Der DFN-RPC, ein Remote Procedure Call Tool

Ralf Allrutz, Rolf Rabenseifner
25. Februar 1992

Der DFN-RPC, ein Remote Procedure Call Tool, wurde entwickelt, um wissenschaftlich technische Anwendungsprogramme zwischen Workstation und Compute Server verteilen zu können. Anhand des DFN-RPCs wird eine Einführung in die RPC-Technologie gegeben. Die Merkmale des DFN-RPCs werden vorgestellt. Am Beispiel eines Anwendungsprogrammes werden die wesentlichen Aspekte bei der Verteilung erörtert. Alle Prozesse laufen unter User-Account.

1 Einleitung

Das DFN Remote Procedure Call Tool (kurz DFN-RPC) wurde vom Rechenzentrum der Universität Stuttgart im Auftrag des Deutschen Forschungsnetzes (DFN-Verein) in Zusammenarbeit mit vier Pilotanwendern[1] entwickelt.

Der DFN-RPC ermöglicht es dem Wissenschaftler und Ingenieur, eigene Anwendungsprogramme zwischen seiner Workstation und einem entfernten Höchstleistungsrechner verteilt auszuführen. Schnelle Simulationsberechnungen auf dem Compute Server, graphische Visualisierung auf der Workstation vor Ort und gleichzeitige Interaktionen ermöglichen einen neuen Arbeitsstil und können die Entwicklungs- und Forschungszyklen verkürzen.

2 Der DFN-RPC

R. Rabenseifner[2]

2.1 Das Prinzip

Remote Procedure Calls sind ein mächtiges, aber auch einfaches Mittel, um Programme zwischen zwei Rechner zu verteilen.

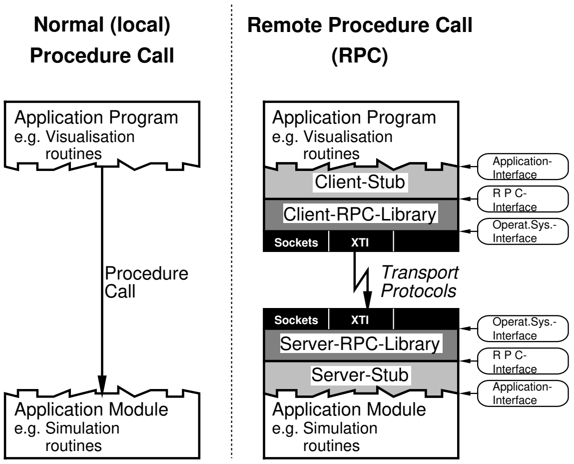

Abbildung 1: Vom „Local“ zum „Remote Procedure Call“

In Abb. 1 werden die Simulationsroutinen im unteren Bildteil auf einen entfernten Rechner ausgelagert. Das Remote Procedure Call Tool stellt die grauen Blöcke. Die Stubs (dt. Stummel) stellen auf der Client-Seite Entries zur Verfügung, die zwar die gleichen Namen und Parameterliste wie die Simulationsroutinen haben, aber nicht deren Procedure-Rümpfe enthalten, sondern nur einen kurzen Code, um die Parameterliste über das Netz zum Server-Stub zu übertragen. Hierbei wird nur eine Procedure-Nummer, alle Feldlängen zur Allocierung der Parameterliste beim Server und alle Input-Parameter geschickt, s.a. Abb. 2. Der Server-Stub baut dann die Parameterliste wieder auf und ruft die eigentliche Simulationsroutine auf. Nach Abschluß der Simulationsroutine werden die Output-Parameter wieder

---

[1] Alfred-Wegener-Institut, Bremerhaven (AWI); Institut für Kernenergetik und Energiesysteme, Stuttgart (IKE); Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Stuttgart (ISD); Lehr- und Forschungsbereich theoretische Astrophysik, Tübingen (TAT)

[2] email: rabenseifner@rus.uni-stuttgart.de

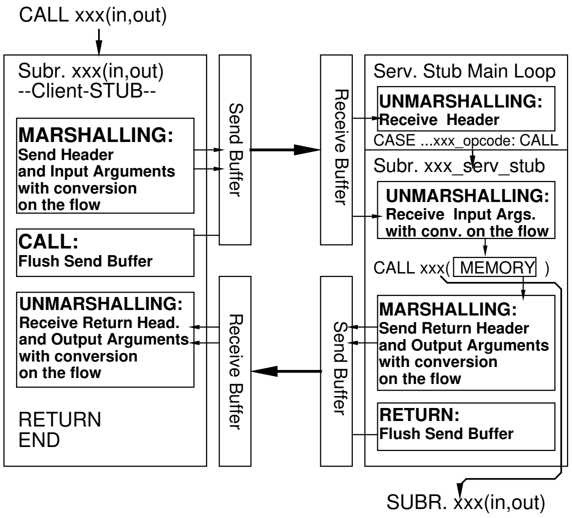

Abbildung 2: Das „Marshalling“, die Aufgabe der Stubs

zum Client zurück gesandt. Beim Senden und Empfangen der Input- bzw. Output-Parameter werden diese automatisch in die entsprechende Zahlendarstellung umgewandelt. [NEL 81]

Typischerweise werden mehrere via RPC zu rufende Routinen, die zu einem Kontext gehören, in einem Prozeß zusammengefaßt.

