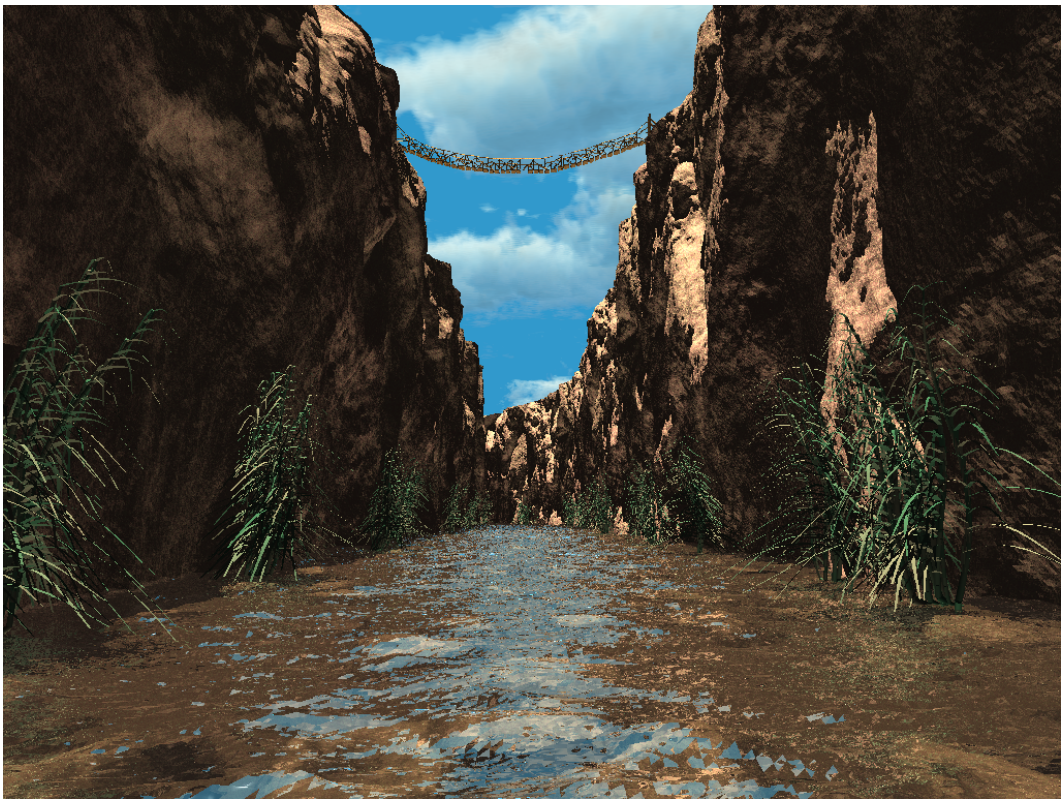
Supercomputer der Beowulf-Klasse

The LinOX Project

Arbeit Angewandte Mathematik
17. November 2000



Micha Surber (Jg. 1981)
<surber.frauenfeld@bluewin.ch>
Weizenstrasse 7
8500 Frauenfeld/TG
052 / 721 34 24
Klasse 7ora, Kantonsschule Frauenfeld

Manfred Stock (Jg. 1981)
<manfred_stock@yahoo.com>
Brunnenwiesen 4
8556 Wigoltingen/TG
052 / 763 27 81
Klasse 7ora, Kantonsschule Frauenfeld

Das Titelbild stammt von Jaime Vives Piqueres. Man kann es bei [27] downloaden. Es wurde mit Pvmpov 3.1.2 auf einem 3 Node Cluster gerendert. Der Cluster bestand aus zwei PIII 600 MHz und einem PPC Workgroup Server 133MHz.

Dieses Dokument wurde mit LaTeX, Version 3.14159, gesetzt.

Dank

Wir bedanken uns an dieser Stelle bei all denjenigen, die uns bei dieser Arbeit unterstützt haben. Insbesondere sind dies:

- Jonathan Gysel, der uns beim Booten des Clusters half und uns mit Nahrung versorgt hat;
- Florian Luethi, der uns die coole Webpage designt und uns moralisch unterstützt hat;
- Lukas Lippert, der uns programmiertechnisch einige Stunden unterstützte;
- Bernhard Brunner, der die Arbeit freundlicherweise korrekturgelesen hat;
- Angelo Lombardi, der uns bei LaTeX-Fragen geholfen hat;
- Mitarbeiter des Hausdienstes, die uns während den Ferien den Zugang zur Infrastruktur ermöglichten;
- Kantonsschule Frauenfeld, die uns die Infrastruktur zur Verfügung stellte.

Inhaltsverzeichnis

1 Einleitung

1.0 GHz, 1.1 GHz und so weiter. Grosse Zahlen, schnelle Rechner. Für einige unserer Anwendungen aber immer noch nicht schnell genug. Nun gibt es verschiedene Möglichkeiten, die Rechenleistung zu verbessern. Die Einen setzen dabei auf teure Superrechner mit extrem schneller Hardware, parallel geschalteten Prozessoren, viel Arbeitsspeicher, schnellen Festplatten und geben dafür riesige Geldmengen aus. Wenn nun aber die finanziellen Mittel nicht im Überfluss vorhanden sind, gibt es auch andere Möglichkeiten, Rechenleistung zu erhalten.

Von der zweiten, günstigeren Variante handelt unsere Arbeit. Supercomputer der Beowulf-Klasse [4] bestehen aus vielen kosteneffizienten Nodes, welche nur über eine minimale Hardware-Ausstattung verfügen müssen. Auf Bildschirm, Grafikkarte, Festplatte, CD-Rom Laufwerk, Soundkarte, etc. kann verzichtet werden. Unerlässlich sind aber Netzwerkkarte, Arbeitsspeicher, Prozessor und Gehäuse mit Netzteil ;-).

Möglich ist dies, weil die Rechner durch ein Netzwerk verbunden sind. Sämtliche Daten, die erforderlich sind, holen sie sich via Bootp (Siehe [7], [8], [9]) und NFS aus dem Netzwerk.

Das Ziel dieser Arbeit ist es, zu zeigen, wie ein solcher Cluster funktioniert, wie man ihn baut und wie man einige praktische Anwendungen darauf ausführt. Dies beinhaltet unter anderem das Programmieren von einfachen, parallelen Anwendungen und das Ausführen diverser Benchmarks für internationale Vergleichbarkeit.

Das Ganze läuft unter dem Namen "The LinOX Project". Die Homepage unseres Projektes ist unter `<http://www.LinOX.cjb.net>` erreichbar.

Teil I
Cluster

2 Was ist ein Cluster?

Ein Cluster besteht aus mehreren Computern, die vernetzt sind und gemeinsam an einer grösseren Rechenaufgabe arbeiten. Er enthält einen Master und verschiedene Nodes. Der Master teilt die Aufgabe auf die Clients auf, erhält von ihnen die Resultate und verarbeitet sie. Im Gegensatz zu den Clients ist der Master ein komplett ausgestatteter Computer (mit Bildschirm, Keyboard, etc.). Von ihm aus startet man die Aufträge und schaut sich die Resultate an.

3 Wie funktioniert ein Cluster?

Die Clients werden vom Master über das Netzwerk gesteuert. Auf dem Master werden die Programme gestartet und über Programmier-Interfaces wie PVM (siehe auch [19]) oder MPI (siehe auch [16]) die Aufträge an die einzelnen Clients gesendet. Diese Schnittstellen sind auch für das sogenanntes "Message-Passing" zuständig. Bei diesem Vorgang werden Daten und Nachrichten zwischen den Prozessen ausgetauscht. Wenn dieser Austausch zwischen verschiedenen Plattformen stattfinden soll, muss noch eine Konvertierung der Daten durchgeführt werden.

4 Wozu dient ein Cluster?

Die Rechenleistung, die ein Cluster bietet, kann für verschiedene Zwecke eingesetzt werden. Es ist möglich, medizinische Vorgänge zu simulieren, die Klimaentwicklung vorauszuberechnen oder auch Atombomben gefahrlos zu testen. Ausserdem kann man die Lösung aufwendiger, rechenintensiver Mathematik-Probleme (z.B. stochastische Probleme mit "Monte-Carlo"-Variante) damit schneller finden.

