DISKRETE OPTIMIERUNG MIT EVOLUTIONSSTRATEGIEN AUF PARALLELEN, VERTEILTEN SYSTEMEN

Diplomarbeit
am Institut für Baustatik
der Universität Stuttgart

vorgelegt von:
cand.-Ing.
Martin Bernreuther

Betreuer:
Prof. Dr.-Ing. S. Holzer
Dipl.-Ing. Kurt Maute

Erklärung

Hiermit versichere ich, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Stuttgart, den 5. August 1996

(Martin Bernreuther)

Vorwort

Die vorliegende Diplomarbeit ist die letzte Teilprüfung meines Hauptdiploms im Studiengang Bauingenieurwesen an der Universität Stuttgart.

Ich bedanke mich bei meinen Betreuern Herrn Prof. Dr.-Ing. S. Holzer und Herrn Dipl.-Ing. Kurt Maute, die jederzeit für Fragen zur Verfügung standen, sowie bei allen Mitarbeitern des Instituts für Baustatik. Für die Durchsicht des Manuskripts und die zahlreichen Anregungen danke ich ebenso Daniela Schmid und Bernd Schweibenz.

Besonderer Dank gilt meinen Eltern, die mir dieses Studium durch ihre Unterstützung möglich machten.

Stuttgart, im August 1996 Martin Bernreuther

Zusammenfassung

Die Strukturoptimierung erfordert umfangreiche Berechnungen. Mit Hilfe paralleler Programmierung können Leistungssteigerungen erzielt werden. Auf eine allgemeine Übersicht über die Strukturoptimierung folgt ein Abriß über die Grundlagen paralleler Systeme. Ein Schwerpunkt wird auf die Parallelisierung der weit verbreiteten Workstationcluster gelegt. Auf Rechnernetze, sowie Übertragungsprotokolle wird ebenfalls eingegangen, da sie die Grundlage für ein verteiltes System darstellen und sich ihre Eigenschaften auf den Betrieb auswirken.

Die parallele Programmierung baut auf PVM (Parallel Virtual Machine), einer frei verfügbaren Programmbibliothek auf. PVM ermöglicht es, mehrere vernetzte Rechner als einen Parallelrechner zu verwenden. Sie stellt insbesondere Kommunikationsroutinen zwischen einzelnen Prozessen zur Verfügung. Neben der Beschreibung, dem Aufbau und der Funktionsweise dieses Programmpakets wird auf die Programmierung ein Schwerpunkt gelegt. Beispiele verdeutlichen die besprochenen Bibliotheksfunktionen.

Die Lösung großer Probleme im mathematischen Bereich der linearen Algebra, die in statischen und dynamischen Problemen vorkommen, benötigt auf parallelen Systemen neue Ansätze. Die Programmbibliothek ScaLAPACK bietet Funktionen zur Lösung solcher Probleme. Eine kurze Einführung in ScaLAPACK bietet Einsicht in dieses Gebiet.

Für die Parameter der Strukturoptimierung sind oft nur bestimmte, diskrete Werte zulässig. Die Wahl von Stabquerschnitten ist in der Praxis vor allem im Stahlbau auf bestimmte Standardprofile beschränkt. Dies wirkt sich auch auf den Optimierungsalgorithmus aus. Gesucht wird ein Algorithmus, der sowohl diesen Ansprüchen gerecht wird, als auch Möglichkeiten zur Parallelisierung bietet. Vorgestellt werden Evolutionsstrategien, die für dieses Aufgabenfeld robust, effizient und in hohem Maße parallelisierbar sind.

In einem selbst programmierten parallelen Fachwerkoptimierungsprogramm werden all diese Komponenten eingesetzt. Anhand eines Beispiels werden die Grundzüge der Verfahren demonstriert. Hierfür wird ein 10-stäbiges Fachwerk gewählt, für das die maximalen Normalspannungen und Knotenverschiebungen vorgegeben sind und für dessen Stäbe aus einer Liste von 42 Querschnitten jeweils ein Querschnitt zugeordnet wird. Unter den 42^{10} ($\approx 1{,}71 \cdot 10^{16}$) Möglichkeiten der Querschnittszuordnungen ist die Struktur zu ermitteln, die unter Einhaltung der maximalen Spannungen und Verschiebungen ein möglichst geringes Gewicht aufweist.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACU	Array Control Unit
ARP	Address Resolution Protocol
ARPA	Advances Research Projects Agency
ATM	Asynchronous Transfer Mode
AVL	Adelson-Velskij-Landis
BLACS	Basic Linear Algebra Communication Subprograms
BLAS	Basic Linear Algebra Subprograms
CD	Collision Detection
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
DNS	DesoxyriboNukleinSäure
EA	Ende-Anfang
EP	Evolutionary Programming
ES	Evolutionsstrategie
FDDI	Fiber Distributed Data Interface
FIFO	First In, First Out
FTP	File Transfer Protocol
GA	Genetic Algorithms
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Standardization Organisation
LAN	Local Area Network
LAPACK	Linear Algebra PACKage
LLC	Logical Link Control
Lx	Layer x
MAC	Media Access Control
Mbps	Megabit/sec
MIMD	Multiple Instruction, Multiple Data
MPMD	Multiple Program, Multiple Data
MPP	Massively Parallel Processing
NFS	Network File System
OSI	Open Systems Interconnection Reference Model
PAR	Positive Acknowledgement with Retransmit
PBLAS	Parallel Basic Linear Algebra Subprograms
PC	Personal Computer
PDU	Protocol Data Units

PE	Processing Element
PID	Process IDentifier
PVM	Parallel Virtual Machine
RM	Resource Manager
SCALAPACK	SCAlable Linear Algebra PACKage
SI	Système International d'Unités
SISD	Single Instruction, Multiple Data
SMTP	Simple Mail Transfer Protocol
SPMD	Single Program, Multiple Data
TCP	Transmission Control Protocol
TID	Task IDentifier
UDP	User Datagram Protocol
uid	(UNIX) User-IDentifier
XDR	eXternal Data Representation standard