2.2 Marshalling

Das Versenden bzw. Empfangen der Parameter wird Marshalling bzw. Unmarshalling genannt. Da die Rechner intern unterschiedliches Byteordering und unterschiedliche Realzahlformate benützen, müssen die Zahlen beim Transfer umgewandelt werden. Die wesentlichen Strategien zur Konvertierung sind:

1. Übertragung in einem fest vorgeschriebenen Format, z.B. IEEE big-endian. Dies ist die einfachste Lösung. Doch wenn die beteiligten Rechner ein anderes, aber identisches internes Format benützen (z.B. zwei VAXen), dann müssen die Daten grundlos zweimal umgewandelt werden. Vgl. z.B. SUN-RPC [SUN 90].
2. Das Transfer-Format ist frei wählbar.
3. Übertragung im Format des Senders. Diese Lösung hat bei einem 64Bit Compute Server den Nachteil, daß eine Realzahl unnötigerweise mit 8 Byte über das Netz zur Workstation gesandt wird und erst dort in eine 4 Byte Realzahl umgewandelt wird. Vgl. OSF/DCE [DCE 90].

Beim DFN-RPC wurde das 2. Verfahren realisiert, wobei als Voreinstellung *IEEE big-endian* gewählt wurde.

2.3 Interface Definition

Die Stubs sind anwendungsspezifisch, müssen aber nicht von Hand programmiert werden, da sie anhand einer Interface Definition automatisch generiert werden können. Die Interface Definition besteht im wesentlichen aus den Procedure-Köpfen. Da beim RPC das Call-by-Referenz durch ein Übertragen der Daten ersetzt werden muß (Call-by-In-Copy-and-Out-Copy), sind zusätzliche Informationen nötig,

- z.B. ist ein In-Copy und/oder Out-Copy nötig oder handelt es sich um einen Workspaceparameter,
- wie kann eine Array-Länge zur Laufzeit ermittelt werden,
- kann ein Parameter mit einer geringeren Genauigkeit übertragen werden?

Der DFN-RPC wurde speziell für Fortran-Anwendungen entwickelt. Er kann aber auch bei C-Programmen eingesetzt werden.

```
C*[RPI_BEG]
      SUBROUTINE exchange (n,a,b,wrk)
      INTEGER  n       [IN]
      REAL     a(n)    [IN,OUT]
      REAL     b(n)    [IN,OUT]
      REAL     wrk(n)  [DUMMY]
C*[RPI_NON]
      DO 1 i=1,n
         wrk(i) = a(i)
         a(i)   = b(i)
         b(i)   = wrk(i)
    1 CONTINUE
      END
C*[RPI_END]
```

Abbildung 3: RPC Interface Definition, ein Beispiel

Abb. 3 zeigt ein kurzes Beispiel. Hier wurden die sog. RPC-Qualifier in das Programm integriert. Dadurch kann das Single-Source-Prinzip erhalten werden, d.h. daß es nicht notwendig ist, eine Kopie der Procedure-Köpfe auf einem besonderen Interface-Definition-File zu halten.

```
C*[RPI_BEG]
      SUBROUTINE sss(n,a,b,c)
      INTEGER  n [IN]
      REAL  a(*) [MAXINDEX = n*(n+1),
     +            IN =     1 : n*n
     +            OUT= n*n+1 : n*n+n]
      REAL b(3,*)[MAXINDEX = n+1,
     +            IN = (1:3, 1:n),
     +            OUT= (1:3, n+1:n+1)]
      DOUBLEPRECISION c(n)[TRANSFER*4]
C*[RPI_END]
```

Abbildung 4: Komplexe Qualifier

Abb. 4 zeigt weitere Qualifier. Mit MAXINDEX kann die Länge variabler Felder festgelegt werden. Mit IN und OUT können die Übertragungsbereiche detailliert festgelegt werden. Mit TRANSFER kann die Übertragungsgenauigkeit verringert werden.

Prinzipiell ist es möglich, die Qualifier nur als Kommentar-Direktiven in Form von speziellen Fortsetzungszeilen einzustreuen:

```
      INTEGER n
C*RPC+          [IN]
```

Neben dem Marshalling normaler Datentypen unterstützt der DFN-RPC auch:

- Alternative Rücksprünge,
- Callback, d.h. eine mittels RPC gerufene Server-Routine kann wiederum via RPC eine Routine im Client-Prozeß aufrufen,
- Entry-Statements mit unterschiedlichen Parameterlisten,
- Zusammenfassen mehrerer Server-Routinen zu einem Modul,
- separate Stubgenerierung für verschiedene Module,
- synchrone, asynchrone (nicht blockierende) und asynchron gepufferte RPCs.

