PVM - Parallel Virtual Machine

PVM ist ein Softwaresystem unter einem Unix-artigen Betriebssystem, das es ermöglicht, Programme auf mehreren Prozessoren koordiniert ablaufen zu lassen; dazu werden Meldungen (meistens mit Daten als Inhalt) zwischen ihnen ausgetauscht (Message Passing Paradigm). Die Kommunikation zwischen den auf einem oder mehreren Prozessoren ablaufenden Prozessen steuern die PVM-Dämonen (je einer pro Prozessor). Alle Prozessoren, auf denen miteinander kommunizierende PVM-Dämonen laufen, bilden eine *Parallele Virtuelle Maschine*. Die Dämonen laufen als Prozesse des Benutzers ab, der sie verwendet. Mehrere Benutzer, die am gleichen Prozessor je einen PVM-Dämon laufen lassen, definieren somit unterschiedliche Parallele Virtuelle Maschinen, gleichgültig, ob sie dabei den gleichen Satz physischer Prozessoren einbeziehen oder nicht. Diese Prozessoren können Teile eines Mehrprozessor-Systems sein oder sich auf mehrere Rechner, die über ein Netzwerk (derzeit nur auf TCP/IP-Basis) gekoppelt sind, verteilen (z.B. Rechner im Internet, gleichgültig, wo sie sich örtlich befinden).

Ein Benutzer kann z. B. - soweit er zu allen Rechnern, die er verwenden will, Zugang hat - eine Virtuelle Parallele Maschine aufbauen, die die Workstation auf seinem Schreibtisch, den Vektorrechner SNI S100/10 und mehrere Rechner vom Typ IBM RS/6000 am EDV-Zentrum der TU Wien, einen Vektorrechner Cray C90 in Los Alamos und mehrere Prozessoren des Parallelrechners Intel Paragon in Graz umfaßt. Seine Workstation kann gleichzeitig Bestandteil einer Parallelen Virtuellen Maschine sein, die sein Kollege aus den Workstations und PC-Rechnern unter Linux des eigenen Instituts zusammenstellt.

Portabilität

Multiprozessor-Programme, die mit Hilfe von PVM erstellt werden, sind portabel für den Einsatz auf ganz unterschiedlich zusammengestellten Virtuellen Parallelen Maschinen: Clustern von Monoprozessoren, Multiprozessoren mit verteilten oder gemeinsamen Hauptspeichern und Clustern aus allen diesen Rechnertypen. Zum Entwickeln und erstem Testen kann ein PVM verwendendes Programm auch zur Gänze auf einem einzigen Prozessor ablaufen.

Anwendungen

Die Ausgangsidee von PVM war der Wunsch, ein (teueres) Mehrprozessorsystem mit verteiltem Hauptspeicher durch das Zusammenspiel mehrerer vernetzter (meist kostengünstiger und ohnehin vorhandener) Rechner ohne besondere Hochleistungsarchitektur emulieren zu können, und so die gleiche Rechenleistung billiger zu erzielen. Dabei war von Haus aus vorgesehen, daß dabei die Zusammenfassung von Rechnern auch unterschiedlicher Hardware und Systemumgebung (jedoch Unix-artig) möglich sein soll.

Es bieten sich aber auch andere Anwendungsgebiete an. So ist es mit PVM möglich, Programme zu schreiben, die von einem Rechner ausgehend Teile auf Rechner mit besser geeigneter Architektur auslagern (z.B. kann im Zuge eines FE-Programms die Erstellung der Ausgangsmatrix auf einem Cluster aus Workstations erfolgen, die Matrixinversion jedoch auf einem - für diese Aufgabe besonders geeigneten - Vektorrechner erfolgen). Ein weiteres Beispiel ist die Kopplung eines schnellen (skalaren) Rechners mit einer grafischen Workstation und einem Rechner, der eine Maschine (z.B. einen Roboter) steuert und dabei auch Meßdaten erfaßt; mit solch einer Parallelen Virtuellen Maschine lassen sich auch numerisch aufwendige Steuerungsaufgaben bewältigen, die grafisch vorab simuliert werden müssen, um noch rechtzeitig Operatoreingriffe in den Ablauf zu ermöglichen.