1 Einleitung

1.1 Strukturoptimierung

„Trial and Error“ nennt man im englischen Sprachraum das Suchen einer Lösung durch Versuche. Bei unbefriedigenden Ergebnissen wird versucht durch Veränderung der Ausgangsbedingungen eine Verbesserung zu erzielen. Dies führt zu einem iterativen Vorgehen. Dieselbe Vorgehensweisen findet man im Ingenieurwesen. Die klassischen, experimentellen Versuche werden dabei mehr und mehr durch numerische Berechnungen ersetzt. Bei den Berechnungen hängen die Ergebnisse von den zugrundegelegten Eingangsdaten ab. Beispielsweise werden beim Bemessen von Querschnitten diese üblicherweise zuerst vom Ingenieur gewählt, um in einer statischen Berechnung die Tauglichkeit nachzuweisen. Geleitet von diesen Nebenbedingungen wird versucht, die Querschnitte solange zu verändern, bis ein Ziel, wie z.B. ein möglichst geringer Materialverbrauch, erreicht ist (siehe Abbildung 1).
Für einfache Zusammenhänge kann dieser iterative Prozeß durch eine Rückrechnung ersetzt

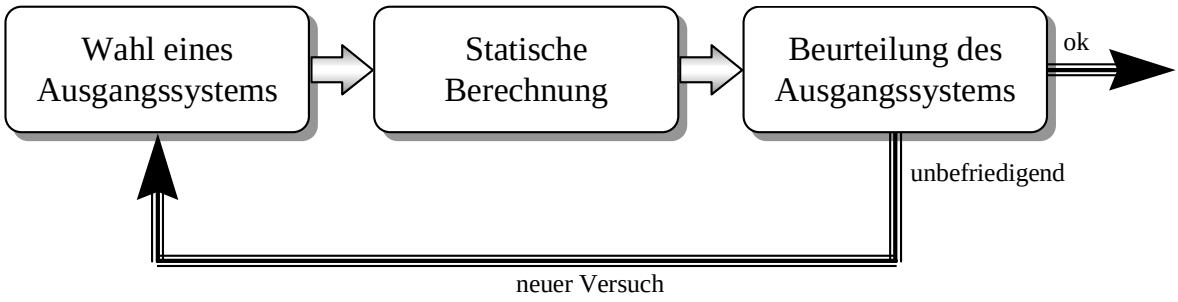


Abbildung 1: Einfaches Bemessungsschema

werden. Die Gesetzmäßigkeiten formt man hierbei so um, daß Eingangswerte und Ergebnisse vertauscht sind. Anhand der Ergebnisse bestimmt man die Eingangswerte direkt. Dies ist im allgemeinen nicht immer möglich.

Bei der *Strukturoptimierung* soll der eingangs erwähnte, vom Ingenieur durchgeführte, iterative Prozeß durch einen mathematischen Algorithmus ersetzt werden (siehe Bletzinger [1]). Die Optimierung, d. h. die Suche nach dem Besten, wird vom Programm selbst durchgeführt und ordnet sich nach Abbildung 2 ein. Der Optimierungsalgorithmus steuert das Ausgangssystem mit Hilfe der Ergebnisse der Analyse.

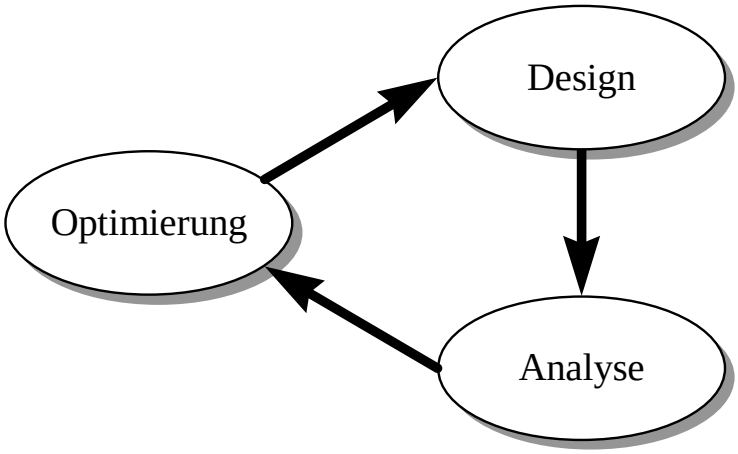


Abbildung 2: Optimiermodell

Bei der *mathematischen Optimierung* müssen *Variablen* oder *Parameter* definiert werden, die veränderbare Eigenschaften des Designs beschreiben. Beispiele für diese Entwurfsvariablen in der Strukturoptimierung sind Querschnittswerte, Koordinaten oder Materialwerte. Dementsprechend spricht man von Querschnitts-, Form- oder Materialoptimierung. Bei der Formoptimierung ist auch eine gleichzeitige Optimierung von Querschnitten und Koordinaten möglich. Die komplexe Topologie- oder Konfigurationsoptimierung verändert zusätzlich die Anordnung von Tragwerkselementen. Diese Variablen spannen einen *Entwurfsraum* $\Re^n$ auf. In der Praxis ist oft eine obere und untere Schranke für die Variablen gegeben. Zum einen müssen die Parameter physikalisch und mechanisch sinnvoll sein, zum anderen müssen sie vorgegebene Grenzwerte einhalten. Ein Beispiel hierfür ist die Optimierung der Breite eines Balkenquerschnitts: die Breite ist nur für positive Zahlenwerte sinnvoll und darf aus produktionstechnische Gründen nicht zu klein oder auch zu groß sein. Solche *Restriktionen* schränken den Entwurfsraum ein. Die *diskrete Strukturoptimierung* läßt für die Entwurfsparameter nur bestimmte Zahlenwerte zu. Ein Beispiel hierfür findet sich im Stahlbau: Die Verwendung von Standardprofilen bedingt bei einer Querschnittsoptimierung eine Auswahl der Querschnitte aus den vorhandenen Profilquerschnitten.

Die Suche nach dem Optimum ist an ein Kriterium gebunden, mit dem unterschiedliche Entwürfe miteinander verglichen werden können. Dieses Kriterium wird durch die *Zielfunktion* f(x) beschrieben. Für zwei zu optimierende Variablen kann man sich diese Funktion als Gebirge vorstellen (siehe z.B. Abbildung 28, Seite 69). Das Optimum kann entweder ein Maximum (Berggipfel) oder ein Minimum (Talsenke) sein. Ein Optimum ist dann gegeben, wenn in unmittelbarer Nähe kein Punkt einen besseren Funktionswert besitzt. Es kann somit

mehrere Optima geben: lokale Optima und ein globales Optimum, das im zulässigen Gebiet den besten Funktionswert aufweist. Der Aufwand dieses globale Optimum zu finden, steigt mit der geforderten Sicherheit an. Bei der diskreten Strukturoptimierung kann die absolute Sicherheit ein globales Optimum zu finden durch eine *totale Enumeration*, dem Durchtesten aller Möglichkeiten, erreicht werden. Dies ist bei komplexeren Aufgabenstellungen aufgrund des immensen Aufwands nicht realisierbar. Bei der *Mehrkriterienoptimierung* sind mehrere Zielfunktionen gegeben, die unterschiedlich gewichtet sein können. Eine einfache Möglichkeit diese Zielfunktionen in eine Ersatzzielfunktion umzuwandeln, ist die Addition der mit Wichtungsfaktoren versehenen Einzelfunktionen. Übliche Zielfunktionen im Bereich der Strukturoptimierung sind Gewicht, gleichmäßige Materialausnutzung, Traglast, Eigenfrequenzen oder Kosten.