S.a. nachfolgende Kapitel.

2.4 Stub-Generator

Abb. 5 zeigt das Kompilations-Schema für Single-Source-Programme. Der Stub-Generator trennt die Source in Client- und Server-Teil und erzeugt die Stubs.

Es empfiehlt sich, daß der Anwendungsprogrammierer die Platten, auf denen die aufgeteilte Client- und Server-Source und die generierten Stubs liegen, auf dem Client- und auf dem Server-Rechner mountet (z.B. mit NFS).

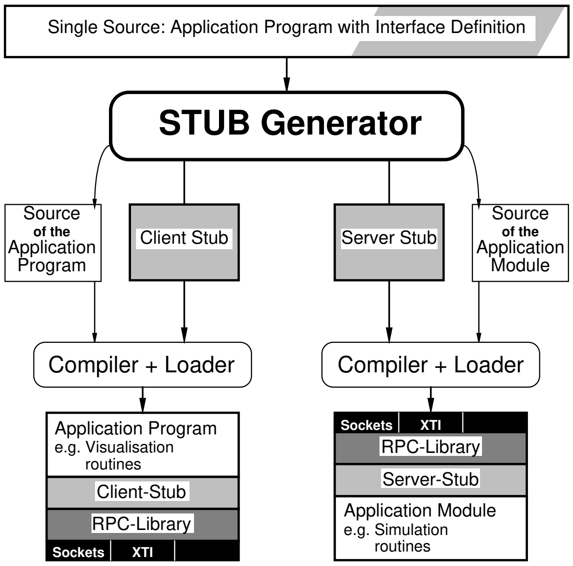

Abbildung 5: Kompilations-Schema für Single-Source-Programme

2.5 Separate Stubgenerierung

Die Interface Definition kann auch getrennt vom Anwendungsprogramm erstellt werden. Die Procedure-Rümpfe entfallen, wie in Abb. 4.

Einsatzgebiete sind

- entfernte Bibliotheken, d.h. man macht existierende Bibliotheken mittels RPC von einem entfernten Rechner aus zugänglich; dies kann eine numerische Library auf einem Vektorrechner (Compute Server), aber auch eine Visualisierungslibrary auf der Workstation (d.h. Client=Supercomputer, Server=Workstation) sein.

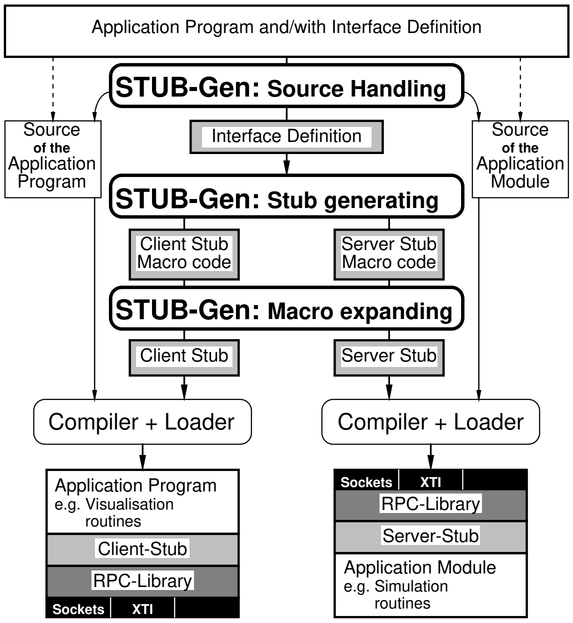

Abbildung 6: Das detailierte Schema der Stub-Generierung

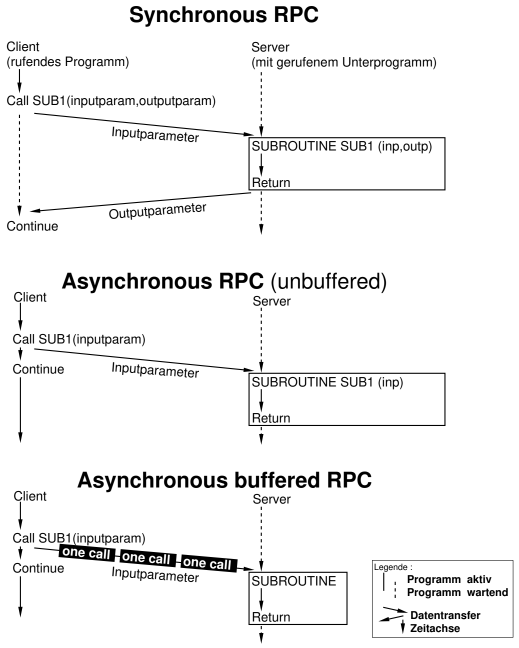

Abbildung 7: Synchronisierung

- modifizierte oder von Hand geschriebene Stubs: Die Stub-Generierung besteht logisch aus drei Phasen. Nach der Aufspaltung der Source in Client- und Server-Teil werden die Stubs in portablem Macrocode generiert. Dieser wird anschliessend in einen kompilierbaren Code umgewandelt. In Spezialfällen, wenn die Mächtigkeit der Interface Definition Language nicht ausreicht, kann der Macrofile vom Anwendungsprogrammierer erstellt oder geändert werden.