In der Schweiz hat vor kurzem die ETH Zürich einen neuen Beowulf-Cluster in Betrieb genommen (siehe [2]), der 502 Prozessoren besitzt. Zum Zeitpunkt der Inbetriebnahme war dies der grösste Beowulf-Cluster Europas. Er wird dort unter Anderem in der Teilchenphysik, der theoretischen Physik und der angewandten Mathematik eingesetzt.

5 Wieso Beowulf-Cluster?

Im Sommer des Jahres 1994 bauten Thomas Sterling und Don Becker, die am CESDIS[1] arbeiteten und vom ESS[2]-Projekt unterstützt wurden, einen Cluster, der aus 16 DX4 Prozessoren bestand. Die einzelnen Computer dieses Clusters waren über Channel Bonded Ethernet verbunden.

Diesen Cluster nannten sie Beowulf. Der Name "Beowulf" stammt aus einer alten, englischen Sage, in welcher ein Held einen Riesen nach dem Anderen besiegt. Die Maschine war ein Erfolg und ihre Idee kam über die NASA schnell in akademische und wissenschaftliche Bereiche. Aus diesen Entwicklungsanstrengungen entstand dann das, was man heute "Beowulf-Project" nennt (siehe auch [4] und [5] sowie [3] und [23]).

[1] Center of Excellence in Space Data and Information Sciences
[2] Earth and space sciences

Teil II
Aufbau des Clusters

6 Konzept

Als Informatikassistenten der Kantonsschule hatten wir die Möglichkeit, die Informatikanlagen für unser Projekt zu verwenden. Ziel war es, aus möglichst vielen Workstations einen Cluster zu bauen. Das Hauptproblem war, dass die Workstations alle auf Windows NT 4.0 laufen. Wir konnten das Projekt aber nur mit Linux durchführen, weil NT nicht ausreichend Stabilität bietet und ein grafisches Betriebssystem zuviel Ressourcen für sich beansprucht, was zu Performanceeinbussen führen würde.

Nun war es aber leider nicht möglich, auf sämtlichen Rechnern Linux zu installieren. Infolgedessen mussten wir einen Weg finden, um ein Linux aus dem Netzwerk zu booten. Nach einigen Tagen und Nächten hatten wir die funktionierende Lösung gefunden: Wir booten die Rechner von einer Diskette mit einem selbst kompilierten Linux-Kernel und weisen ihm via Bootp eine IP-Adresse, einen Hostnamen und ein Verzeichnis auf dem Server mit seinen Daten zu. Dieses Verzeichnis mountet der Rechner dann über NFS vom Server.

Als Programmier-Interface für parallele Anwendungen haben wir PVM[3] (siehe [19]) verwendet. Um den PVM-Daemonen auf den Clients zu starten, muss man diese per RSH[4] steuern können. Nach diesen Arbeiten ist der Cluster funktionsfähig gewesen: Wir mussten "nur" noch nach passenden Anwendungen suchen.

Wir haben hauptsächlich PVMPOV[5] (siehe [20]) verwendet, eine PVM-Version des bekannten Raytracers Povray[6] (siehe [18]). Damit haben wir auch Benchmarks gemacht, mit deren Hilfe wir die Rechenleistung unseres Clusters mit anderen Clustern international vergleichen konnten.

7 Der Bootvorgang beim Client

Das Booten eines Rechners ist ein vergleichsweise komplizierter Vorgang. Bei Linux hat man nun die Möglichkeit, an jeder Stelle des Vorgangs einzugreifen.

Von dieser Möglichkeit haben wir Gebrauch gemacht. Wir haben dabei schon beim Kern des Betriebssystems angefangen: Wir haben uns einen eigenen, monolithischen Linux-Kernel [13] konfiguriert (Kernel-Version 2.2.16) und kompiliert. Dabei haben wir dem Kernel mitgeteilt, dass er sich IP-Adresse und Hostname via Bootp und das Root-Dateisystem per NFS vom Server holen soll.

Der nächste Schritt bestand in der Anpassung der Bootskripte. Da ein Unix-Rechner normalerweise von einer Harddisk startet und dort auch Daten (Logfiles, temporäre Dateien, Swap-Partition) ablegt, mussten wir diese Skripte teilweise sehr stark anpassen, denn die Festplatte durfte auf keinen Fall mit irgendwelchen Daten beschrieben werden, weil dies die NT-Installation in Mitleidenschaft gezogen hätte. Also mussten wir für einige Verzeichnisse Ram-Laufwerke erstellen (`/etc`, `/var`, `/tmp`) und Swap deaktivieren.

7.1 Kernelkonfiguration

Bei der Konfiguration des Kernels musste man vor Allem darauf achten, dass der Kernel monolithisch ist, das heisst: Dass er nicht modular aufgebaut ist, alle Treiber müssen in den Kernel einkompiliert sein. Für uns hiess das, dass wir Netzwerkkartentreiber, NFS-Filesystem-Treiber und Bootp-Unterstützung unbedingt einkompilieren mussten. So erhielten wir nach der Kompilation einen relativ minimalen Kernel, der für unsere Zwecke angepasst war. Danach haben wir ihn auf eine Diskette geschrieben und das Root-Device auf NFS abgeändert.

[3] Parallel Virtual Machine
[4] Remote-Shell
[5] PVM-Povray
[6] Persistence of Vision Raytracer

7.2 Bootvorgang im Detail

Beim Booten des Kernels werden alle notwendigen Treiber geladen. Am Ende des Kernelstarts sendet er Bootp-Requests ins Netzwerk, worauf ihm ein Bootp-Server seine IP, den Hostnamen und das Root-Verzeichnis auf dem Server zuweist. In diesem Root-Verzeichnis befinden sich unter vielem Anderem die Bootskripte. Diese werden nach dem Mounten des Root-Dateisystems vom `INIT`-Prozess gestartet. Das für uns wichtigste Skript war `/sbin/init.d/boot`, welches einige Änderungen erforderte.

Im Folgenden beschreiben wir den Ablauf dieses Bootskriptes etwas genauer.

Zu Beginn wird das `/proc`-Device gemountet. Danach werden verschiedene Optionen bearbeitet und ein Daemon (bdflush) gestartet. Nun werden einige Verzeichnisse vom Server gemountet: `/usr`, `/bin`, `/sbin`, `/lib` und `/home`. Im `/usr`-Verzeichnis befinden sich Programme, Quelltexte, etc. Das `/bin`-Verzeichnis enthält mehrere kleine, aber wichtige Programme, z.B. `dd`, `cp`, `kill`, usw. `/sbin` beinhaltet verschiedene Programme, die der Systemadministrator bei seiner täglichen Arbeit braucht. Im `/lib`-Directory sind einige Programmbibliotheken abgelegt und das `/home`-Verzeichnis beherbergt die Benutzerdaten.

Nach dem Mounten folgt das Erstellen von drei Ramlaufwerken für die Verzeichnisse `/tmp`, `/etc` und `/var`. Diese Massnahme erwies sich als notwendig, weil in diese Verzeichnisse verschiedene Daten geschrieben werden müssen und dies nicht zu übermässigem Traffic im Netzwerk führen soll. Da es nur temporäre Daten sind, ist es irrelevant, dass sie nach einem Reboot des Clients nicht mehr vorhanden sind. Daher werden die Daten der Verzeichnisse in diese Ramdisks kopiert:

```
#/tmp
mke2fs /dev/ram0
mount /dev/ram0 /tmp
chmod -R a+w /tmp

#/etc
mkdir /tmp/etc
cp -r /etc /tmp/
mke2fs /dev/ram1
mount /dev/ram1 /etc
cp -r /tmp/etc/* /etc
rm -r /tmp/etc

#/var
cp -r /var /tmp/
mke2fs /dev/ram2
mount /dev/ram2 /var/
cp -r /tmp/var/* /var/
rm -r /tmp/var/
chmod -R a+w /var
```

Nun wird das pts-Device nach `/dev/pts` gemountet, falls dies noch nicht geschehen ist. Danach bildet `ld`[7] seinen Cache neu, `zic`[8] setzt die Zeitzone und das Loopback-Device wird initialisiert. Es folgt noch das Aufsetzen des Netzwerks, das heisst, dass Hostname und Domainname gesetzt werden. Am Ende werden noch einige Aufräumarbeiten ausgeführt, andere Skripte gestartet, Kernelnachrichten gesammelt und mehrere weitere (hier nicht sehr wichtige) Optionen aktiviert.