Sind zusätzliche *Nebenbedingungen* gegeben, die erfüllt werden müssen, liegt ein *beschränktes Problem* vor. Für *unbeschränkte Probleme* existieren keine Nebenbedingungen. Nebenbedingungen teilen den Entwurfsraum in zulässige und unzulässige Bereiche, je nachdem ob die Nebenbedingungen verletzt sind. Die Nebenbedingungen sind entweder durch *Gleichheitsbedingungen* h(x) oder *Ungleichheitsbedingungen* g(x) gegeben. Gleichheitsbedingungen reduzieren die Dimension des Entwurfsraums. Ungleichheitsbedingungen schränken den Entwurfsraum ein. Sie müssen nur berücksichtigt werden, falls sie verletzt werden. Bei Berücksichtigung sind sie *aktiv*, ansonsten *nicht-aktiv*. Die Menge der aktiven Nebenbedingungen wird als *aktiver Satz* bezeichnet. Ist der aktive Satz der Optimallösung bekannt, können die Ungleichungen in Gleichungen umgewandelt werden.

Die mathematische Optimierung kann durch die folgende allgemeine Form beschrieben werden:

$$\hat{f}(\vec{x}) = Min\left(f(\vec{x})\right) \tag{1}$$

unter Berücksichtigung von:

$$h_j(\vec{x}) = 0\,; \quad j = 1,\ldots,m_e \tag{2}$$

$$g_j(\vec{x}) \le 0\,; \quad j = m_e + 1,\ldots,m \tag{3}$$

$$\vec{x} \in S\left\{x_{Li} \le x_i \le x_{Ui}\,; i = 1,\ldots,n\right\} \subset \Re^n \tag{4}$$

Es werden meist Minimum-Probleme untersucht. Ist das Optimum in Form eines Maximums gesucht, kann dieses Problem leicht auf ein Minimum-Problem transponiert werden:

$$Max\left(f(\vec{x})\right) = -Min\left(-f(\vec{x})\right) \tag{5}$$

In Kapitel 6 wird auf die Evolutionsstrategien eingegangen, die in die Gruppe der nicht-deterministischen Optimierungsverfahren einzuordnen sind. Bei *nicht-deterministischen Optimierungsverfahren* wird anstelle einer komplizierten (deterministischen) Vorschrift, wie z.B. der Berechnung lokaler Gradienten, vom Zufall Gebrauch gemacht. Im Gegensatz zur Monte-Carlo Methode, bei der nur blind gesucht wird, baut sich die Evolutionsstrategie mit Hilfe von Informationen aus vorhergehenden Versuchen indirekt eine Art „Gedächtnis“ auf, welches die Metrik der deterministischen Optimierungsverfahren wiederspiegelt.

1.2 Parallele Systeme

Die Strukturoptimierung erfordert umfangreiche numerische Berechnungen. Dies gilt sowohl für die durchzuführenden statischen oder dynamischen Analysen, als auch für die eigentlichen Optimierungsalgorithmen. Komplexer werdende Aufgaben erfordern immer leistungsfähigere Computer.
Der Weg entscheidende Leistungssprünge zu erzielen, führt zur Parallelität. Hierbei führen mehrere Rechnereinheiten zusammen Berechnungen durch und teilen sich somit die Arbeit. Dieses Konzept wird nicht nur im Bereich der Höchstleistungsrechner verfolgt deren Computerkomponenten an ihre physikalischen Leistungsgrenzen stoßen, sondern auch im Workstation-Bereich, wo der durch die Parallelisierung erzielte Rechnerverbund eine kostengünstige Alternative zum Höchstleistungsrechner darstellen kann.
Selbst bei modernen PC-Betriebssystemen wird momentan ein Schwerpunkt auf die Unterstützung von Netzwerkfunktionalitäten gelegt. Dies ist eine Reaktion auf die wachsende Popularität, die das weltumspannende Internet gerade erfährt, sowie der steigenden Verbreitung von lokalen Netzwerken (Local Area Networks). Der eigenständige PC kann damit auf gemeinsame Ressourcen, wie Drucker, Daten oder Sicherungssysteme zugreifen. Faktisch gleicht ein Hochleistungs-PC heute einer Workstation. PCs sind im Bereich des Bauingenieurwesens in der Praxis sehr weit verbreitet und bilden somit die Basis eines breiten Anwendungsfelds.
In einem Rechnerverbund vernetzter Rechner wird die Rechenkapazität der einzelnen Computer die meiste Zeit nicht voll ausgeschöpft. Somit geht Rechenleistung ungenutzt verloren. Auch können große Berechnungen, die für einen einzelnen Rechner zu aufwendig sind, auf mehrere Rechner verteilt noch lösbar sein! Ein paralleler Ansatz kann damit zur Effizienzsteigerung als auch der Lösbarkeit komplexerer Probleme dienen.

Parallele Berechnungen sind aber nur sinnvoll, wenn auf parallelen Algorithmen aufgebaut wird. Deshalb müssen neue Wege und Verfahren entwickelt werden. Anhand der in Kapitel 6 vorgestellten Evolutionsstrategien wird im besonderen auf Gesichtspunkte der parallelen Umsetzung des Algorithmusses eingegangen. Das Erstellen von parallelen Programmen ist sehr viel aufwendiger und komplizierter und somit auch fehleranfälliger als das Erstellen sequentieller Programme. Die automatische Parallelisierung durch den Compiler ist heute noch nicht in zufriedenstellender Qualität möglich und es ist fraglich, ob es entscheidende Verbesserungen geben wird. Das Erkennen der Programmlogik und der Datenabhängigkeiten sind die entscheidenden Faktoren. Eine effiziente Umsetzung ist im Ingenieurbereich nur durch direkte parallele Programmierung erreichbar. Es ist entscheidend, das Programmierkonzept auf diesen Umstand abzustimmen. Hierfür sind umfangreiche Kenntnisse im Bereich der parallelen Programmierung nötig.