Abb. 6 zeigt das generelle Schema der Stubgenerierung mit dem DFN-RPC.

2.6 Synchronisierung

Normalerweise ist ein RPC vollständig synchronisiert. Wie in Abb. 7 oben dargestellt, wartet der Server bis der Client einen RPC Auftrag schickt. Nach der Datenübertragung der Input-Parameter wartet der Client bis die Server-Routine beendet ist und die Output-Parameter zurückübertragen sind. Dies entspricht der üblichen Semantik eines Unterprogramm-Aufrufs.

Mit diesen *synchronen* RPCs kann man aber die Parallelität der beteiligten Prozessoren nicht ausnützen.

Der DFN-RPC sieht daher zwei weitere Synchronisierungsformen vor: den *asynchronen* und den *asynchron gepufferten* RPC. Die Anwendungsproceduren müssen hierbei ohne Output-Parameter sein. Bei beiden Formen bekommt das Client-Anwendungsprogramm nach dem Marshalling der Input-Parameter die Kontrolle zurück. Es wird garantiert, daß die Server-Routinen, in der durch die RPCs des Client-Programms bestimmten Reihenfolge, aufgerufen werden. Auf grund der Flußkontrolle (das Netz kann als eine Pipe mit endlicher Puffergröße betrachtet werden) kann der Client blockiert werden, bis der Server-Prozeß die Input-Daten empfangen hat. Von einer mit asynchronem RPC aufgerufenen Server-Routine aus sind keine Callbacks zulässig.

Wenn Client- und Server-Prozeß keine gemeinsame Daten besitzen (kein gemeinsames Memory, keine gemeinsame Files auf einem Fileserver, etc.), dann erfüllen die asynchronen RPCs auch die Semantik eines normalen Procedure-Aufrufs.

2.7 Prozeßstart

Beim DFN-RPC sind bei einer verteilten Anwendung mindestens zwei Prozesse beteiligt, der Client-Prozeß, welcher das Hauptprogramm enthält und ein Server-Prozeß, welcher die mittels RPC aufrufbaren Unterprogramme enthält. Es sind auch mehrere Server-Prozesse möglich.

Der DFN-RPC ermöglicht, daß alle beteiligten Prozesse unter User-Account laufen.

Die beteiligten Prozesse können alle von Hand gestartet werden. Man kann aber auch ausgehend von einem Prozeß, im folgenden *Master*[3] genannt, die weiteren beteiligten Prozesse starten.

Der DFN-RPC enthält hierzu einige prototypische Beispiele. Es können beliebige existierende Startmechanismen, wie z.B. rsh oder nqs eingesetzt werden. Prinzipiell wird die für den Datentransport notwendige Transport-Verbindung (Connection) vom Slave zum Master aufgebaut, s. Abb. 8. Der Master-Prozeß eröffnet einen Port (Transport Service Access Point, TSAP) und teilt dem Slave beim Prozeßstart die Portnummer mit. Dieser kann dann die Verbindung zum Master aufbauen.

Bei dieser Lösung benötigt man keinen sog. Portmapper[4] und keine global bekannte Portnummer[5] oder Anwendungsnummer[6].

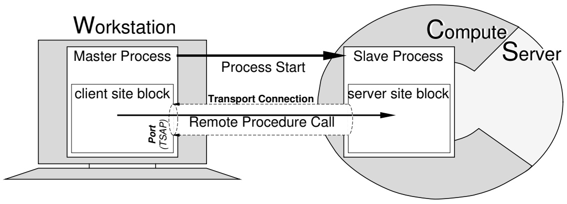

Abbildung 8: Prozeßstart und Verbindungsaufbau

2.8 Protokoll und Semantik

Beim DFN-RPC kann man als Transport-Protokoll wahlweise TCP/IP, Ultra TP4 oder ISO-OSI TP4 benützen, s. Abb. 9, [TAN 89].

Der Anwender wählt einen Protokollstack nur durch eine entsprechende Adressenangabe aus. Die Protokollstacks selbst unterscheiden sich stark in der Interkonnectivität, Transferrate und Roundtriptime[7].

Die verbindungsorientierten Transport-Protokolle garantieren eine sog. Exactly-Once Semantik, bzw. genauer, alle sind RPCs erwartungsgemäß durchgeführt worden, wenn der Anwendung vom Verbindungsaufbau bis zum Verbindungsabbau vom DFN-RPC kein Fehler berichtet wurde, Dies gilt auch für die asynchronen RPCs.