Da auch zwei unabhängig auf verschiedenen Rechnern gestartete Programme im Zuge der Abarbeitung über PVM verbunden werden können, kann man langlaufende Programme so programmieren, daß der Benutzer, wann immer er es will, den Fortgang der Rechnungen von seiner Workstation aus verfolgen kann (eventuell über ein Visualisierungsprogramm) und bei offensichtlicher Fehlentwicklung (z.B. bei Iterationen) das Programm sauber beenden oder die weitere Berechnung beeinflussen kann. Dies ist über PVM auch bei der Abarbeitung eines Batch-Jobs möglich.

Aufbau einer *Parallelen Virtuellen Maschine*

Eine *Parallele Virtuelle Maschine* wird durch den Aufruf des PVM-Dämons (*pvmd3*) auf einem der beteiligten Rechner aktiviert, dieser Rechner wird dadurch zum Master; das bedeutet aber nicht, daß auch der Master-Prozeß eines PVM-Programms (entspricht dem Hauptprogramm eines konventionellen Prozesses) darauf ablaufen wird. Dieser Dämon startet dann auf allen anderen Rechnern, die zur *Parallelen Virtuellen Maschine* hinzugefügt werden, jeweils einen *pvmd3*. Alle diese Dämonen laufen als Benutzerprozesse ab, es sind daher in keiner Phase Superuser-Privilegien im Spiel.

Im interaktiven Betrieb ruft der Benutzer zum Initialisieren der *Parallelen Virtuellen Maschine* das Programm *pvm* auf. Auf der damit geschaffenen PVM-Konsole kann der Benutzer die *Parallele Virtuelle Maschine* schrittweise durch Hinzufügen weiterer Prozessoren aufbauen. Die PVM-Konsole bietet dem Benutzer darüber hinaus noch weitere Unterbefehle zur Überwachung und Steuerung. Sie kann mit dem Unterbefehl *quit* verlassen werden; dabei bleibt jedoch die *Parallele Virtuelle Maschine* weiter bestehen - auch bei Verlassen der interaktiven Sitzung. Bei weiteren Aufrufen von *pvm* - auch nach einem neuerlichen Login - meldet die Konsole dies sofort. Über die PVM-Konsole kann jederzeit die *Parallele Virtuelle Maschine* durch Hinzufügen oder Entfernen von Prozessoren verändert werden.

Die *Parallele Virtuelle Maschine* kann jedoch auch direkt mit dem Befehl **`pvmd3 cluster_datei&`** von der Unix-Eingabeaufforderung aus gestartet werden. Dabei ist *pvmd3* der Aufruf des PVM-Dämons und *cluster_datei* eine Textdatei, die die Netzadressen der beteiligten Rechner sowie zusätzliche Informationen enthält. Man kann jedoch auch ein PVM-Programm schreiben, das selbst die *Parallele Virtuelle Maschine* startet und sich dann daran anschaltet.

Batchsysteme wie DQS, die PVM unterstützen, bauen die *Parallele Virtuelle Maschine* für jeden Job vor dem Starten der Verarbeitung aus den zum Batch-Pool gehörenden Rechnern auf.

Starten von Programmen in der *Parallelen Virtuellen Maschine*

Über die Konsole kann jetzt ein Programm mit dem Unterbefehl *spawn* in einfacher oder mehrfacher Ausführung gestartet werden. Diese Programme laufen somit sofort in der *Parallelen Virtuellen Maschine* ab und können sofort miteinander kommunizieren. Jedes dieser Programme kann im Zuge seiner Verarbeitung aber weitere Programme in der *Parallelen Virtuellen Maschine* abspalten. Der Benutzer kann jetzt über sein Terminal jeden beliebigen Prozeß starten. Soll ein so gestartetes Programm in der *Parallelen Virtuellen Maschine* ablaufen, muß es sich an diese zuerst anschalten (siehe weiter unten).