2 Paralleles und verteiltes Rechnen

2.1 Grundlagen

2.1.1 Klassifikation

Nach einer Klassifikation von Flynn [4] werden Rechner grob in vier Gruppen eingeteilt (siehe Tabelle 1). Während die einfachen von-Neumann-Rechner in die Klasse der SISD-Computer fallen, kann mit mehreren vernetzten Rechnern dieser Art ein verteiltes System der MIMD-Klasse aufgebaut werden. Für parallele Systeme sind vor allem die SIMD- und MIMD-Rechner interessant.

SISD Single Instruction Single Data	SIMD Single Instruction Multiple Data
Einprozessorrechner (von-Neumann-Rechner)	Vektor- , Arrayrechner
MISD Multiple Instruction Single Data	MIMD Multiple Instruction Multiple Data
Pipelinerechner	Mehrprozessorenrechner Verteilte Systeme (z.B. Workstationcluster)

Tabelle 1: Klassifikation nach Flynn[4]

2.1.1.1 Single Instruction Multiple Data - Konzept

Das SIMD-Konzept baut auf einer *synchronen Parallelität* auf: Auf allen Prozessorelementen (PE) werden dieselben Instruktionen zeitgleich ausgeführt. Es existiert nur ein Kontrollfluß, der auf einem zentralen Steuerrechner (ACU: Array Control Unit) abläuft. Dieser steuert die einzelnen PEs, die dieselben Instruktionen auf ihren lokalen Daten ausführen oder inaktiv bleiben. Vor allem bei der Vektorarithmetik wird dies vorteilhaft eingesetzt. Im Gegensatz zu den ungekoppelten PEs eines Vektorrechners sind beim Arrayrechner die einzelnen PEs über ein Verbindungsnetzwerk gekoppelt. SIMD-Rechner können als massiv parallele Systeme ausgeführt werden. Die Connection Machine[*] arbeitet z.B. mit 65536 relativ leistungsschwachen 1-Bit Prozessoren und die MasPar[†] mit 16384 4-Bit Prozessoren.

[*] Hersteller: Thinking Machines, Bedford, Massachusetts,USA (http://www.think.com/)
[†] Hersteller: Digital Equipment Corporation (DEC), Maynard, Massachusetts,USA (http://www.digital.com/)

2.1.1.2 Multiple Instruction Multiple Data - Konzept

Im Gegensatz zu SIMD-Rechnern werden MIMD-Rechner mit *asynchroner Parallelität* betrieben. Hierbei wird jeder Rechner über einen eigenständigen Kontrollfluß gesteuert. Sollen zwischen den einzelnen PEs Daten ausgetauscht werden, so ist bei der asynchronen Parallelität eine Synchronisation notwendig. MIMD-Systeme können über einen gemeinsamen Speicher verfügen. Diese enge Koppelung der einzelnen PEs findet man oft bei Mehrprozessorrechnern. Erfolgt die Koppelung nur lose über ein Verbindungsnetzwerk mit Nachrichtenaustausch, so liegt ein verteiltes System vor. Für MIMD-Rechner werden leistungsfähige Prozessoren verwendet. Die Anzahl der PEs ist limitiert, da die Kommunikation und somit die Synchronisationskosten mit wachsender Anzahl der PEs ansteigen.
Es kann bei der MIMD-Klassifikation noch zwischen MPMD (Multiple Program/Multiple Data) und SPMD (Single Program/Multiple Data) unterschieden werden (siehe Geiger [3]). Während beim MPMD-Modell die einzelnen PEs völlig unterschiedliche Programme abarbeiten, gibt es beim SPMD-Konzept nur noch ein Programm. Im Gegensatz zum SIMD-Rechner können sich die PEs beim SPMD-Konzept in unterschiedlichen Zweigen des Programms befinden und somit unterschiedliche Instruktionen ausführen. Das SPMD stellt ein Bindeglied vom MIMD- zum SIMD-Konzept dar.

2.1.2 Parallelitätsebenen

Eine weitere Einteilung kann in Parallelitätsebenen, den Abstraktionsgraden der Parallelität, erfolgen. Die Ebenen unterscheiden sich in der *Granularität*, mit der die Größe der zu parallelisierenden Blöcke klassifiziert wird. Ein solcher Block kann eine einzelne Prozessoranweisung (Bitebene) oder ein gesamtes Programm (Programmebene) sein. Die Ebenen beschreiben somit den Abstraktionsgrad der Parallelisierung. Je tiefer die Ebene, desto feinkörniger und je höher, desto grobkörniger die Parallelität (sieheTabelle 2).

Granularität	Ebene	Verarbeitungseinheit	Beispiel
grobkörniger	Programmebene	Job, Task	Multitasking-Betriebssystem
↑	Prozedurebene		MIMD-System
↓	Ausdruckebene		SIMD-System
feinkörniger	Bitebene		Arithmetical Logical Unit

Tabelle 2: Parallelitätsebenen

2.1.2.1 Programmebene

Die Programmebene stellt die grobkörnigste Parallelitätsebene dar. Es werden komplette unabhängige Programme gleichzeitig abgearbeitet. Die Verwaltung kann vollständig von einem Multitasking-Betriebssystem übernommen werden.
Falls die Anzahl der auszuführenden Programme die Anzahl der PEs übersteigt, kann eine quasi-parallele Ausführung simuliert werden, indem den Programmen Zeitscheiben eines

PEs zugeordnet werden. Dies wird vom Scheduler unter Berücksichtigung von Prioritäten verwaltet.

2.1.2.2 Prozedurebene

Laufen mehrere Programmteile parallel ab, so liegt eine Parallelität in der Prozedurebene vor. Die parallel ablaufenden Abschnitte des Programms nennt man Prozesse und können als Unterprogramme verstanden werden. Um einen leistungsmindernden Datenaustausch zwischen den Prozessen zu minimieren, sollten sie möglichst unabhängig voneinander sein. MIMD-Systeme bauen auf dieser Parallelitätsebene auf.
Werden einem PE mehrere Prozesse zugeordnet, müssen die Prozesse quasi-parallel ausgeführt werden. Analog dem Modell auf Programmebene in Kapitel 2.1.2.1 wird dies von einem Scheduler mit Zeitscheiben bewerkstelligt. In Abbildung 3 werden die drei Zustände „Bereit“, „Rechnend“ und „Blockiert“, sowie deren Abhängigkeiten dargestellt.