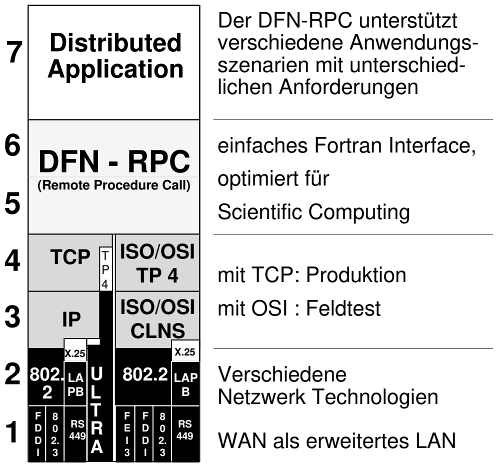

Abbildung 9: DFN-RPC im ISO-7-Schichten-Modell

3 Verteilte Anwendungen

Abb. 11 zeigt typische Szenarien, wie Anwendungen im Bereich der wissenschaftliche-technische Simulation und Visualisierung zwischen Workstation und Compute Server verteilt werden können.

Das Programm ist in einen Compute-Teil, einen Visualisierungsteil und eine Graphics-Library aufgeteilt. In Abb. 11 enthält der

---

[3] Der Master-Prozeß kann der Client-Prozeß, aber auch ein Server-Prozeß sein

[4] Deamon unter Root-Account dem alle Ports von den Anwendungsprozessen mitgeteilt werden, vgl. OSF/DCE [DCE 90]

[5] vgl. SUN-RPC [SUN 90]

[6] z.B. der Universal Unique Identifier UUID bei OSF/DCE

[7] Zeit, die ein Remote Procedure Call mit einer leeren Server-Routine benötigt

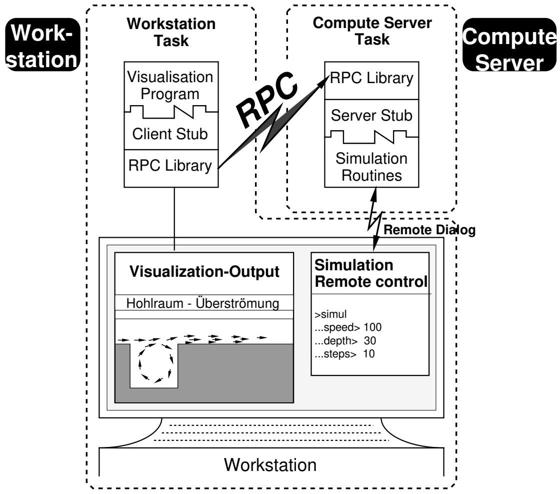

Abbildung 10: Überströmung eines Hohlraumes

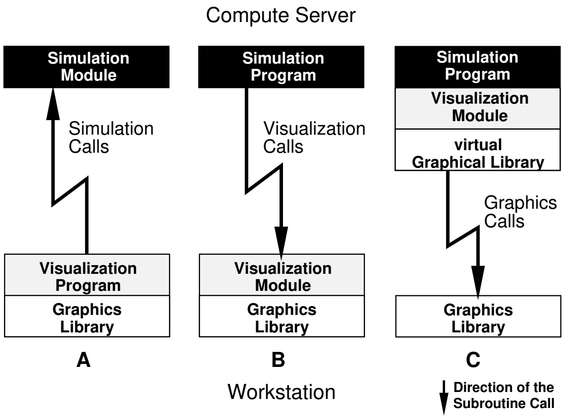

Abbildung 11: Szenarien

Compute-Teil die Simulationsrechnung - dann spricht man von „verteilter Simulation und Visualisierung“, er kann aber auch aus einem rechenintensiven Teil der Visualisierung bestehen - dann handelt es sich um eine „verteilte Visualisierung“.

Im Fall A ist das Hauptprogramm auf der Workstation, d.h. sie ist der Client, und es macht Remote Procedure Calls zum Compute-Teil auf dem Compute Server.

Im Beispiel in Abb. 10 wird hierzu eine verteilte Visualisierung gezeigt. Die Integration der Strom- und Streichlinien zur Darstellung einer Hohlraum-Überströmung wird auf dem Compute Server auf der Basis eines 30 Mbyte großen zeitabhängigen Strömungsfeldes berechnet. Diese Integrations-Rechnungen wurden auf den Compute Server ausgelagert, da die Visualisierung animiert stattfinden muß und daher trotz der 30 MByte kein Paging stattfinden darf. Der Wissenschaftler an der Workstation kann nun interaktiv Farbsonden setzen und die Strömungssimulation mit den von ihnen ausgehenden Streichlinien untersuchen. Außerdem kann er über ein Menü verschiedene Darstellungsformen wählen, z.B. kann er ein Sägmehl einstreuen und die resultierenden Bahnlinien sofort verfolgen. Er kann also graphisch-interaktiv, unter Einbeziehung eines Supercomputers, die Ergebnisse der Strömungsberechnung untersuchen.