Abbau der *Parallelen Virtuellen Maschine*

Der Unterbefehl *halt* der PVM-Konsole baut die *Parallele Virtuelle Maschine* ab und schließt die Konsole. Vor Verlassen einer interaktiven Sitzung sollte die *Parallele Virtuelle Maschine* immer abgebaut werden, da der PVM-Dämon ja den virtuellen Speicher in jedem der Rechner belastet - wie jeder andere Prozeß auch. Die Batchsysteme führen dies nach Abschluß eines Jobs automatisch durch. Die *Parallele Virtuelle Maschine* kann auch von einem in ihr laufenden Programm abgebaut werden. Damit wird auch jeder in ihr laufende Prozeß (inklusive des den Abbau initiierenden) abgebrochen.

Struktur eines PVM-Programms

Dieser Abschnitt soll einen kurzen Überblick über die wichtigsten PVM-Einfügungen in ein Fortran- oder C-Programm geben. Detailinformationen sind dem *PVM User's Guide and Reference Manual* zu entnehmen, der auch die `man`-Seiten der PVM-Unterprogramme enthält.

Auf allen Prozessoren laufen vollständige Prozesse mit eigenem Adreßraum ab; jeder von ihnen besteht also aus einem Haupt- und den von ihm direkt oder indirekt aufgerufenen Unterprogrammen, einschließlich der PVM-Routinen.

Im folgenden werden nur die zum Verständnis der Funktionsweise wichtigsten PVM-Funktionen kurz beschrieben; wird dabei ein Unterprogrammname angeführt, so handelt es sich dabei um den in einem C-Programm zu verwendenden Aufruf; bei FORTRAN-Programmen enthält der Aufruf (immer mit CALL) an Stelle des Unterstreichungszeichens „_“ den Buchstaben „f“. Alle PVM-Funktionen melden über spezielle Ergebniswerte, wenn ihre Durchführung nicht erfolgreich war und was der Grund dafür war.

Bevor die Dienste von PVM in Anspruch genommen werden, muß ein Prozeß sich mit *pvm_mytid* an die *Parallele Virtuelle Maschine* anschalten; als Ergebnis wird er zum PVM-Prozeß und erhält einen *task identifier* zugeordnet, dessen Wert die Funktion *pvm_mytid* an das rufende Programm zurückgibt. Diese Größe dient im folgenden zur eindeutigen Kennzeichnung des Prozesses innerhalb der *Parallelen Virtuellen Maschine*. Sie wird in allen Kommunikationen anderer Prozesse mit ihm verwendet.

Bei wiederholtem Aufruf dieser Funktion wird nur mehr der *task identifier* zurückgegeben. Dies geschieht auch beim Aufruf in einem Programm, das ohnehin über den PVM-Dämon gestartet wurde und damit automatisch in der *Parallelen Virtuellen Maschine* läuft.

Ein PVM-Prozeß kann nun weitere Prozesse in der *Parallelen Virtuellen Maschine* mit der Funktion *pvm_spawn* abspalten. Mit einem Aufruf können ein oder mehrere Instanzen des gleichen Prozesses initiiert werden. Der Aufruf enthält den (relativen oder absoluten) Pfad zu einer Datei, die das Programm in ablauffähiger Form enthält; in allen Rechnern, in denen so ein Prozeß gestartet werden soll, muß daher der Zugriff auf eine Datei dieses Namens (und selbstverständlich mit dem richtigen Programm) möglich sein. Die *task identifier* aller so gestarteten Prozesse werden an das rufende Programm zurückgegeben.

Die unter PVM abgespaltenen Prozesse erfahren über die Funktion *pvm_parent* den *task identifier* des Prozesses, von dem aus sie gestartet wurden. Somit können Vater- und Sohnprozeß einander direkt erkennen.

Zum Übertragen von Daten an einen anderen Prozeß in der *Parallelen Virtuellen Maschine* wird zuerst ein Datenpuffer eingerichtet; ein Prozeß kann zu einem Zeitpunkt mehrere Sendepuffer haben, aber nur einer ist jeweils aktiv. Das heißt, nur in ihn können Daten gepackt und nur er kann abgeschickt werden. Da eine *Parallele Virtuelle Maschine* aus Rechnern mit verschiedenen Datenformaten bestehen kann, umfaßt das Packen der Daten auch eine Konversion in das Austauschformat XDR; diese Konversion entfällt, wenn alle Rechner die gleiche Prozessorarchitektur haben oder wenn dies explizit verlangt wird.