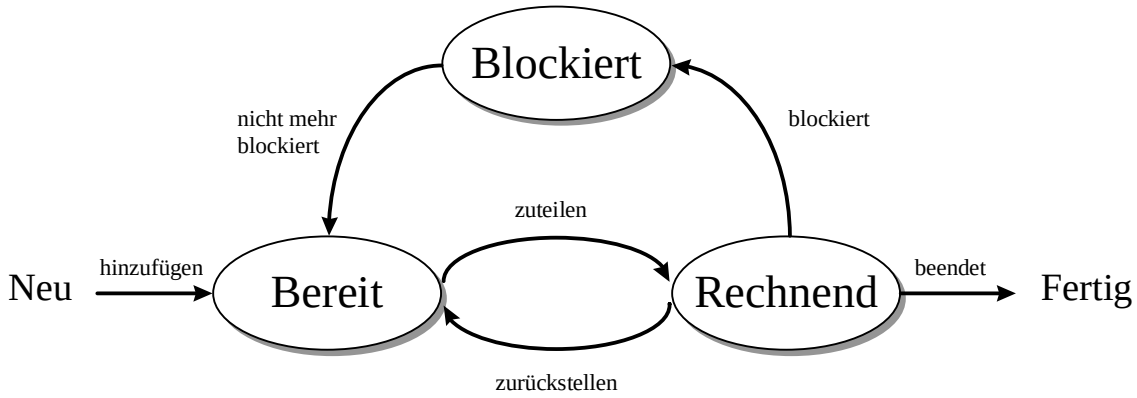


Abbildung 3: Prozeßzustände

Neue Prozesse werden in die „Bereit“-Warteschlange eingereit. Nacheinander wird jedem Prozeß der „Bereit“-Warteschlange eine „Rechnend“-Zeitscheibe zugeteilt. Falls der Prozeß in dieser Zeit nicht terminiert, wird er wieder in die Warteschlange zurückgestellt. Bei Prozessen, die auf Ereignisse warten und somit blockiert sind, wäre es unnötig eine „Rechnend“-Zeitscheibe zuzuteilen und so Rechenzeit zu verschwenden. Solche Prozesse werden in den „Blockiert“-Zustand abgeschoben und nach Eintritt des erwarteten Ereignisses wieder in die „Bereit“-Warteschlange eingereiht.

2.1.2.3 Ausdruckebene

In arithmetischen Ausdrücken können voneinander unabhängige Teile gleichzeitig abgearbeitet werden. Hier kann noch zwischen der Anweisungsebene und der Instruktionsebene differenziert werden.
Ein einfaches Beispiel für eine Parallelität auf Anweisungsebene ist eine Vektoraddition, die von einem einfachen SISD-Rechner in einer Schleife sequentiell durchgeführt werden muß und von einem Rechner mit einer Anzahl PEs > Anzahl Vektorkomponenten in einem Schritt berechnet werden kann. Die *Daten-Parallelität* zeigt die Feinkörnigkeit dieser Par-

allelitätsebene auf. Jedem Prozessor wird hier ein Datenelement zugeordnet. Bei einem SISD-Rechner stehen hingegen einer aktiver CPU eine Mehrzahl passiver Daten gegenüber, was als *von-Neumann-Flaschenhals* bezeichnet wird. Für eine Parallelisierung auf dieser Ebene ist es wichtig, folgende Abhängigkeiten zu vermeiden:

- *echte* oder *scheinbare Datenabhängigkeiten*: Während bei echten Datenabhängigkeiten eine Anweisung die Ergebnisse der vorhergehenden benötigt, entstehen scheinbare Datenabhängigkeiten durch ungeschickten Programmierstil.
- *prozedurale Abhängigkeiten* aufgrund von z.B. Programmverzweigungen
- *operationale Abhängigkeiten*, wenn Betriebsmittel von verschiedenen Anweisungen blockiert werden.

Für voneinander unabhängige Anweisungen spielt die Reihenfolge der Bearbeitung keine Rolle!

Die Instruktionsebene bezieht sich auf Basisoperatoren eines Ausdrucks. Als Beispiel soll der Vergleich einer parallelen und sequentiellen Addition mehrerer Komponenten in Tabelle 3 dienen. Diese Parallelisierung kann von einem Compiler, der mit Hilfe eines Parsers den Ausdruck aufspalten muß, selbständig erkannt werden.

parallel:								**sequentiell:**							
($\log_2$n Schritte)	1	+	2	+	3	+	4	(n-1 Schritte)	1	+	2	+	3	+	4
1.Schritt:		3		+		7		1.Schritt:		3		+	3		
2.Schritt:				10				2.Schritt:			6			+	4
								3.Schritt:				10			

Tabelle 3: Parallele/sequentielle Addition

2.1.2.4 Bitebene

Eine parallele Ausführung von Bit-Operationen wird heute von jedem gängigen Mikroprozessor durchgeführt.

2.1.3 Anwendungen verteilter Systeme

Nach Rothermel [9] können Anwendungen verteilter Systeme in folgende Klassen eingeteilt werden:

- *Hochleistungsanwendungen*:
  Eine Berechnung wird in kleinere Teilberechnungen aufgeteilt. Diese Teilberechnungen können ohne größeren Kommunikationsaufwand von verschiedenen Computern ausgeführt werden. Als Voraussetzung für die Parallelisierbarkeit muß die Reihenfolge der Abarbeitung der Teilberechnungen beliebig und damit unabhängig voneinander sein. Unter „parallele Anwendung“ soll im folgenden bezüglich verteilter Systeme eine Anwendung dieser Klasse verstanden werden.

- *Anwendungen mit funktionaler Spezialisierung*:
  Jedem einzelnen Rechner wird eine Aufgabe zugeteilt, für die er sich besonders eignet. Beim Client-Server Konzept wird vom Server ein Dienst angeboten und vom Client genutzt. Typische Serverdienste sind Drucken, Filezugriffe oder Datenbankabfragen. Bei einem Finite Elemente Programm könnte z.B. die Berechnung auf einem Superskalarrechner erfolgen, während anschließend die grafische Aufbereitung und Ausgabe auf einer Grafikworkstation erfolgt. Die Computer nutzen hier jeweils Ressourcen des anderen Rechners.
- *Fehlertolerante Anwendungen*:
  Der Ausfall eines Systems wird von einem anderen aufgefangen. Dazu muß die Funktionalität redundant vorhanden sein. Mit Hilfe der verteilten Systeme kann hier die Zuverlässigkeit gesteigert werden.
- *Inhärent verteilte Anwendungen*:
  Dienste wie das Electronic Mail sind auf einem Rechner betrieben sinnlos. Ein verteiltes System ist hier eine notwendige Bedingung.

