Batch-Betrieb auf dem SPP-System

- ❑ Es stehen i.A. 2 Subcomplexe für parallele Jobs zur Verfügung(gsm8, gsm32)
- ❑ Die nqs-queues lauten **gsm8** und **gsm32**.
- ❑ Alle Batch-Aufträge werden auf einen der Prozessoren im Subcomplex gelenkt.
  ➡ Der "master"-Prozeß muß (!) als letzter Prozeß terminieren.
- ❑ Absetzen eines Auftrags: **`qsub batchfile`**
- ❑ Abfrage eines Jobs in der Warteschlange: **`qstat -a gsm8`**
- ❑ Abfrage eines laufenden Jobs: **`qstat -a gsm8_b`**
- ❑ Abbruch eine Batch-Jobs: **`qmgr abort request REQUEST ID`**

Aufbau eines PVM3-Batch-files

```
#!/bin/csh

# @$-q gsm32
# @$
# Löschen von "Altlasten"
/usr/lib/pvm/pvm_clean
# Starten der Virtuellen Maschine
/usr/lib/pvm/pvmd ~/pvm3/hostfile &
sleep 20
cd ~/pvm3/sc_spawn

# Starten des Group-Servers
pvmgs &
sleep 10
# Starten der Applikation
sc_spawn
# Terminieren verbliebender Prozesse
echo reset | /usr/lib/pvm/pvm
sleep 10
# Terminierung der Virtuellen Maschine
echo halt | /usr/lib/pvm/pvm
# Aufräumen
/usr/lib/pvm/pvm_clean ; rm core
```

Eigenschaften von MP-Bibliotheken

❏ Statische Prozeßverteilung (z.B. durch Eintrag in eine Konfigurationsdatei)

- Vorteile:
  - Effizienter Broadcast durch Aufbau einer Baumstruktur (O(log n))
  - Keine zusätzlichen Deamons erforderlich
- Nachteil:
  - Unflexibel
  - Nachstarten von Prozessen nicht möglich
  - Prozeßstart (meist über remsh) sehr zeitaufwending, da Authentisierung notwendig.
  - Keine gesichterte asynchrone Kommunikation möglich, da kein lokaler Daemon vorhanden ist, der die Botschaft zwischenspeichert und danach für eine sichere Übertragung sorgt.
- Vertreter:
  - TCGMSG (Argonne National Laboratory)

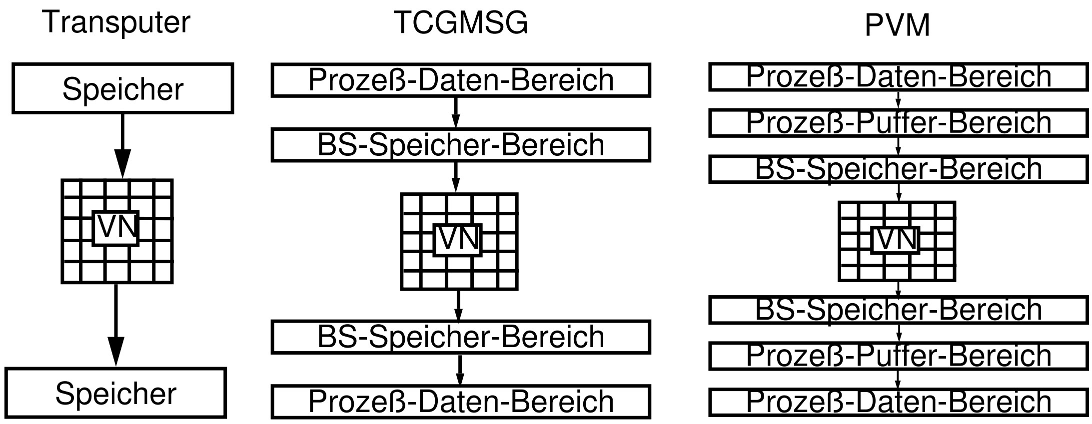
Eigenschaften von MP-Bibliotheken

❏ Dynamische Prozeßverteilung

- Vorteile:
  - Flexibel
  - Prozeßkontrolle über einen Satz von Deamons möglich
  - Startupzeit für Prozesse sehr gering (1 Kommunikation + 1 fork())
  - Gesicherte Asynchrone Kommunikation möglich
  - Fehlererkennung/Fehlermeldung/Fehlertoleranzmaßnahmen möglich
- Nachteile:
  - Meist zusätzliche Deamons erforderlich
  - Aufwendiger Braodcast (O(n))
- Vertreter:
  - PVM (Oak Ridge National Laboratory)
  - Message Passing Interface (MPI) U.S. Department of Energy
- **Informationen zu PVM + MPI sind mit dem Programm xnetlib erhältlich !**

