Kurzuebersicht Chameleon

Institut fuer Wissenschaftliches Rechnen
Nils Schmeisser, 90/07/01, November 1994

0. Einrichtung

.profile ergaenzen zu:

```
export TOOLSDIR=/usr/local2/gsys/cham/tools.core
export P4DIR=/usr/local2/gsys/SLES/p4/p4-1.3/lib
export TOOLSHOSTS=/usr/local2/gsys/cham/tools.core/comm
export TOOLSARCH=rs6000
```

1. Starten einer Applikation

```
<Executable> -np <Anzahl Prozessoren>
optional:    -pihosts <Maschinenname>[<Namenliste>]
```

2. Programmierung

2.1. Grundgeruest

```
/****** Chameleon includes ******/

#include "tools.h"
#include "comm/comm.h"

/*******************************/

int worker(int argc,char ** argv) {
  .
  .
  .
  return status;
}

main(int argc,char **argv) {
  PICall(worker,argc,argv);
}
```

int PICall(int (*worker)(),int argc,char **argv)	Startet den parallelen Abschnitt des Programms.
int PImytid	Haelt die Prozessor ID Nummer.
int PInumtids	Haelt die Anzahl der Prozessoren.

2.2. Kommunikation

2.2.1. PE <-> PE

a) "blocking" Operationen:

Bsp.:

```
int n;
if (PImytid==0) {
  scanf("%d",&n);
  PIbsend(0,&n,sizeof(int),1,MSG_INT);
} else
  PIbrecv(0,&n,sizeof(int),MSG_INT);

/* PE 0 liest n ein und sendet es an PE 1 */
```

PIbsend(int tag,void *buffer,int length,int destination int data_type)	Sendet *length* Bytes aus dem Puffer *buffer* an Prozessor *destination* und fuegt der Nachricht den Tag *tag* zu.
PIbrecv(int tag,void *buffer,int maxlength,int data_type)	Empfaengt Daten, die mit dem Tag *tag* versehen sind, im Puffer *buffer*. Der Empfang wird nach *maxlength* Byte abgebrochen.

b) "non-blocking" Operation:

Bsp.:

```
PISendId_t id1,id2;
...
PInsend(...,id1); /* Start von Sendung 1 */
...
PInsend(...,id2); /* Start von Sendung 2 */
...
PIwsend(...,id1); /* Beenden von Sendung 1 */
...
PIwsend(...,id2); /* Beenden von Sendung 2 */
```

PInsend(int tag,void *buffer,int length,int destination int data_type,PISendId_t id)	Startet das Senden von *length* Byte aus dem Puffer *buffer* an Prozessor *destination* und fuegt der Nachricht den Tag *tag* zu. Die Sendung wird mit *id* identifiziert.
PIwsend(int tag,void *buffer,int length,int destination int data_type,PISendId_t id)	Beendet den, mit *id* identifizierten Sendevorgang
PInrecv(int tag,void *buffer,int maxlength,int data_type, PIRecvId_t id)	Empfaengt Daten, die mit dem Tag *tag* versehen sind, im Puffer *buffer*. Der Empfang wird nach *maxlength* Byte abgebrochen.
PIwrecv(int tag,void *buffer,int maxlength,int data_type, PIRecvId_t id)	Beendet den, mit *id* identifizierten Empfangsvorgang

c) "Test" Operationen:

Bsp.:

```
do {
  do_something();
} while (!(PInprobe(tag)));
PIbrecv(tag,buffer,maxlength,datatype);
```

PInprobe(int tag)	Testet, ob eine Nachricht mit dem Tag *tag* eingetroffen ist.
PIbprobe(int tag)	Wartet auf das Eintreffen einer Nachricht mit dem Tag *tag*.

2.2.2. Broadcasts

Bsp.:

```
int n;
if (PImytid==0) {
  scanf("%d",&n);
  PIbcast(&n,sizeof(int),1,PSAllProcs,MSG_INT);
} else
  PIbcast(&n,sizeof(int),0,PSAllProcs,MSG_INT);
/* PE 0 liest n ein und sendet es an alle PE's */
```

PIbcast(void *buffer,int size,int issrc, ProcSet *procset,int datatype)	Broadcast an alle, in der Menge *procset* enthaltenen Prozessoren. Nur der Sender setzt *issrc* auf 1, alle anderen setzte *issrc* auf 0.

2.2.3. Gathering

```
Bsp.: float lbuf,f(int,float);
      int lsize=sizeof(float);
      int i,*gsizes=(int*)malloc(sizeof(int)*PInumtids);
      float *gbuf=(float*)malloc(lsize*PInumtids);

      for (i=0;i<PInumtids;i++) gsizes[i]=lsize;
      lbuf=f(PImytid,0);
      PIgcolx(&lbuf,gsizes,gbuf,PSAllProcs,MSG_FLT);

      /* gbuf enthaelt nach dem Gathering auf allen Prozessoren der
         Menge PSAllProcs einen Vektor von Funktionswerten f(i,0) */
```

PIgcolx(void *localdata,int *globalsizes, void *globaldata,ProcSet *procset,int datatype)	Sammelt auf allen Prozessoren aus *procset* die lokalen Daten *localdata* von allen, in *procset* angegebenen Prozessoren im Puffer *globaldata*. Die Groessen aller lokalen Daten sind bekannt und werden vorher im Feld *globalsizes* eingetragen. Die Daten von Prozessor i stehen an i-ter Stelle im Puffer *globaldata*.
PIgcol(void *localdata,int localsize,void *globaldata, int globalsize,int *numglobals,ProcSet procset, int datatype)	Analogon zu *PIgcolx*, aber mit unbekannter Groesse aller lokalen Daten. *localsize* enthaelt diese Information, *globalsize* gibt die **Gesamtgroesse** des Empfangspuffers *globaldata* an. In *numglobals* wird die Anzahl der empfangenen Bytes zurueckgegeben.

2.3. Synchronisation von Prozesseinheiten

PIgsync(ProcSet *procset)	Synchronisiert alle Prozessoren aus *procset*.

2.4. Processor Sets

```
      ProcSet *PSAllProcs;
      int i;

      PSAllProcs=PSCreate(0);
      for (i=0;i<PInumtids;i++) PSAddMember(PSAllProcs,&i,1);
      PSCompile(PSAllProcs);

Bsp.: int SetName(int procid) {
        if (procid!=2) return procid % 2; else return -1;
      }
      ...
      ProcSet *set;
      set=PSPartition(SetName(PImytid),PSAllProcs);
```

Dieses Programmfragment erzeugt waehrend der Laufzeit zwei disjunkte Mengen *set* mit den Namen 0 und 1, wobei 0 die geradzahlig numerierten PE's ausser PE 2 und 1 die ungeradzahligen PE's enthaelt.

ProcSet *PSCreate(int name)	Erzeugt eine leere Prozessormenge mit dem "Namen" *name*.
PSAddMember(ProcSet *procset,int *members, int nummembers)	Fuegt der Menge *procset* die *nummembers* Prozessoren zu, die im Feld *members* angegeben sind.
PSCompile(ProcSet *procset)	Gibt die Prozessormenge *procset* zur Verwendung frei.
PSDestroy(ProcSet *procset)	Zerstoert die Prozessormenge *procset*.
ProcSet *PSPartition(int name,ProcSet *procset)	Erzeugt auf jedem Prozessor eine Instanz vom Typ ProcSet und ordnet den jeweiligen Prozessor aus der Menge *procset* der Menge mit dem Namen *name* zu.