2.1.4 Beurteilungskriterien für parallele Anwendungen

Die Leistungsfähigkeit einer parallelen Anwendung mit n PEs kann unter verschiedenen Gesichtspunkten beurteilt werden:
Steht die Frage im Vordergrund, um wieviel schneller die Anwendung im Vergleich zur rein sequentiellen Abarbeitung ist, kann dies durch den*Speedup* ausgedrückt werden:

$$Sp^{(n)} = \frac{T_{ges}^{(1)}}{T_{ges}^{(n)}} \leq n \tag{6}$$

Die Gesamtzeit T_{ges} setzt sich bei der parallelen Anwendung aus der CPU-Zeit T_{cpu} und der Zeit für Kommunikation T_{com} und Synchronisation T_{syn} zusammen, während sie im sequentiellen Fall der CPU-Zeit entspricht:

$$T_{ges}^{(n)} = T_{cpu}^{(n)} + T_{syn}^{(n)} + T_{com}^{(n)} \tag{7}$$

Eine n-fache Geschwindigkeit kann also nicht erreicht werden, weil Kommunikation und Synchronisation Zeit kosten. Zudem sind auch in parallelen Anwendungen sequentielle Programmteile enthalten.
Wird der Speedup auf die Belastung eines PEs umgerechnet, spricht man von der*Effizienz*:

$$E^{(n)} = \frac{Sp^{(n)}}{n} \leq 1 \tag{8}$$

Um die Auswirkungen der sequentiellen Teile einer parallelen Anwendung zu bestimmen, wird die CPU-Zeit der Anwendung in einen sequentiellen Teil T_{seq} und einen parallelisierbaren Teil T_{par} aufgeteilt, wobei p den prozentualen Anteil der Zeit T_{seq} an der gesamten CPU-Zeit T_{cpu} darstellt:

$$T_{cpu} = T_{seq} + T_{par} \tag{9}$$

$$p = \frac{T_{seq}}{T_{cpu}} \tag{10}$$

Damit ergibt sich für den Speedup:

$$Sp^{(n)} = \frac{T_{cpu}}{T_{seq} + \frac{T_{par}}{n} + T_{syn} + T_{com}} \tag{11}$$

und damit

$$Sp^{(n)} = \frac{T_{cpu}}{p \cdot T_{cpu} + \frac{(1-p)}{n} \cdot T_{cpu} + T_{syn} + T_{com}} = \frac{1}{p \cdot \left(1 - \frac{1}{n}\right) + \frac{1}{n} + \frac{T_{syn} + T_{com}}{T_{cpu}}} \tag{12}$$

Hieraus folgt die Aussage des *Amdahl'schen Gesetzes* [5], daß auf einem „idealen" Parallelrechner, bei dem keine Zeit für Synchronisation und Kommunikation anfällt, der Speedup mit

$$Sp^{(n)} = \frac{T_{seq} + T_{par}}{T_{seq} + \frac{T_{par}}{n}} = \frac{1}{p \cdot \left(1 - \frac{1}{n}\right) + \frac{1}{n}} \tag{13}$$

definiert und somit mit

$$\lim_{n \to \infty} Sp^{(n)} = \frac{1}{p} \tag{14}$$

begrenzt ist.

Dies ist auch aus Abbildung 4 ersichtlich:

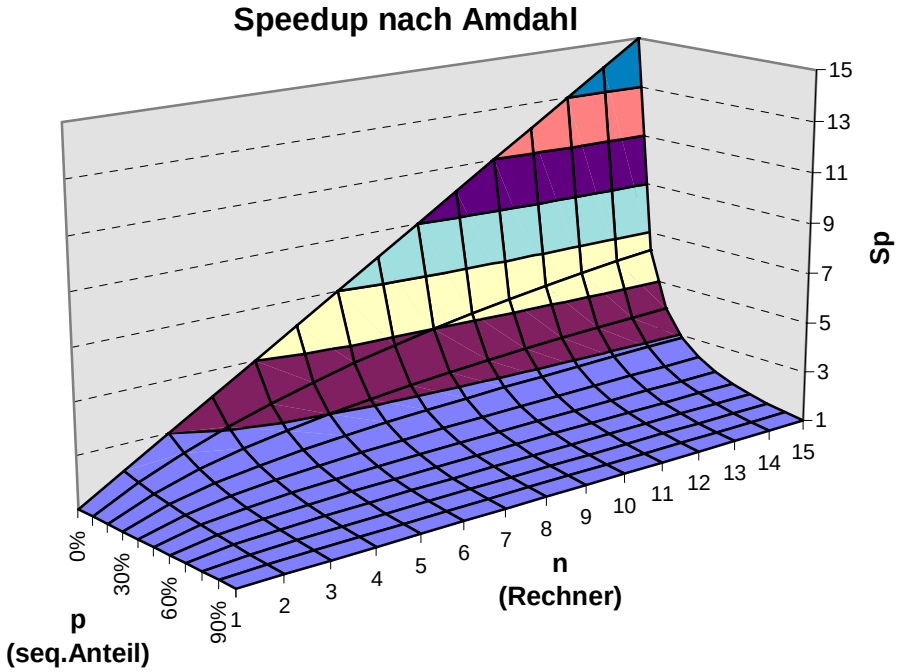


Abbildung 4: Speedup nach Amdahl[5]

Bei parallelen Anwendungen steht aber oft nicht die Zeitverkürzung einer Berechnung im Vordergrund, sondern die Möglichkeit in gleicher Zeit ein größeres Problem berechnen zu können. Dieser Sichtweise trägt der *Scaleup* Rechnung:

$$Sc^{(n)} = \frac{S^{(n)}}{S^{(1)}} \tag{15}$$

$$T_{ges}^{(n)}\left(S^{(n)}\right) = T_{ges}^{(1)}\left(S^{(1)}\right) \tag{16}$$

In diesen Formeln soll S die Problemgröße wiedergeben. Der Scaleup gibt an, um wieviel größer das Problem bei gleicher Berechnungsdauer für die parallele Implementierung in Bezug auf die sequentielle ist. Im Ingenieurwesen ist der Scaleup oft wichtiger als der Speedup. Die sequentiellen Teile eines parallelen Programmes sind normalerweise von der Problemgröße unabhängig, während der parallelisierbare Teil mindestens proportional steigt. Nach Gustafson [6] erhält man so einen linearen Scaleup.