Im Fall B liegt das Hauptprogramm auf dem Compute Server, d.h. er ist nun der Client, und die Workstation übernimmt die Rolle des Servers (bzgl. der RPCs).

Zur Realisierung der Fälle A und B benötigt man ein Remote Procedure Call Tool, das einfach zu programmieren ist. Beim Programm-Entwurf sollte man darauf achten, daß man das Interface, an dem das Programm verteilt wird, so wählt, daß ein Minimum an Daten und synchronen RPCs übers Netz gehen.

Im Fall C wird das komplette Anwendungsprogramm auf dem Compute Server ausgeführt. Nur die Graphics Library befindet sich auf der Workstation.

In den meisten Anwendungsfällen wird man hier X Window, PEX[8] oder DGL[9] verwenden. Wenn man eine beliebige Graphics Library einsetzen will, kann man dies mittels eines RPC Tools realisieren.

Die meisten Remote Procedure Calls können hier asynchron gepuffert realisiert werden.

Der Einsatz einer verteilten Graphics Library ist vor allem für den Anwendungsprogrammierer sehr komfortabel, da er quasi nur die Graphics Library durch die Remote Graphics Library ersetzen muß.

Im nächsten Kapitel wird ein komplexeres Beispiel vorgestellt.

---

[8] PHIGS in X Window

[9] Distributed (Iris) Graphics Language

4 Verteilte Anwendungen in der theoretischen Astrophysik

R. Allrutz[10]

4.1 Motivation

Das Grundproblem der Astrophysiker ist, daß sie mit den Objekten ihrer obskuren Begierde keine Experimente anstellen können. Die einzigen Daten, die zur Überprüfung theoretischer Modelle herangezogen werden können, stammen aus Beobachtungen des Alls mit allen erdenklichen (Geld-)Mitteln. Die theoretischen Modelle entziehen sich oft einem analytischen Zugang, sodaß zur Präparation von Observablen, also den Beobachtungsgrößen, die Mitwirkung von Computern unerläßlich ist. Oft will man die dynamische Simulation zwei- oder dreidimensional visualisieren, man denke z.B. an die Simulation eines Raumsondenfluges durch das Planetensystem. Dabei stellen diese Probleme zum einen oft große Anforderungen an die numerische Leistungsfähigkeit des benützten Rechners, zum anderen wird durch die geforderte Visualisierung ein guter grafikfähiger Arbeitsplatz benötigt. Diese Anforderungen suggerieren bereits eine *Zweiteilung* des Programmes, in dem der numerische Teil auf einem dafür geeigneten Rechner läuft und der Visualisierungsteil entsprechend.

Beobachtbar auf der Erde sind von entfernten Sternen „nur“ Spektren und Lichtkurven. Ziel der numerischen Simulationen ist es, diese Beobachtungsgrößen aus einem konsistenten Modell zu gewinnen. An Modellen zur Erklärung der verschiedensten Phänome mangelt es nicht, aus der Physik wird von der Geometrisierung des Raumes (Allgemeine Relativitätstheorie) bis zur Quanten-Elektrodynamik alles benötigt. Diese Spannweite bringt zwangsläufig eine gewisse Spezialisierung mit sich, und so befaßt sich die Arbeitsgruppe, in der wir den DFN-RPC einsetzen, mit der Teilchensimulation von Scheiben.

4.2 Scheiben

Scheiben sind im All auf allen Längen- und Zeitskalen sehr verbreitet, man denke nur an die Saturnringe, an die Ebene, in der die Planeten um die Sonne laufen, oder an die Spiralstruktur vieler Galaxien. Auch Sterne im Endstadium ihres Lebens, das sind weiße Zwerge, Neutronensterne und schwarze Löcher, können von Scheiben umgeben sein. Viele Sterne existieren in Doppelstern-Systemen, von denen meist der kleinere eine solche Scheibe ausbildet, indem er vom größeren Masse abzieht. Solche rotierenden Doppelstern-Systeme sind gewissermaßen kosmische Leuchttürme, deren komplexes Lichtkurven-Verhalten aus verschiedenen Aspekten theoretisch untersucht wird. Eine weitere wichtige Scheibenart sind die protoplanetaren Scheiben, die als Grundlage für Modelle zur Entstehung des Planetensystems dienen. Im Laufe dieses Projektes befaßten wir uns vor allem mit der Teilchensimulation von protoplanetaren Scheiben.

4.3 Teilchensimulation

Die Teilchensimulation versucht, das dynamische Verhalten eines Systems aus mikroskopischer Sicht zu simulieren, also aus der Sicht der einzelnen Elemente, aus denen sich das Gesamt-Phänomen zusammensetzt. Der andere Ansatz ist der hydrodynamische, in dem die Mittelung über die mikroskopischen Teile schon vorweggenommen ist und nur noch Differential-Gleichungen für kontinuierliche Felder übrig sind.