Beim Versenden der Daten des Puffers wird dem Paket auch noch der *task identifier* des Zielprozesses und eine Kennung (*tag*) explizit und der *task identifier* des Ursprungsprozesses implizit mitgegeben.

Jeder Prozeß kann abfragen, ob an ihn gerichtete Datenpakete vorhanden sind, er kann sie unbedingt einlesen oder nur jene, die von einem bestimmten anderen Prozeß kommen und/oder mit einer bestimmten Kennung versehen sind. Die so empfangenen Daten stehen dann in gepackter Form in einem Eingabepuffer zur Verfügung. Es kann auch hiervon gleichzeitig mehrere geben, aber nur auf den gerade aktiven kann in jedem Prozeß zugegriffen werden. Um die Informationen weiterverarbeiten zu können, müssen die Daten noch entpackt werden; dabei muß der entpackende Prozeß wissen, welche Teile des Puffers er in welche Datentypen umwandeln muß, da eine Beschreibung des Datentyps nicht mitgeliefert wird (wie bei jeder Eingabe von einer ganz gewöhnlichen sequentiellen Datei).

Innerhalb eines Prozesses kann auch die Konfiguration der *Parallelen Virtuellen Maschine* abgefragt werden einschließlich der Prozessorarchitektur der teilnehmenden Rechner, es können zusätzliche Rechner aufgenommen oder ausgeschieden werden.

Jeder Prozeß sollte sich vor seiner Beendigung von der *Parallelen Virtuellen Maschine* verabschieden (*pvm_exit*). Dies kann natürlich auch schon ab dem Zeitpunkt geschehen, wo der Prozeß nur mehr auf sich selbst gestellt weiterarbeiten kann.

Dynamische Prozeßgruppen

PVM-Prozesse können Gruppen definieren, in die weitere Prozesse dynamisch aufgenommen werden können und aus denen sie sich auch wieder entfernen können.

Eine wichtige Anwendung dieses Konzepts ist die Möglichkeit, die Abarbeitung von Prozessen zu synchronisieren. Jeder Prozeß einer Gruppe, der die Funktion *pvm_barrier* aufruft, bleibt an dieser Stelle mit der Verarbeitung solange stehen, bis die von ihm gewünschte Anzahl von Prozessen derselben Gruppe dieselbe Funktion aufgerufen hat.

Prozesse, die unabhängig voneinander gestartet wurden, können sich an eine *Parallele Virtuelle Maschine* anschalten (einer von ihnen kann sie sogar aufbauen), sich dort einer dynamischen Gruppe anschließen und dadurch miteinander kommunizieren. Eine Anwendung davon ist die schon zu Beginn des Artikels angesprochene Möglichkeit, Batch-Prozesse semi-interaktiv zu überwachen und zu steuern.

Bestehende PVM-Installationen am EDV-Zentrum

Am EDV-Zentrum gibt es einen Cluster aus 9 Workstations IBM RS/6000, der zum Entwickeln und Austesten eingerichtet wurde, der jedoch auch im geringen Umfang einen Produktionsbetrieb ermöglicht.

Es stehen meist mehrere (teilweise untereinander inkompatible) Versionen von PVM im Verzeichnis */usr/local/* bereit. Auf die zu jedem Zeitpunkt empfohlene Version kann über das Verzeichnis */usr/local/pvm3* zugegriffen werden.

Zum Redaktionszeitpunkt war die aktuelle Version vom PVM 3.3.6. Versionen, die in den ersten beiden Zahlen übereinstimmen, sind aufwärtskompatibel. PVM-Funktionen eines Programms können auch aus Bibliotheken mit höherer dritter Zahl bedient werden. Die Kommunikation zwischen Programm und lokalem PVM-Dämon und zwischen Dämonen auf verschiedenen Rechnern ist auch sichergestellt. Versionen mit höherer dritter Zahl enthalten Korrekturen, unterstützen mehr Rechnerplattformen und bieten manchmal neue Funktionen. Versionen mit unterschiedlichen ersten beiden Ziffern sind nicht kompatibel.