2.2 Verbindungsstrukturen

Um ein Gesamtsystem zu bilden, müssen die einzelnen Knoten verbunden werden. Über diese Verbindungsstruktur werden Nachrichten ausgetauscht oder es wird auf einen gemeinsamen Speicher (shared memory) zugegriffen. Anzustreben ist eine möglichst hohe *Konnektivität*, d. h. eine möglichst geringe Distanz zwischen zwei Prozessoren oder Speichermodulen. Die Distanz bzw. Pfadlänge stellt den kürzesten Weg, d. h. die minimale Anzahl von Verbindungsstücken (links) zwischen zwei Knoten dar. Demzufolge sollte der *Durchmesser* eines Netzwerkes, der maximale Abstand zweier Knoten, möglichst klein sein.
Die Anzahl der Verbindungen je PE wird als *Grad* bezeichnet und ist, wie auch die Netzwerk-Bandbreite, limitiert. Der Aufbau einer Verbindungsstruktur stellt also eine Optimierung der folgenden Größen dar:

- *Übertragungsgeschwindigkeit*:
  Datenmenge, die in einer bestimmten Zeit zwischen zwei Knoten übertragen werden kann.
- *Latency* (Latenzzeit):
  Zeit zwischen dem Abschicken einer Nachricht und der Ankunft beim Empfänger.
- *Skalierbarkeit*:
  Erweiterbarkeit bzw. Grenzen der Knotenanzahl, für die das Konzept sinnvoll ist.
- eine möglichst hohe Zahl von Verbindungen, die gleichzeitig aufbaubar sind.

Beim Verschicken kleiner Nachrichten ist die Latency oft wichtiger als die Übertragungsgeschwindigkeit, die dann bei größeren Datenmengen Auswirkungen zeigt.
Bezogen auf eine grobe Einteilung der Verbindungsstrukturen in die Klassen

- Punkt-zu-Punkt Verbindungen
- Bus-Netzwerke und
- Netzwerke mit Schaltern

wird nur auf die ersten beiden Klassen eingegangen.

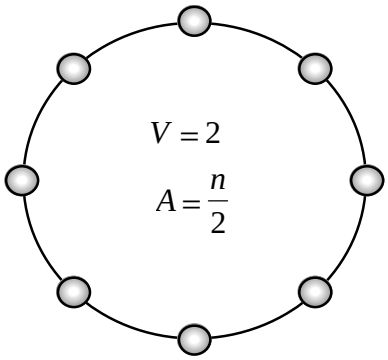
2.2.1 Punkt-zu-Punkt Verbindungen

In den Abbildungen werden folgende Bezeichnungen verwendet:

n	die Anzahl der PEs im Netzwerk
V	die Verbindungsleitungen je PE (Grad)
A	der maximaler Abstand zwischen zwei PEs

Die Eignung einer Netzwerktopologie wird neben diesen Parametern hauptsächlich von dem zu lösenden Problem bestimmt.

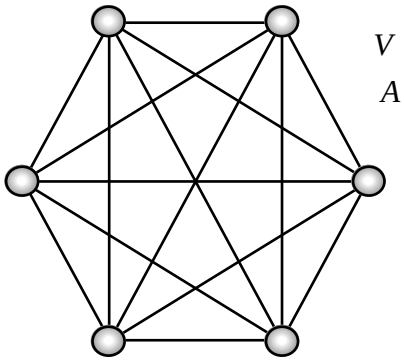
2.2.1.1 Ring



Bei der Ringstruktur werden pro PE nur zwei Verbindungsleitungen benötigt. Der Abstand der am weitest entfernten PEs ist mit $\frac{n}{2}$ allerdings ungünstig. Das schränkt die Erweiterbarkeit, die ansonsten problemlos möglich ist, ein.

Abbildung 5: Ring

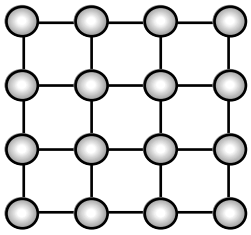
2.2.1.2 Vollständiger Graph



Beim vollständigen Graph ist die Konnektivität optimal. Die Produktionskosten sind aber mit $\frac{n(n-1)}{2}$ sehr hoch, so daß diese Struktur für eine große Anzahl PEs nicht mehr durchführbar ist.

Abbildung 6: Vollständiger Graph

2.2.1.3 Gitter und Torus

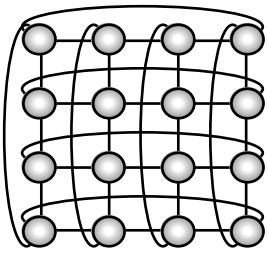


$V = 4$

$A = 2 \cdot \sqrt{n} - 2$

Quadratisches Gitter

(4-facher Nearest-Neighbor)



$V = 4$

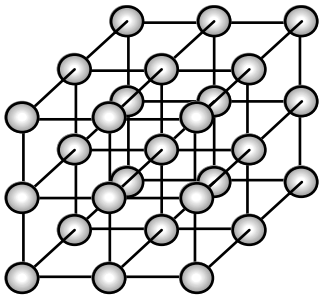
$A = \sqrt{n}$

Quadratischer Torus

(4-facher Nearest-Neighbor)

Abbildung 7: Quadratisches Gitter und Torus

Gitterstrukturen werden häufig eingesetzt. Falls die in Abbildung 7 dargestellte 4-fache Nearest-Neighbor-Gitterstruktur mit den Diagonalverbindungen ergänzt wird, erhält man die 8-fach Nearest-Neighbor-Gitterstruktur mit V = 8 und A = $\sqrt{n}$-1 . Beim Übergang vom 4-fachen zum 8-fachen Nearest-Neighbor-Torus müssen zusätzlich noch die diagonalen Randverbindungen ergänzt werden und man erhält V = 8 und A = $\frac{\sqrt{n}}{2}$. Gitter können auch als hexagonale oder dreidimensionale kubische Gitter ausgeführt werden (siehe Abbildung 8).



$V = 6$

$A = 3 \cdot \sqrt[3]{n} - 3$

Abbildung 8: Kubisches Gitter

2.2.1.4 Sonstige Punkt-zu-Punkt Verbindungsstrukturen

Beim Hypercube ist die Anzahl der PEs vorherbestimmt. Ein Hypercube der Dimension 0 besteht aus einem PE. Aufgrund einer rekursiven Definition besteht ein Hypercube der Dimension i aus zwei Hypercubes der Dimension i-1, wobei entsprechende PEs verbunden werden (siehe Abbildung 9).