Ein Teilchen in solch einer Simulation wird charakterisiert durch seinen Ort, seine Geschwindigkeit und durch weitere verschiedene skalare Größen wie Temperatur, Druck und Dichte. Für diese Größen gibt es Bewegungsgleichungen, die wir mit den Adam's Inter- und Extrapolations-Formeln integrieren, wobei die meiste numerische Arbeit allerdings darin besteht, die Kräfte aus den Orten und den Geschwindigkeiten zu berechnen. Typische Teilchenzahlen für die Simulation liegen zwischen 2000 und 10000. Damit liegen bei 2-D-Simulationen für einen Zeitschritt bei `DOUBLE PRECISION`-Genauigkeit zwischen 100kB und

---

[10] email: allrutz@tat.physik.uni-tuebingen.de

500kB Daten vor. Die Visualisierung auf einem 1280*1024 Pixel Bildschirm mit 256 Farben erfordert bereits etwa 1,3MB Information.

4.4 Vor dem DFN-RPC ...

wurden auf der Cray-2 des RUS die Integrationen als Batch-Job ausgeführt und die Daten anschließend formatiert via Belwue an unsere Workstations in Tübingen übertragen, wo sie nachträglich visualisiert, gefilmt und archiviert werden. Dies hat verschiedene Nachteile:

- Die Dateien liegen auf den Platten des benützten Rechners und müssen mehr oder weniger umständlich transferiert werden.
- Das zu übertragende Datenvolumen ist größer als eigentlich nötig, weil die beteiligten Rechner unterschiedliche Zahldarstellungen benützen und deswegen die Daten formatiert abgespeichert werden müssen.
- Es gibt keine Kontrolle über die laufende Simulation, schlecht gewählte oder falsche Parameter fallen erst bei der visuellen Kontrolle hinterher auf.

Trotz dieser Nachteile gab es jedoch auch einen Vorteil: Bis auf die Zeit der Datenübertragung muß immer nur *ein* Rechner in Betrieb sein.

4.5 Mit dem DFN-RPC ...

verschwinden alle erwähnten Nachteile:

- Die Daten können auf den lokalen Platten abgelegt werden, ohne daß der Benutzer explizit den Netzwerk-Transfer bedienen muß.
- Das Datenvolumen auf dem Netz entspricht der gewählten Transfer-Kodierung, die ist in jedem Fall binär und trägt damit nicht unnötig zur Netzwerkbelastung bei. Die möglicherweise nötige Datenkonversion wird transparent vom DFN-RPC durchgeführt.
- Es ist eine ständige Kontrolle der Simulation möglich, weil jeder Zeitschritt lokal visualisiert werden kann.

Es schleicht sich aber ein neuer Nachteil ein: beide beteiligten Rechner und das vermittelnde Netzwerk müssen für die gesamte Laufzeit des Programmes verfügbar sein. In der Praxis ist dies allerdings kein Problem, da die Verfügbarkeit dieser Komponenten doch sehr hoch ist.

4.6 propla

Bei dem Programm `propla` handelt es sich um eine Teilchensimulation von protoplanetaren Scheiben. Der numerische Teil ist in Fortran geschrieben und wird von einer grafischen Oberfläche aufgerufen, die in C geschrieben ist und sowohl X-Windows/MOTIF als auch Silicon Graphics GL verwendet. In der bestehenden Implementierung läuft der Fortran-Teil als ganzes auf dem Compute-Server, die Oberfläche macht pro Zeitschritt einen Remote Procedure Call.

An dieser Stelle zeigt sich ein wichtige Erfahrung, die wir beim Einsatz des DFN-RPC machten. Wenn das zu verteilende Programm als Workspace einen oder mehrere `COMMON`-Blocks enthält, ist es einfacher, das gesamte Programm als Unterprogramm zu formulieren, anstatt die durch Profiling als CPU-Zeit-Fresser enttarnten Funktionen zu extrahieren und auf den Compute-Server zu bringen. Die Existenz globaler Variable stört natürlich das Konzept jeder RPC-Implementierung, weil die beteiligten Rechner ihren Speicher ja nicht teilen und somit kein gemeinsamer Zugriff auf solche Variablen möglich ist.

Typische Rechenzeiten pro Visualisierungs-Schritt liegen im Sekundenbereich. Für diese Zeiten kann die Oberfläche nicht auf Benutzeraktionen reagieren. Dies sind Zeiten, die in einer grafisch-interaktiv orientierten Umgebung bereits als unzumutbar lang empfunden werden. Zur Lösung dieses Dilemmas bieten sich zwei Alternativen an.

1. Durch die Einführung eines explizit asynchronen RPC-Call (und damit einer Erweiterung der `FORTRAN`-Semantik) könnte

man sich die inheränte Parallelität der zwei beteiligten Rechner nutzbar machen. Dabei müßte die Oberfläche dann irgendwie von dem Ende der Remote Procedure Kenntnis erhalten.

