

```
$ gcc -o  program program.c -lpvm3
```

- Ausführen des Programms
  - erst PVM-Dämon starten
  - absoluten Pfadnamen verwenden, wenn `argv[0]` als Pfad für die Client-Prozesse genutzt wird

```
$ /cluster/bin/program  -x -y  -z
```

Weitere Informationen

- PVM im Internet:
  - Homepage:
    http://www.csm.ornl.gov/pvm/pvm_home.html
  - PVM Book:
    http://www.netlib.org/pvm3/book/pvm-book.html
  - Man Pages online:
    http://www.csm.ornl.gov/pvm/man/manpages.html
- Fragen?