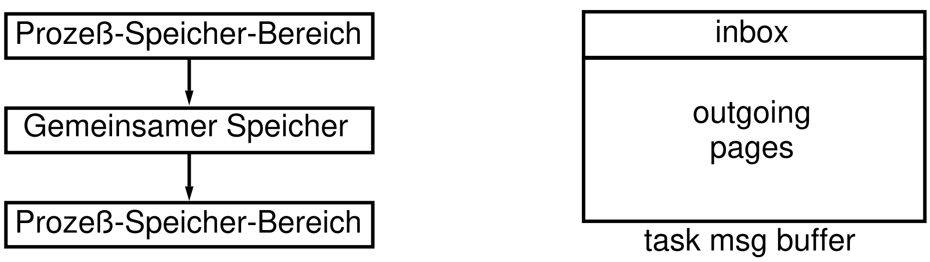
Eigenschaften von MP-Bibliotheken

- Unterstützung heterogener Systeme
  - Automatische Konvertierung der Datenformate (PVM, TCGMSG)
    z.B. Convex Native Floating Point ⟺ IEEE
    Hinweis: CS-Tools der Firma MEIKO benötigen expilizite xdr-Kodierung
- Unterstützung synchroner Kommunikation mit Daten eines einzigen Typs ohne Kodierung/Dekodierung und Zwischenpufferung möglich



Eigenschaften von MP-Bibliotheken

- Unterstützung von asynchroner Kommunikation
  - Vorteil: Teilweises Verbergen von Lastungleichgewicht (Sender muß nicht auf Empfänger warten, der Empfänger jedoch auf den Sender.)
- Unterstützung von Shared Memory Architekturen
  - Vorteil: Geringe Setupzeit und hoher Durchsatz durch Umgehung des Betriebssystems
  - Beispiel: TCGMSG auf KSR2und Sequent Symmetry
    PVM3 auf Convex SPP und KSR2



Schlußbemerkung

❏ Programmierung paralleler Systeme ist nicht einfach.

❏ Strukturierung bewirkt eine Fehlerreduzierung.

❏ Fehlerabfrage bewirkt eine Fehlerlokalisation.

❏ Fehlerquelle Nr. 1 sind Zeiger-Fehler

"Gehört zu jedem Zeiger auch ein Speicherbereich?"

❏ Fehlerquelle Nr. 2 sind Verklemmungen

"Können zur Laufzeit unauflösbare Abhängigkeiten entstehen?"

❏ ...

❏ Fehlerquelle Nr.x sind Systemfehler

☞ ausführliche Fehlerbeschreibung an: **`problems-spp@rrze`**

Viel Erfolg !

Übungsaufgabe zu PVM3

❏ Schreiben Sie ein PVM-Programm zur parallelen Glättung eines 2-D-Gitters.

❏ Sequentielles Beispiel: ~demo/Beispiel/grid.c

❏ Inititalisierung:

```
void init_grid(grid,dim)
double *grid;
int dim;
{
    int i,x,y;
    for (x = 0 ; x < dim ; x++)
        for(y = 0 ; y < dim ; y++)
            GRID(x,y) = 0.0;

    for (i = 0 ; i < dim ; i++)
        {
            GRID(0,i) = (1.0/dim)*(dim-1-i);
            GRID(i,(dim-1)) = (1.0/dim)*i ;
            GRID(i,0) = (1.0/dim)*(dim-1-i);
            GRID((dim-1),i) = (1.0/dim)*i;
        }
}
```

Übungsaufgabe zu PVM3

- Glättung

```
void smooth_grid(grid,dim,iterations)
double *grid;
int dim, iterations;
{
    int i,x,y;
    for (i = 0 ; i < iterations ; i++)
        {
            for (y = 1 ; y < dim-1 ; y++)
                for(x = 1 ; x < dim-1 ; x++)
                 GRID(x,y) =(GRID(x-1,y)+GRID(x+1,y)+GRID(x,y+1)+GRID(x,y-1)
                             /4.0;
            print_grid(grid,dim,i);
        }
}
```

- Abgabe der Lösung erfolgt durch eine email an problems-spp@rrze mit Angabe des Verzeichnisses auf dem SPP-System bis zum 31.7.94.
  - Parallelisierung: Partitionierung des Gitters (z.B. in Streifen)
  - Das Lösungsprogramm soll mit einer beliebigen Prozessorzahl ablaufen können. Der Aufwand beträgt ca. 250 Zeilen kommentierter Code.

xpvm-Console