Nun wechselt der Rechner vom Runlevel N zum Runlevel 2. Hier werden noch weitere Dienste gestartet: Die seriellen Ports werden konfiguriert, das Netzwerkdevice (`eth0`) aufgesetzt, Routing aktiviert, Syslog-Service gestartet, Zufallszahlengenerator initialisiert, `inetd` gestartet und die Keymap `qwertz/de-lat1-nd.map.gz` geladen.

Nach erreichen des Runlevels 2 erhält man am Client einen Rootlogin-Prompt. Man könnte sich jetzt als `root` einloggen - das ist jedoch nicht notwendig, da die Clients über das Netzwerk gesteuert werden, kann allerdings für Diagnosezwecke überaus sinnvoll sein.

[7] GNU linker
[8] Time Zone Compiler

7.3 Aufgaben des Servers beim Clientstart

Während des Starts der Clients muss der Server auf Bootp-Requests antworten und auch schon die benötigten Verzeichnisse exportieren. Wenn nun viele Diskless-Clients am Cluster beteiligt sind, muss der Server auch genug leistungsfähig sein. In unserem Fall war der Server ein 600Mhz schneller Pentium III mit 256Mb Arbeitsspeicher und 18Gb SCSI-Platte. Diese Leistung ist für unser Projekt ausreichend. Wenn man allerdings mit mehr Clients arbeiten will, ist man auf stärkere Server angewiesen. Dort wäre es dann zum Beispiel sinnvoll, einen vom Master getrennten Fileserver zu benützen, der über ein schnelles Festplattensystem, RAID[9], verfügt.

8 Konfiguration des Servers

Die Konfiguration des Servers ist nicht ganz trivial. Zuerst muss man den Bootp-Server einrichten. Dies geschieht in der `inetd.conf` mit der folgenden Zeile:

```
bootps  dgram   udp     wait    root    /usr/sbin/bootpd        bootpd -c /tftpboot
```

Nach einem `killall -HUP inetd` ist der Bootp-Server aktiv. Man muss allderdings in der `bootptab` für jeden Client eine Zeile wie:

```
wks55:hd=/tftpboot/192.168.100.44:ip=192.168.100.44:ha=006008afc1d3:bf=null:
```

einfügen. Durch diese Anweisung wird dem Computer mit der `MAC`-Adresse `00:60:08:af:c1:d3` der Name `wks55` und die IP-Adresse `192.168.100.44` zugewiesen. Ausserdem erfährt der Client, dass sein Root-Verzeichnis auf dem Server im Verzeichnis `/tftpboot/192.168.100.44` liegt, ihn aber kein Bootfile erwartet.

In diesem Verzeichnis findet der Client ein Root-Verzeichnis vor, das alle benötigten Dateien für das Booten enthält. Das Erstellen dieses Verzeichnisses geschieht mit zwei Skripten, die wir während des Installierens des Clusters noch auf SuSE-Linux portieren mussten, da sie ursprünglich für RedHat geschrieben waren.

Zuerst haben wir dabei mit dem Skript sdct[10] ([22]) ein Template erstellt, woraus man danach mit dem zweiten Skript (adcn[11] ([1])) die Root-Verzeichnisse für die Clients erzeugt.

8.1 Das `sdct`-Skript

Dieses Skript erstellt das NFS-Root-Template für die Diskless-Clients. In diesem Template befindet sich eine komplette Unix Root-Verzeichnisstruktur [11]. Dabei sind allerdings einige Verzeichnisse leer. In diese Verzeichnisse werden die Original-Verzeichnisse des Servers beim Booten des Clients gemountet. Das ist zum Beispiel notwendig, weil die Verzeichnisstruktur beim Ablauf der Clusterprogramme auf jedem beteiligten Rechner identisch sein sollte und im Falle des `/home`-Verzeichnisses sogar muss.

8.2 Das `adcn`-Skript

Die Aufgabe dieses Skriptes besteht darin, angepasste Kopien des Templates für jeden Node des Clusters zu erstellen. Dabei müssen einige Dateien angepasst werden, so zum Beispiel der Hostname und die `fstab`. Ausserdem fügt das Skript auf dem Server bei den Dateien `bootptab`, `hosts` und `exports` den neuen Node hinzu. Nach diesen Arbeiten wird der NFS-Server neu gestartet und noch etwas aufgeräumt. Nun wird mit

```
echo "Done."
```

die Meldung ausgegeben, dass das Skript seine Arbeit erledigt hat und man den Client von der Kernel-diskette booten kann.

[9] Redundant Array of Independent Disks
[10] setup diskless client template
[11] add computing node

Teil III
Parallele Ausführung von Programmen

Um auf den vernetzten, aus dem Netzwerk gestarteten Linux-Rechnern nun parallele Programme auszuführen, braucht man entsprechende Programmierschnittstellen. Wir haben PVM verwendet, weil es darüber viel Dokumentationen gibt (siehe auch: [19]). Ausserdem hat es sich als De Facto Standard durchgesetzt und ist für Fortran und C/C++ verfügbar.

Die Aufgabe von PVM besteht darin, Daten zwischen den einzelnen Rechnern des Cluster auszutauschen und gegebenenfalls zu konvertieren. Zudem ist es für die Verteilung der Aufträge zuständig. An einem kleinen Beispiel werden wir nun die Funktionsweise von PVM etwas näher aufzeigen.

9 Der Start des PVM-Systems

Bevor man das System starten kann, muss man sicherstellen, dass man auf den zum Cluster hinzuzufügenden Nodes via RSH Befehle ausführen kann, weil der lokale PVM-Daemon so gestartet wird. Dies erforderte bei unserem Cluster eine Änderung der PAM[12]-Konfiguration (PAM: Siehe auch [17]). Auf dem Master startet man nun mit dem Befehl `pvm` den PVM-Daemon und ein Shellartiges Interface zu PVM:

```
s861002@lsvr03:~ > pvm
pvm>
```

Jetzt fügt man mit dem Befehl `add [Computername]` die Clients zum Cluster hinzu. Es ist allerdings auch möglich, eine Datei mit einer Liste der Clients als Parameter beim Start von `pvm` zu übergeben und sich so viel Tipparbeit zu ersparen, da die Clients dann automatisch hinzugefügt werden. Nun kann man, zum Beispiel in einer anderen Konsole, PVM-Anwendungen starten. Als Alternative zu dieser Startvariante gibt es `xpvm` (siehe Abbildung 1, Seite 8). Dies ist ein grafisches Interface für PVM, das sich mit der Maus bedienen lässt. Damit lässt sich der Cluster beinahe "zusammenklicken". Dieses Programm verfügt auch über die Möglichkeit, den Ablauf der PVM-Programme grafisch darzustellen.

Abbildung 1: `xpvm`, das grafische Interface von `pvm`

[12] Pluggable Authentication Modules

9.1 Kompatibilität

Das PVM-System ist so aufgebaut, dass man es Plattformübergreifend einsetzen kann. Versuchsweise haben wir daher noch einen Workgroup Server 8550/132 von Apple in einen kleinen Cluster aufgenommen. Dieser Server besitzt einen PowerPC-Prozessor und läuft auf SuSE Linux 6.4 für PowerPC. Wir mussten in erster Linie nur das `/home`-Verzeichnis unseres Servers via NFS mounten und die Prozessorspezifischen Teile von PVM in die Verzeichnisstruktur von `/home/$HOME/$PVM_ROOT` hineinkopieren (in der Variable `$PVM_ROOT` ist das Verzeichnis angegeben, das die PVM-Dateien enthält). Nun konnte man den PowerPC wie andere Nodes in der Cluster aufnehmen:

```
s861002@lsvr03:~ > pvm
pvm> add macsvr01
add macsvr01
1 successful
                    HOST     DTID
                 macsvr01    80000
pvm> conf
conf
3 hosts, 2 data formats
                    HOST     DTID     ARCH   SPEED       DSIG
                   lsvr03    40000    LINUX    1000 0x00408841
                 macsvr01    80000 LINUXPPC    1000 0x0658eb59
                   lsvr05    c0000    LINUX    1000 0x00408841
pvm>
```

Die Verwendung anderer Plattformen setzt allerdings meist eine Neukompilation der Programme voraus. Die wichtigsten Programme (PVM, Pvmpov) waren schon bei der SuSE-Distribution dabei. Daher mussten wir nur die eigenen Anwendungen neu kompilieren, was aber aufgrund der Struktur und Unterstützung von PVM kein Problem darstellte. Grundsätzlich dürften die meisten UNIX-Varianten ohne Probleme in einen Beowulf-Cluster aufgenommen werden können, da sie untereinander vergleichsweise kompatibel sind.