Auch am Vektorrechner SNI S100/10 wurde PVM für Testzwecke installiert. Zum Laden von Programmen werden auch noch die Bibliotheken *libsocket.a* und *libnsl.a* - in dieser Reihenfolge - benötigt. Anwendungsbeispiele sind: Überwachung von Batch-Jobs, Auslagern einzelner Teilaufgaben, die die Architektur optimal nutzen.

Dokumentation in Form von PostScript- und gewöhnlichen Textdateien stehen am Master-Rechner des Clusters *cl6k0.tuwien.ac.at* im Verzeichnis */usr/local/pvm3.3/pvm3/doc* zur Verfügung; derzeit ist jedoch an diesem Rechner kein anonymer FTP-Server eingerichtet.

Installation

Die Installation und Nutzung von PVM erfordert keine Superuser-Privilegien. Jeder Benutzer kann die benötigten Dateien von seinem Verzeichnis aus verwenden. Es ist jedoch sinnvoll, auf Rechnern, die von mehreren Benutzern verwendet werden, PVM in einem allgemein zugänglichen Verzeichnis anzulegen (z.B.: */usr/local*).

Die erforderlichen Dateien können von der zentralen Verteilerstelle (netlib.org) über anonymous ftp, mit Hilfe des X Window-Programms xnetlib (ebenfalls von dort erhältlich) oder über World-Wide Web (<*URL: http://www.eece.ksu.edu/pvm3/pvm3.html*>) abgeholt werden. Es ist geplant, ab Sommersemester 1995 die aktuelle Version von PVM und die entsprechenden Dokumentationen auch über das Informationssystem der TU Wien lokal zugänglich zu machen.

Vorhandene PVM-Implementierungen

Die derzeit gültige Liste der unterstützen Rechnerarchitekturen umfaßt unter anderen:

DEC Alpha/OSF (Mono- und Multiprozessor)
DEC Mips und Microvax
Sequent Balance
CM2
CM5
Convex
Cray
HP 9000
Intel Hypercube, IPSC/2 und Paragon
IBM RS/6000 und Power-4
NeXT
SGI (Mono- und Multiprozessor (letztere ab IRIS 5.0))
Sun4 (Mono- und Multiprozessor) und unter Solaris
SNI/Fujitsu unter UXP/M.
PC (386, 486, Pentium) unter Linux, SCO Unix, BSDI, 386BSD, NetzBSD, FreeBSD

Für DOS und Windows, Windows NT sowie OS/2 gibt es aus technischen Gründen keine PVM-Implementierungen.

Ein PVM-Cluster kann also auch aus vielen gekoppelten PCs aufgebaut werden (der Supercomputer des armen Mannes/Instituts).

Auskünfte, Kurse

Weitere Auskünfte über PVM können beim Autor eingeholt werden. Es ist geplant, im Sommersemester 1995 einen Kurs über PVM-Programmierung abzuhalten. Interessenten werden gebeten, Herrn Willy Weisz zwecks Terminkoordination zu kontaktieren.

Dokumentation

Neben den oben erwähnten Dateien am Rechner *cl6k0.tuwien.ac.at* gibt es im World-Wide Web Dokumentation über PVM:

An Introduction to PVM Programming
<*URL: http://www.epm.ornl.gov/pvm/intro.html*>

The New PVM Web Page
<*URL: http://www.eece.ksu.edu/pvm3/pvm3.html*>

Es gibt auch bereits ein Buch von den Schöpfern von PVM:
A User's Guide and Tutorial for Network Parallel Computing, Al Geist et al., MIT Press, ISBN 0-262-57108-0.
Eine PostScript-Datei dieses Buches kann vom ftp-Server *netlib2.cs.utk.edu* unter dem Namen *pvm3/book/pvm-book.ps* geholt werden.
Die html-Version ist über die Adresse <URL: *http://www.netlib.org/pvm3/book/pvm-book.html*> abrufbar.

Willy Weisz