2. Der lokale Prozeß wird aufgeteilt in zwei Prozesse, die ihre Daten über ein *shared memory* austauschen. Einer der beiden macht den synchronen (blockierenden) RPC und der andere kann in der Zwischenzeit auf Benutzeranforderungen reagieren. Für die Kommunikation zwischen beiden Prozessen können die UNIX-üblichen Mechanismen (Pipes, Signale, Sockets, Semaphore) verwendet werden.

Das zweite Szenarium ist verwirklicht und soll im folgenden Bild 12 veranschaulicht werden:

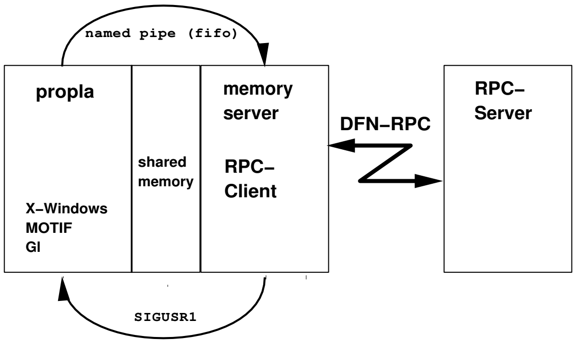

Abbildung 12: `propla` schickt seine Kommandos an `memsrv` über ein FIFO. Da `memsrv` asynchron fertig wird, schickt `memsrv` das Unix-Signal `SIGUSR1` an `propla`, um die Beendigung des Auftrags anzuzeigen.

Durch diese Konstruktion erhält man auf der lokalen Seite ein hohes Maß an Flexibilität, die auf verschiedene Weise genutzt werden kann. So wird zum einen ein Batch-Lauf möglich, ohne daß lokal eine grafische Oberfläche laufen muß, zum anderen kann im Hintergrund ein Archiv-Server mitlaufen, der die anfallenden Daten auf ein Backup-Medium schreibt, denkbar ist auch ein Film-Server, der eine Kamera steuert und für jedes neue Bild belichtet.

4.7 Erfahrungen und Ausblick

Die Methode *Remote Procedure Call* ist für die meisten Anwendungs-Programmierer im technisch-wissenchaftlichen Bereich ein neues Werkzeug, das nach unseren bisherigen Erfahrungen in seiner Realisierung als DFN-RPC durchaus weitere Beachtung verdient. Obwohl wir bislang den Stub-Generator für die Verteilung unserer Anwendung noch nicht nutzen konnten, war der DFN-RPC vergleichsweise einfach einzusetzen. Die Integration in gemischt-sprachige Applikationen geht problemlos. Die Schnittstelle zwischen den beteiligten Rechnern wird vom Benutzer bestimmt und nicht durch ein Software-Paket. Beim Einsatz von X-Windows, GL oder PHIGS hat man keinen direkten Einfluß auf die Datenmengen, die über das Netzwerk geschickt werden. Die Methoden zur Datenkonversion sind für etliche Rechnerkombinationen realisiert. So wird gängigerweise bei uns auf der Cray-2 (RUS) und auf der Convex C220 der Zentralen Datenverarbeitung der Uni Tübingen (ZDV) gerechnet und deren Zahldarstellungen transparent in die von den meisten Workstations benützte IEEE-Darstellung umgewandelt, so daß Datenfiles lokal ohne Netzwerk-Transfer-Programme gleich in platzsparendem binären Format abgelegt werden können.

Insgesamt bietet sich dem Anwendungs-Programmierer mit dem DFN-RPC eine gute Möglichkeit, Resourcen verschiedenster Rechner nahezu transparent zu nutzen und damit dem wissenschaftlichen Rechnen eine neue Qualität zu verleihen.

Literatur

[DCE 90] Open Software Foundation (OSF): Remote Procedure Call in a Distributed Computing Environment. White Paper, OSF, Oct. 1990.

[ERT 91] T. Ertl, et al.: Visualisierung in der Astrophysik. Lehr- und Forschungsbereich Theoretische Astrophysik der Universität Tübingen. Beitrag zum Workshop Visualisierungstechniken, Rechenzentrum der

Universität Stuttgart und Gesellschaft für Informatik, FG 4.1.2. Kurzfassungen der Beiträge. Stuttgart. Juni 1991.

[NEL 81] Bruce-Jay Nelson: Remote Procedure Call. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsilvania. 3. Mai 1981.

[RAB 91] Rolf Rabenseifner: Distributed Applications between Workstation and Supercomputer using ISO/OSI Protocols. Proceedings, Twenty-Seventh Semi-Annual Cray User Group Meeting, S. 80-84. London. 1991.

[SUN 90] SUN microsystems: Network Programming Guide. Part Number 800-3850-00, 1990.

[TAN 89] A.S. Tanenbaum: Computer Networks. Prentice-Hall International, 2.Ed. 1989.