Theoretisch wäre es sogar möglich, Computer mit Microsoft Windows in ein PVM-System aufzunehmen. Bei einem kurzen Versuch hatten wir jedoch aufgrund einiger massiver Einschränkungen dieses Betriebssystems keinen Erfolg.

10 Hallo, Welt!

Dieses kleine Master-Programm `hello.c` zeigt die wichtigsten Grundlagen der PVM-Programmierung. Zuerst gibt es mit `pvm_mytid()` seinen Task Identifier[13] aus. Es startet jetzt mit der Funktion `pvm_spawn()` eine Kopie eines anderen Programmes, des Slave-Programmes (`hello_other`), auf einem Node. Wenn diese Funktion erfolgreich beendet wurde, wartet der Master nach dem Kommando `pvm_recv()` auf eine Antwort des Slaves. Nun gibt das Programm die Botschaft des anderen Programmes an der Konsole aus, welche mittels `pvm_upkstr()` aus der Message extrahiert wurde. Auch der Tid des anderen Tasks wird dabei ausgegeben. Dieser Vorgang wird wiederholt, solange `i<=9` ist. Dadurch werden mehrere Tasks erzeugt, welche sich dann mit unterschiedlichen Tid's melden. Am Ende wird durch `pvm_exit()` das Programm vom PVM-System getrennt.

```
/*
   Program hello.c
*/

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

[13] Bei PVM erhält jeder Auftrag, Task, eine eindeutige ID, den sog. Task Identifier, kurz Tid.

```
main()
{
   int cc[10], tid[10],i;
   char buf[10][100];

   printf("i'm t%x\n", pvm_mytid());

   for ( i=0; i<=9; i++ ){
      cc[i] = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid[i]);
      if (cc[i] == 1) {
         cc[i] = pvm_recv(-1, -1);
         pvm_bufinfo(cc[i], (int*)0, (int*)0, &tid[i]);
         pvm_upkstr(buf[i]);
         printf("from t%x: %s\n", tid[i], buf[i]);
      } else
      printf("can't start hello_other\n");
   }
   pvm_exit();
   exit(0);
}
```

Das Slave-Programm `hello_other.c` wird vom obigen Masterprogramm gestartet. Seine erste Handlung ist das Abfragen der Tid des Masterprozesses, die es via `pvm_parent()` erfährt. Danach fragt es den Hostnamen des Clients ab, auf dem es läuft, und überträgt ihn nach dem initialisieren des "Send Buffers" mit `pvm_initsend()` per `pvm_send()` zum Master. `pvm_pkstr()` platzierte den String im Sendepuffer.

```
/*
   Slave-Program 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);
}
```

10.1 Die Ausgabe

Die Ausgabe dieses Programms sieht auf einem 9 Node Cluster so aus:

```
Linux lsvr03 2.2.14 #1 Mon Mar 13 10:54:26 GMT 2000 i686 unknown
```

```
s797010@lsvr03:~ > pvm
pvmd already running.
pvm> conf
conf
22 hosts, 1 data format
                     HOST     DTID     ARCH   SPEED       DSIG
                   lsvr03    40000    LINUX    1000 0x00408841
                    wks24    80000    LINUX    1000 0x00408841
                    wks25    c0000    LINUX    1000 0x00408841
                    wks38   100000    LINUX    1000 0x00408841
                    wks48   140000    LINUX    1000 0x00408841
                    wks53   180000    LINUX    1000 0x00408841
                    wks73   1c0000    LINUX    1000 0x00408841
                    wks09   200000    LINUX    1000 0x00408841
                    wks17   240000    LINUX    1000 0x00408841
                    wks18   280000    LINUX    1000 0x00408841
pvm> quit
quit

Console: exit handler called
pvmd still running.
s797010@lsvr03:~ > cd /home/s797010/pvm3/bin/LINUX/&& ./hello
i'm t40008
from t340001: hello, world from 192.168.100.72
from t380001: hello, world from 192.168.100.31
from t3c0001: hello, world from 192.168.100.66
from t400001: hello, world from 192.168.100.82
from t440001: hello, world from 192.168.100.77
from t480001: hello, world from 192.168.100.33
from t4c0001: hello, world from 192.168.100.39
from t500001: hello, world from 192.168.100.62
from t540001: hello, world from 192.168.100.63
s797010@lsvr03:~/pvm3/bin/LINUX >
```

Zuerst wird `pvm` gestartet und mit dem Befehl `conf` werden die Nodes aufgelistet. Danach wird das Programm `hello` auf dem Server gestartet. Jeder Client sendet nun seine Zeile zurück.

Teil IV
Clusteranwendungen

11 MP3-PVM

MP3 [14] ist ein verlustbehaftetes Kompressionsverfahren für Audiodaten, welches vom Fraunhofer Institut [12] entwickelt wurde. Es ist auf der ganzen Welt sehr verbreitet. Eine im MP3-Format kodierte Musikdatei ist ca. zehn mal kleiner als ein unkomprimiertes Audiofile. Um diese Dateien zu erstellen, braucht es eine grössere Rechenleistung. Im Internet haben wir ein PVM-Programm gefunden, das diese Arbeit auf mehrere Rechner aufteilen kann: MP3-PVM [15].

Dieses Program grabbt von einer Audio-CD Wave-Files, die es im Cluster in MP3-Dateien umwandelt.

Neben den Sourcen für das eigentliche Programm braucht man noch einen Ripper und einen Encoder. Wir verwendeten `cdparanoia` [10] und `bladeenc` [6].

11.1 Ablauf des Programms

An diesem Beispiel sieht man, wie mit verschiedenen Rechnern parallel gearbeitet werden kann (siehe Abbildung 2, Seite 12). Der Server hat ein CD-Rom-Laufwerk, womit er die Wave-Dateien grabbt. Dies kann vom Server alleine getan werden, weil nicht das Auslesen, sondern das Komprimieren viel Rechenleistung benötigt.

Die Dateien werden in ein Verzeichnis geschrieben, auf welches alle Nodes über NFS zugreifen können. Jeder Client erhält dann den Befehl, ein Stück zu kodieren.

Der Nachteil dieses Verfahrens ist, dass es keinen Sinn macht, einen Cluster zu verwenden, der mehr Nodes besitzt als die zu bearbeitende CD Stücke. Eine schönere Lösung wäre die Zerteilung der Songs. Dadurch könnte man die Arbeit an beliebig viele Rechner verteilen und den Vorgang beschleunigen.

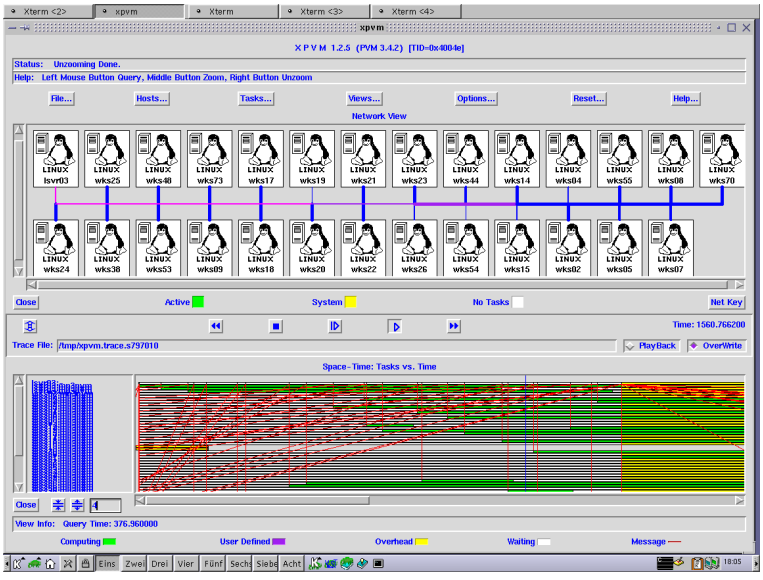


Abbildung 2: Die Ausgabe von XPVM beim Ausführen von mp3pvm

12 PVM-Povray

Raytracing ist eine Methode, um auf einem PC photorealistische Bilder zu berechnen. Der erste Teil der Arbeit ist das Beschreiben einer Szene. Dies wird etwa folgendermassen gemacht: Man gibt an, wo

man welche Körper zeichnen will, weist ihnen Farben und Texturen zu und gibt eventuell noch andere Optionen an. Zusätzlich werden noch eine Kameraposition und Blickrichtung sowie eine oder mehrere Lichtquellen benötigt, damit man überhaupt etwas sehen kann. Zum Thema Raytracing siehe auch [18], [21].

Das folgende Beispiel beschreibt eine rote Kugel vor einem orangen (`Coral`) Hintergrund:

```
#include "colors.inc"

background { color Coral }

camera {
   location <0,2,-3>
   look_at <0,1,2>
}

sphere {
   <0,1,2> 2
   texture {
      pigment {color Red}
   }
}

light_source { <2,4,-3> color White }
```

Povray ist ein hochwertiger Raytracer, der frei erhältlich ist, das auch im Sourcecode. Dies ermöglichte auch die Portierung auf PVM. Erhältlich ist Povray für die Plattformen Amiga, SunOS, i86 Linux, Macintosh und sogar DOS und Windows 95/98/NT/2000/ME.

Das Prinzip des Raytracings ist es, dass man für jeden einzelnen Pixel die Farbe berechnet, in der er dargestellt werden soll. Die Berechnung der einzelnen Punkte ist also völlig unabhängig und kann deshalb gut auf die verschiedenen Nodes aufgeteilt werden. Jeder Node kann mit Hilfe von NFS auf das `/home`-Verzeichnis des Servers zugreifen und dort die Povray-Source auslesen. Vom Server wird ihm dann eine bestimmte Anzahl Pixel zugeteil, die er berechnet und deren Farbe er an den Server zurück sendet. Raytracing ist also eine ideale Clusteranwendung, weil es enorm Rechenaufwendig und zugleich sehr gut auf mehrere Rechner aufzuteilen ist.

Im Internet findet man diverse POV-Ray-Sourcen, die oftmals aufwendige Bilder beschreiben. Diese beinhalten nicht nur mit Texturen versehene Objekte, sondern auch Spiegelungen und Transparenzeffekte.

12.1 Ein Beispielbild

Die PVM-Source `Strike.pov` (siehe auch [26]) ist eine Darstellung einer Seeschlacht. Die Quelle ist ein 642 Zeilen langes Textdokument, in welches diverse Dateien eingebunden sind. Der auf dem Bild (Abbildung 3, Seite 14) dargestellte Rauch und die Reflexionen auf dem Wasser (insbesondere auf den Wellen) sind sehr aufwendig zu berechnen.

Das Rendern des Bildes auf einem 28 Node Cluster ergibt etwa folgende, stark verkürzte Ausgabe:

```
s861002@lsvr03:~/pvm3/bench/krieg > pvmpov Strike.ini +IStrike.pov +v
Persistence of Vision(tm) Ray Tracer Version 3.1g.Linux.gcc
  This is an unofficial version compiled by:
  Jakob Flierl <flierl@luga.de> - PVMPOV Version 3.1e.2
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing PVMPOV
  Spawning pvmpov with 29 PVM tasks on 29 hosts...
  ...29 PVM tasks successfully spawned.
  Waiting up to 120s for first slave to start...
```

Abbildung 3: `Strike.pov`, mit PVMPov gerendert.

```
  Slave 0 successfully started.
Parsing Options
  Input file: Strike.pov (compatible to version 3.1)
  Remove bounds.........On  Split unions.........Off
  Library paths: /usr/share/povray31 /usr/share/povray31/include
Output Options
  Image resolution 800 by 600 (rows 1 to 600, columns 1 to 800).
  Output file: Strike.png, 24 bpp PNG

⋮

  Render Stream to console.........On
  Statistics Stream to console....On
  Warning Stream to console.......On

Starting frame 0...
 Slave 1 at wks70 successfully started.
 Slave 2 at wks26 successfully started.
 Slave 3 at wks38 successfully started.
 Slave 4 at wks04 successfully started.
 Slave 5 at wks24 successfully started.
 Slave 6 at wks09 successfully started.
 Slave 7 at wks44 successfully started.
 Slave 8 at wks73 successfully started.
```

```
 Slave 9 at wks19 successfully started.
 Slave 10 at wks07 successfully started.
 Slave 11 at wks25 successfully started.
 Slave 12 at wks08 successfully started.
 Slave 13 at wks17 successfully started.
 Slave 14 at wks05 successfully started.
 Slave 15 at wks53 successfully started.
 Slave 16 at wks20 successfully started.
 Slave 17 at wks21 successfully started.
 Slave 18 at wks02 successfully started.
 Slave 19 at wks14 successfully started.
 Slave 20 at wks30 successfully started.
 Slave 21 at wks18 successfully started.
 Slave 22 at wks22 successfully started.
 Slave 23 at wks39 successfully started.
 Slave 24 at wks13 successfully started.
 Slave 25 at wks15 successfully started.
 Slave 26 at wks54 successfully started.
 Slave 27 at wks48 successfully started.
 Slave 28 at wks23 successfully started.
89.47 of blocks complete.Not using wks04 for reassignment (66%)e 0).
90.11 of blocks complete.
Slave at wks04 has exited.

⋮

98.68 of blocks complete.   600 of  600 lines finished (in frame 0).
Finishing frame 0...rtw. 600


Waiting for remaining slave stats.


PVM Task Distribution Statistics:
           host name  [ done ] [ late ]           host name  [ done ] [ late ]
              lsvr03  [12.78%] [11.67%]               wks70  [11.20%] [ 8.75%]
               wks30  [ 0.37%] [ 0.11%]               wks07  [ 5.33%] [ 6.67%]
               wks08  [ 3.63%] [ 2.72%]               wks05  [ 2.35%] [ 0.00%]
               wks39  [ 2.35%] [ 0.00%]               wks44  [ 8.80%] [ 6.93%]
               wks54  [ 1.28%] [ 0.00%]               wks14  [ 3.28%] [ 2.43%]
               wks13  [ 3.04%] [ 0.61%]               wks15  [ 0.21%] [ 0.11%]
               wks04  [ 2.24%] [ 0.00%]               wks02  [ 1.92%] [ 0.00%]
               wks17  [ 0.59%] [ 0.05%]               wks18  [ 0.80%] [ 0.00%]
               wks19  [ 2.13%] [ 0.00%]               wks20  [ 1.92%] [ 0.00%]
               wks21  [ 4.91%] [ 2.75%]               wks22  [ 0.11%] [ 0.05%]
               wks23  [ 1.44%] [ 0.05%]               wks24  [ 1.92%] [ 0.00%]
               wks25  [ 1.49%] [ 0.00%]               wks26  [ 3.84%] [ 1.65%]
               wks38  [10.35%] [ 6.56%]               wks73  [ 6.45%] [ 8.59%]
               wks48  [ 1.28%] [ 0.00%]               wks53  [ 1.92%] [ 0.00%]
               wks09  [ 2.08%] [ 0.00%]

⋮

Time For Trace:    5 hours 31 minutes  27.0 seconds (19887 seconds)
```

```
    Total Time:    5 hours 31 minutes  27.0 seconds (19887 seconds)
s861002@lsvr03:~/pvm3/bench/krieg >
```

Zuerst werden die einzelnen Nodes initialisiert. Danach werden die ersten Rechenaufgaben an die Slaves verteil. Am Schluss gibt es eine ausführliche Auswertung der Resultate, indem angezeigt wird, wer wieviel Prozent der Arbeit erledigt hat. Daraus erkennt man auch, welche Rechner schneller sind. Die `wks38` ist zum Beispiel ein PII 350[14] und hat deshalb bedeutend mehr Arbeit erledigt. Der `lsvr03` (der Server) hat nicht viel mehr berechnet, da er die Arbeit noch aufteilen musste. `wks70` (auch ein PII 350) war die einzige Maschine, die mit 100MBit ans Netzwerk angeschlossen war, weshalb sie noch mehr (11.2%) erledigt hat als `wks38` (10.35%). Dies zeigt, dass man die Leistung noch etwas optimieren könnte, wenn man die Bandbreite des Netzwerks erhöhen würde.

Wenn man dieselbe Berechnung nur mit dem Server (PIII 600Mhz, 256MB) ausführt, dauert es beinahe doppelt so lange:

```
Time For Trace:   11 hours 39 minutes   3.0 seconds (41943 seconds)
    Total Time:   11 hours 39 minutes   3.0 seconds (41943 seconds)
```

13 Die Berechnung von π mit Hilfe eines Beowulf-Clusters

Zur Veranschaulichung der Funktion eines Clusters haben wir eine eigene PVM-Anwendung programmiert. Das Programm berechnet π. Dies ist eine sehr rechenaufwendige Arbeit und deshalb für einen Cluster gut geeignet.

13.1 Wie berechnet man π?

Die Formel, um die Fläche eines Kreises zu berechnen, lautet folgendermassen:

$$A = r^2 \cdot \pi$$

Anhand der Kreisfläche und des Radius eines Kreises kann man also π ausrechnen. In unserem Programm betrachten wir ein Einheitsquadrat und, darin liegend, den Einheitskreis. Die Fläche des Einheitsquadrates ist 4, die des Einheitskreises ist π.

Wir verteilen nun im Einheitsquadrat Punkte und testen, ob diese in einem Quadranten des Einheitskreises liegen oder nicht. So entsteht folgende Gleichung:

$$\frac{Punkte\ im\ Kreis}{Total\ Punkte} = \frac{\pi}{4}$$

Wenn man den ganzen Kreis benützte, würde das Resultat vier mal weniger genau, weil man in jedem Quadranten wieder die gleichen Punkte berechnet. Deshalb betrachten wir in unserem Programm nur den ersten Quadranten.

13.2 Wie berechnet man π mit einem Cluster?

Wir teilen den ersten Quadranten in `nodes * nodes` Quadrate auf. Jeder Node berechnet `nodes` Tasks, die jeweils einem Quadrat entsprechen.

Wie schon beim `Hallo, Welt!` (siehe auch 10) besteht die Software aus einem Programm für den Master und einem für die Nodes. Das Masterprogramm wird auf dem Server gestartet. Es startet dann auf den Nodes jeweils das Clientprogramm. Dieses Programm sendet das Resultat an den Master zurück, welcher die Daten auswertet. Auf diese beiden Programme werden wir im Folgenden genauer eingehen.

[14] Hier fällt ausserdem die im Vergleich zu den Pentium MMX 200 leicht bessere FPU der PII zusätzlich ins Gewicht

13.2.1 pi.c

Hier erklären wir kurz den Aufbau von `pi.c`.

Zuerst wird mit dem folgenden Befehl das Headerfile `pvm3.h` eingebunden. Dieses enthält den Sourcecode der PVM-Funktionen des Programms und lag der PVM-Distribution bei.

```
#include "pvm3.h"
```

Nach der Variablendeklaration werden einige Variablen initialisiert, wie zum Beispiel `steps` und `nodes`. Der Wert der Variable `steps` ist die Wurzel der Anzahl Punkte, die jeder Node berechnet. Dies, weil jeder Client `steps * steps` Punkte zugeteilt bekommt. `nodes` ist die Anzahl der beteiligten Nodes. Es werden `nodes * nodes` Tasks gestartet, damit jeder Rechner gleich viele Tasks bekommt:

```
steps = 100000;
nodes = 1;
```

Die folgenden Schleifen sorgen dafür, dass der Quelltext dazwischen `nodes*nodes` mal ausgeführt wird. Dabei ändern sich jeweils die Werte der Variablen `x` und `y`, welche als Koordinaten dienen und angeben, welches Quadrat man gerade bearbeitet:

```
for(x=0;x<=nodes-1;x++){
   for(y=0;y<=nodes-1;y++) {
      //Dieser Quelltext wird nodes*nodes mal ausgef\"uhrt.
   }
}
```

Es ist nicht sinnvoll, ein Quadrat an einen Rechner weiterzugeben, wenn ohnehin alle darin enthaltenen Punkte innerhalb oder ausserhalb des Kreises liegen. Dies testen wir für jeden Eckpunkt. Wenn die vier Eckpunkte im Kreis liegen, dann liegen auch alle Punkte des Quadrates im Kreis. Mit den folgenden Programmzeilen wird das getestet. Ist ein Eckpunkt im Kreis, erhöht sich `drin` um 1:

```
q=x/nodes;
w=y/nodes;
z=q*q+w*w;
if (z<=1){
   drin++;
}

q=(x+1)/nodes;
w=y/nodes;
z=q*q+w*w;
if (z<=1){
   drin++;
}

q=x/nodes;
w=(y+1)/nodes;
z=q*q+w*w;
if (z<=1){
   drin++;
}

q=(x+1)/nodes;
w=(y+1)/nodes;
z=q*q+w*w;
if (z<=1){
   drin++;
}
```

Wenn `drin` nun grösser oder gleich 4 ist, dann muss man an der Variable `resultat` nichts verändern. Wenn `drin` kleiner oder gleich 0 ist, dann muss man $\frac{4}{nodes \cdot nodes}$ zum Resultat addieren:

```
if (drin>=4){

}
if (drin<=0){
   result=result+4/(nodes*nodes);
}
```

Wenn `drin` zwischen 0 und 4 liegt, das heisst, wenn nicht alle Punkte innerhalb oder ausserhalb des Kreises liegen, muss das Quadrat berechnet werden.

Mit der folgenden Zeile wird auf dem Node das Programm `pi_other` gestartet:

```
cc[x] = pvm_spawn("pi_other", (char**)0, 0, "", 1, &tid[y]);
```

Danach wird dem Node die Information gesendet, welches Quadrat er berechnen soll. Dies geschieht mit dem Array `ort` des Typs `int`. `pvm_parent();` fragt die `tid` des Master-Prozesses ab, um dem Node den Absender der Nachricht mitzuteilen. So weiss dieser, an wen er später das Resultat zurücksenden muss.

`pvm_initsend(PvmDataDefault);` öffnet einen Puffer, in den man die Daten, die versendet werden sollen, speichern kann. Mit `pvm_pkdouble(ort, 2,1);` werden die Informationen in den Puffer geschrieben. Das Wort `double` bedeutet hier, dass man einen Wert des Typs `double` sendet. In der Klammer steht, dass der Array `ort` gesendet wird und dass dieser aus höchstens 2 Elementen besteht.

Mit dem Befehl `pvm_send()` wird der Inhalt des Puffers an den Node mit dem Tid `tid[y]` gesendet:

```
if ((1<=drin)&&(drin<=(3))){
   cc[x] = pvm_spawn("pi_other", (char**)0, 0, "", 1, &tid[y]);
   if (cc[x] == 1){
      printf("Task erfolgreich an t%x uebergeben\n",tid[y]);

      count++;

      ort[0]=x;
      ort[1]=y;

      ptid = pvm_parent();
      pvm_initsend(PvmDataDefault);
      pvm_pkdouble(ort,2,1);
      pvm_send(tid[y], 1);

   } else
      printf("konnte pi_other auf t%x nicht starten\n",tid[x];
}
```

Nun wartet der Master auf die Resultate der Clients und wertet diese aus:

```
for (i=1;i<=count;i++){
   bufid=pvm_recv(-1, -1);
   pvm_bufinfo(bufid, &dum, &dum, &stid);
   pvm_upklong(buf, 1, 1);
   // printf("Habe von t%x %f erhalten\n",tid[i], buf[0]);
   result=result+buf[0]/(steps*steps*nodes*nodes)*4;
   printf("sein result ist %f\n",result);
   pvm_initsend(PvmDataDefault);
   pvm_send(stid, 0);
}
```

Wegen Problemen mit der Variablengrösse (siehe auch 13.4.1) haben wir nicht die Punkte, die im Kreis liegen summiert, sondern diejenigen, die nicht im Kreis sind. Um als Ausgabe wirklich π zu erhalten, muss man noch die folgende Rechnung durchführen:

```
pi=4-result;
```

Nun hat man ein Resultat für π berechnet, welches auf der Konsole ausgegeben sowie in eine Datei geschrieben wird:

```
printf("\npi = %e \n\n\n",pi);
datei = fopen("pi.txt","w");
fprintf(datei,"%e", pi);
```

13.2.2 pi_other.c

Das Programm `pi_other.c` hat beinahe die gleichen Variablen deklariert wie `pi.c`. Deshalb werden wir hier nicht weiter auf diese Deklaration eingehen.

Als erstes empfängt das Programm vom Master `pi.c` die Koordinaten des Quadrates, das es berechnen muss. Der Befehl `bufid = pvm_recv(-1, -1);` schreibt den empfangenen Puffer in die Variable `bufid`. Mit dem Befehl `pvm_bufinfo(bufid, dum, um, tid);` wird getestet, ob die Message wirklich vom Master stammt. `pvm_upkdouble(ort, 2, 1);` entpackt dann den Puffer und entnimmt ihm einen zweistelligen Array der Klasse `double`, den wir `ort` nennen.

Im weiteren Teil des Programmes werden die Variablen `ortx` und `orty` verwendet, weshalb hier die Elemente des Arrays in diese beiden Variablen übertragen werden:

```
bufid=pvm_recv(-1, -1);
pvm_bufinfo(bufid, &dum, &dum, &stid);
pvm_upkdouble(ort, 2, 1);

ortx=ort[0];
orty=ort[1];
```

Die folgenden `for`-Schleifen bestimmen für `nodes * nodes` Koordinaten, ob sie im Kreis liegen oder nicht. Liegen sie ausserhalb des Kreises, wird die Variable `send[0]` um eins erhöht:

```
for(i=(ortx+(0.5/steps*nodes))/nodes;i<=(1+ortx-(0.5/steps*nodes))/nodes\
       ;i=i+1.0/(steps*nodes)){
   for (o=(orty+(0.5/steps*nodes))/nodes;o<=(1+orty-(0.5/steps*nodes))/nodes\
            ;o=o+1.0/(steps*nodes)){
      z=i*i+o*o;
      if (z>1){
         send[0]++;
      }
   }
}
```

Die Funktion zum Zurücksenden der Werte ist die selbe wie schon beim Masterprogramm. Nur wurde dort mit dem Befehl `pvm_pkdouble();` eine Variable des Typs `double` gesendet. In diesem Beispiel wird mit Hilfe des Kommandos `pvm_pklong();` ein `longint` gesendet. Dies vor allem, weil ein `longint` um einiges grössere Zahlen speichern kann und weil ein `double` zu wenig exakt ist (mehr zur Wahl der Variablen: Siehe auch 13.4.2):

```
ptid = pvm_parent();
pvm_initsend(PvmDataDefault);
pvm_pklong(send,1,1);
pvm_send(ptid, 1);
```

13.3 Die Resultate

Bei einer Konfiguration mit `nodes=5;` und `steps=10000;` liefert das Programm folgende Ausgabe:

```
s797010@lsvr05:~/pvm3/bin/LINUX > time  pi

 Pi berechnen mit einem Linux-Cluster

 (c) by  Manfred Stock und Micha Surber
       Wigoltingen  Frauenfeld  2000

Ich bin t40101

Task erfolgreich an t40102 uebergeben
Task erfolgreich an t40103 uebergeben
Task erfolgreich an t40104 uebergeben
Task erfolgreich an t40105 uebergeben
Task erfolgreich an t40106 uebergeben
Task erfolgreich an t40107 uebergeben
Task erfolgreich an t40108 uebergeben

 Habe 7 Tasks gestartet.

sein result ist 0.264987
sein result ist 0.275438
sein result ist 0.351598
sein result ist 0.362049
sein result ist 0.571802
sein result ist 0.781555
sein result ist 0.857714
result = 8.577143e-01

pi = 3.142286e+00

real    1m15.749s
user    0m0.010s
sys     0m0.000s

s797010@lsvr05:~/pvm3/bin/LINUX >
```

13.4 Probleme bei solchen Berechnungen

Bei unserem Programm wird π mit einem PIII, 600MHz in 1 Minute und 16 Sekunden auf 2 Nachkommastellen genau berechnet, was nicht besonders exakt ist. Das Programm ist so optimiert, dass es praktisch nur noch das berechnet, was es muss. Trotzdem erreicht man kein vernünftiges Resultat. Wenn man mit den eingesetzten Zahlen rechnet, macht es Sinn, dass man kein genaueres Resultat erreicht. Denn mit `nodes=5;` und `steps=10000;` erreicht man für den Quotienten und den Divisor des Bruches, der π beschreibt, höchstens Werte von $2.5 \cdot 10^9$. Dabei kann man aber im besten Fall eine Zahl auf 9 Stellen genau festlegen. Und diese stimmen in unserem Fall nicht zwingend. Man müsste zum Teilen viel grössere Zahlen verwenden. Dies ist aber aus folgenden Gründen nicht möglich:

13.4.1 Grösse der Variablen

In 13.3 haben wir ein Beispiel mit `nodes=5;` und `steps=10000:` durchgerechnet. Dies ergibt Zahlen von bis zu $2.5 \cdot 10^9$. Eine Variable des Typs `int`, die man normalerweise für solche Berechnungen von

ganzzahligen Werten verwenden würde, reicht nur bis $3.2768 \cdot 10^4$. Wenn man noch weiter rechnet, bekommt man $-3.2768 \cdot 10^4$. Ein `longint`, den wir schliesslich auch verwendeten, reicht bis 2^{32} oder $4.69 \cdot 10^9$. Dies zeigt, dass man mit einer Auflösung von $10'000 \cdot 10'000$ Punkten pro Node schon an der oberen Grenze angelangt ist.

Man könnte nun die Anzahl der Nodes erhöhen und so das Ergebnis genauer machen. Nur wirkt sich z.B. die Verdoppelung der Nodeanzahl in einer Vervierfachung der Genauigkeit aus, dies ergibt aber nicht einmal eine Nachkommastelle mehr.

13.4.2 Die Genauigkeit der Variablen

Wenn man in c++ zwei Variablen des Types `int` dividiert, ist der Output zwangsläufig der nächsttiefere `int`. Das heisst, $\frac{5}{2}$ ist nicht gleich 2.5, sondern gleich 2. Deshalb musste eine der zwei Zahlen, die man bei der letzten Berechnung dividiert, ein `double` sein. Der Nachteil einer Variablen des Typs `double` ist aber, dass er die Zahlen nur auf einige Stellen genau speichert. Folglich verliert man hier wieder viel Genauigkeit.

13.5 Möglichkeiten zur Lösung des Problems

Eine Möglichkeit zur Lösung dieses Problems besteht darin, dass man eigene Variablentypen implementiert. In c und c++ gibt es die Möglichkeit, Variablen selber zu definieren. Diese könnte man dann beliebig genau machen und so genauere Resultate erzielen. Das Problem hierbei ist, dass man für diese eigenen Variablen auch eigene Rechenoperationen programmieren müsste, etc.

Um das Problem mit der Integerdivision (siehe 13.4.2) zu beheben, gäbe es die Möglichkeit, einen eigenen Divisionsalgorithmus zu programmieren. Dieser würde auf der schriftlichen Divisionsmethode, die man in der Primarschule lernt, beruhen und die Ziffern in eine Datei ausgeben.

Aus Zeitmangel hatten wir leider nicht mehr die Möglichkeit, diese Lösungen in die Praxis umzusetzen.

Teil V
Benchmarks

14 Povray

14.1 `skyvase.pov`

Das Bild `skyvase.pov` (siehe auch [24] und Abbildung 4, Seite 22) ist ein im Internet weit verbreiteter Benchmark. Da er noch aus den Anfängen des Raytracings stammt, ist er für heutige Rechner schon fast zu klein, um sie wirklich auszureizen, weil ein System seine volle Leistung erst nach einer gewissen Zeit entfaltet.

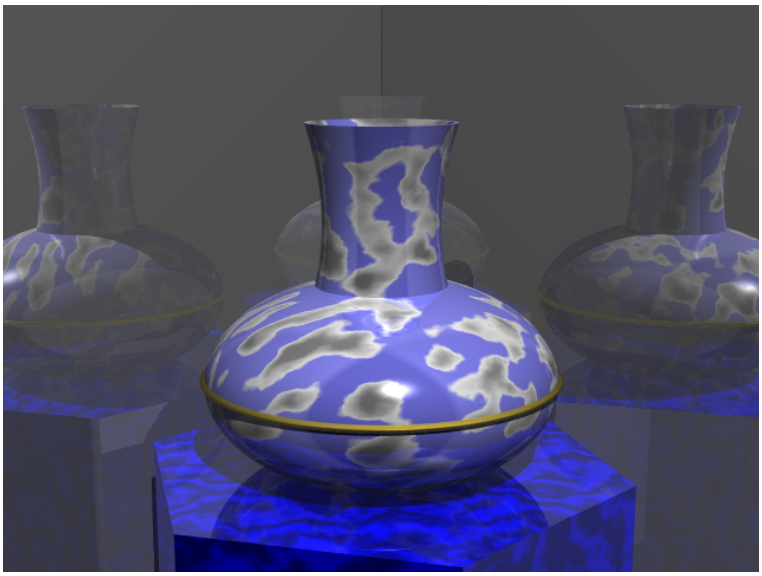


Abbildung 4: `skyvase.pov`: Ein Benchmark aus dem Internet

Ein grosses Problem eines Clusters ist, dass er viele Daten über das Netzwerk austauschen muss. In unserem Fall war das leider nur ein 10Mbit-Netzwerk. Dadurch sollte man darauf achten, dass man möglichst wenig Kommunikation benötigt. Dies bedeutet: Wenn man einen Node zum Cluster hinzufügt, wird nicht nur die Rechenleistung erhöht. Auch der Aufwand um die Arbeit aufzuteilen und die Netz- und Netzwerkbelastung werden grösser. Das heisst, dass bei einer Verdoppelung der Nodeanzahl nicht mit einer Verdoppelung der Leistung zu rechnen ist. Diesen Benchmark haben wir mit verschiedenen Clusterzusammensetzungen durchgerechnet und die Statistik, die in Abbildung 5 auf Seite 24 und in der Tabelle 1 auf Seite 23 dargestellt ist, erstellt[15].

Das Histogramm (Abbildung 5, Seite 24) hat bei $x = 15$ ein Minimum, das heisst, dass mit 15 Nodes die Berechnungen am schnellsten erledigt werden. Mit mehr Nodes überwiegt der Aufwand der Aufteilung und die Netzwerkbelastung die zusätzliche Rechenleistung.

Wir erreichten mit unserem Beowulf eine Zeit von 19 Sekunden für den `skyvase`-Benchmark, was einem Platz unter den Top-50 der Bestenliste entspricht (siehe auch [25] und Abbildung 6, Seite 24).

Zur Sicherheit haben wir dieses Resultat mit `time` noch einmal überprüft und sind auf folgende Ausgabe gekommen:

```
s797010@lsvr03:~/pvm3/examples/vase > time pvmpov -i skyvase.pov +v1 +ft -x +mb\
   25 +a0.300 +j1.000 +r3 -q9 -w640 -H480 -S1 -E480 -k0.000 -mv2.0 +b1000
```

[15] Der verwendete Cluster bestand jeweils aus einem Server (PIII 600Mhz, 256MB) und einer Variablen Anzahl Nodes (PII 200Mhz, 64MB). Als Vergleich haben wir noch das Resultat mit der nicht parallelisierten Anwendung Povray in die Grafik (Abbildung 5) integriert (PIII 600MHz, 256MB).

Anzahl Nodes	benötigte Zeit
0 (nur Server)	65 sec
1	56 sec
2	47 sec
3	39 sec
4	35 sec
5	33 sec
6	31 sec
7	28 sec
8	28 sec
9	26 sec
10	25 sec
11	24 sec
12	22 sec
13	22 sec
14	21 sec
15	19 sec
16	22 sec
17	22 sec
18	30 sec
19	22 sec
20	22 sec

Tabelle 1: Wie lange brauchen wieviele Rechner für die Berechnung von `skyvase.pov`?

```
Time For Trace:    0 hours  0 minutes  19.0 seconds (19 seconds)
    Total Time:    0 hours  0 minutes  19.0 seconds (19 seconds)

real    0m19.396s
user    0m0.650s
sys     0m0.420s
```

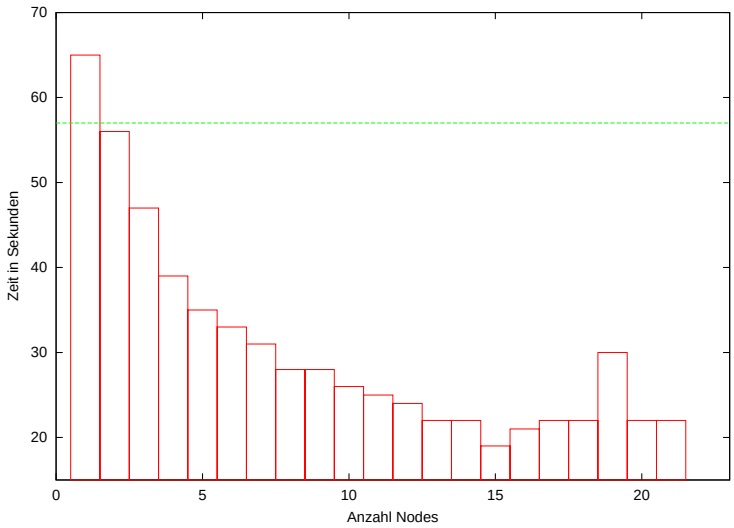


Abbildung 5: Wie lange brauchen wieviele Rechner für die Berechnung von `skyvase.pov`?

Abbildung 6: Die Rangliste des Skyvase.pov Benchmarks (siehe auch [25])

Teil VI
Appendix

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Add Disk-less Client Node, ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less/adcn.

[2] Asgard, http://www.asgard.ethz.ch/.

[3] Die Beowulf Legende, http://legends.dm.net/beowulf/.

[4] Beowulf-Projekt, http://www.beowulf.org.

[5] Beowulf at NASA/GSFC, http://beowulf.gsfc.nasa.gov/.

[6] Bladeenc, http://bladeenc.mp3.no/.

[7] Bootstrap protocol (Bootp), www.ietf.org/rfc/rfc0951.txt.

[8] Clarifications and Extensions for the Bootstrap Protocol, www.ietf.org/rfc/rfc1532.txt.

[9] DHCP Options and BOOTP Vendor Extensions, www.ietf.org/rfc/rfc1533.txt.

[10] CD-Paranoia-Homepage, http://www.xiph.org/paranoia/.

[11] Filesystem Hierarchy Standard, http://www.pathname.com/fhs/.

[12] Fraunhofer Institut, http://www.fraunhofer.de.

[13] Linux-Kernel, http://www.kernel.org.

[14] MP3, http://www.iis.fhg.de/amm/techinf/layer3/index.html.

[15] mp3pvm-Download, http://gamma.cbos.org/mp3pvm/.

[16] MPI, Message Passing Interface, http://www-unix.mcs.anl.gov/mpi/.

[17] PAM, The Linux-PAM System Administrators's Guide, http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-6.html.

[18] Povray-Homepage, http://www.povray.com.

[19] PVM-Homepage, http://www.epm.ornl.gov/pvm/pvm_home.html.

[20] PVMPOV-Homepage, http://www.luga.de/~flierl/pvmpov/.

[21] Ray Tracing: What is Ray Tracing?, http://www.povray.org/documents/rayfaq/part1/faq-doc-2.html.

[22] Set-up Disk-less Client Template, ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less/sdct.

[23] Short history of Beowulf, http://www.interex.org/hpworldnews/hpw907/05nt.html.

[24] POVBench: Skyvase, http://www.haveland.com/povbench/.

[25] skyvase Highscore, http://www.haveland.com/index.htm?povbench/index.htm.

[26] strike.pov, http://www.povray.org/preview/irtc-cd3/stills/19981231/view/strike.htm.

[27] Contest von povray.org, http://www.irtc.org/stills/1999-08-31.html.

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If

there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.