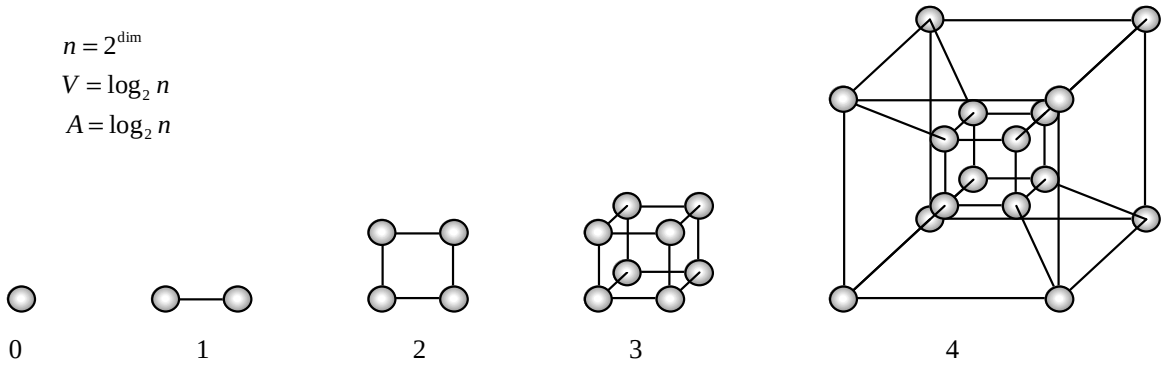


Abbildung 9: Hypercubes der Dimensionen 0 - 4

Eine weitere Klasse stellen Baumstrukturen wie der „Binärbaum“ oder der „Quadtree“ dar.

2.2.2 Busartige Verbindungen

Diese Art der Verbindung kann auch in lokalen Netzen verwendet werden. Neben dem Bussystem wird auch auf den Token-Ring eingegangen.

2.2.2.1 Bussystem

2.2.2.1.1 Funktionsweise

Bei diesem System hängen alle Knoten gleichberechtigt an einem Kommunikationsmedium. Die Kommunikation kann bitseriell oder parallel erfolgen. Beim Senden wird die Nachricht zur eindeutigen Identifikation des Empfängers mit einer Zieladresse versehen. Der Empfang wird quittiert. Alle anderen Knoten sind von diesem Vorgang nicht betroffen. Das Versenden einer Nachricht an mehrere Empfänger zugleich (Broadcast) ist ebenfalls möglich. Es darf sich aber immer nur eine Nachricht auf dem Kommunikationsmedium befinden. Deshalb ist das Empfangen mehrerer Nachrichten zu einem Zeitpunkt nicht möglich.
Der Zugriff kann durch verschiedene Mechanismen geregelt werden:

- *Auswahl*:
  Der sendeberechtigte Knoten wird zentral oder dezentral von einem Algorithmus ausgewählt.
- *Reservierung*:
  Ein sendewilliger Knoten meldet seinen Wunsch an und läßt sich den Zugriff reservieren, wobei der Zeitpunkt des Zugriffs vorherbestimmbar ist.

- *Random-access*:

Jeder Knoten kann zu jedem Zeitpunkt senden. Beim gleichzeitigen Senden zweier oder mehrerer Knoten wird dieser Zugriffskonflikt von der Kollisionsbehandlung gelöst. Der Sender muß dann nach einer zufälligen Zeit ein weiteres Mal senden. Hierbei existiert keine Busvergabesteuerung (Anarchie).

2.2.2.1.2 Wertung

Die Erweiterbarkeit ist optimal, da pro neuem Knoten nur eine zusätzliche Verbindung notwendig ist. Auch erweist sich das Bussystem beim Ausfall eines Knotens fehlertolerant, da nur dieser Knoten betroffen ist. Der Ausfall des Kommunikationsmediums legt jedoch das gesamte System lahm. Das größte Problem des Bussystems ist die Skalierbarkeit, da ab einer bestimmten Anzahl Knoten ein Sättigungszustand erreicht wird. Das Kommunikationsmedium kann nicht mit steigender Anzahl der Knoten mitwachsen.

2.2.2.2 Token-Ring

2.2.2.2.1 Funktionsweise

Die Rechnerknoten werden in Ringform verbunden. In diesem Ring kreist ein Bitmuster mit dem ein Zustand signalisiert wird: das Token. Ein Freitoken zeigt an, daß gerade kein Knoten sendet. Ein sendebereiter Knoten wartet auf ein Freitoken, verwandelt es in ein Besetzttoken und hängt Empfängeradresse und Nachricht an. Beim Empfänger wird das Besetzttoken zum Freitoken. Für den Fall des Sendens an mehrere Empfänger (Broadcast) fertigt sich jeder Empfänger eine Kopie der Nachricht an. Die Nachricht wird erst vom Netz genommen, wenn sie wieder beim Sender eintrifft, der in diesem Fall dann das Besetzt- in ein Freitoken umwandelt. Das Freitoken gibt einem weiteren Knoten die Möglichkeit Daten zu senden.
Eine weitere Form einer Ringstruktur ist der *Slotted Ring*, bei dem Nachrichtencontainer fester Größe (slots) kreisen, die die Statusinformation und die Daten beinhalten. Ein freier Slot kann vom Sender „gefüllt“ und vom Empfänger „geleert“ werden.

2.2.2.2.2 Wertung

Wie beim Bussystem ist die Erweiterbarkeit beim Ring mit einer zusätzlichen Verbindung optimal. Die Beschränkung der Anzahl der maximal möglichen Knoten ist allerdings weniger stark. Der Kommunikationsoverhead ist größer, so daß die Leistungsfähigkeit bei geringer Kommunikation im Vergleich zum Bus niedriger ist.

2.3 Asynchrone Parallelität

Nachdem ein allgemeiner Überblick über Parallelität, Netzwerke und Hardwareeigenschaften gegeben wurde, wird im folgenden näher auf die asynchrone Parallelität und insbesondere auf die Implementierung auf Workstationcluster eingegangen. Die asynchrone Parallelität erfolgt auf MIMD-Rechnern (siehe 2.1.1.2) auf Prozedurebene (siehe 2.1.2.2) mit grobkörniger Granularität.

2.3.1 Probleme

Bei lose gekoppelten verteilten Systemen mit Nachrichtenaustausch steht die Kommunikation im Vordergrund, während bei eng gekoppelten Systemen mit gemeinsamen Speicher der Zugriff von mehreren Prozessen auf einen Speicherbereich eine Problematik darstellt. Für den Nachrichtenaustausch muß, wie auch für den geregelten Zugriff auf einen Speicherbereich, eine Synchronisation erfolgen. Die Synchronisation wirkt sich leistungsmindernd auf das System aus. Auch bei lose gekoppelten Systemen kann beim „Virtual Shared Memory System“ ein gemeinsamer Speicher simuliert werden. Hierfür ist ein zusätzlicher Aufwand an Kommunikation und Verwaltung nötig.

2.3.1.1 Inkonsistente Daten

Eine willkürliche Verzahnung von Zugriffen auf einen Speicherbereich ist im allgemeinen nicht zulässig. Fehler dieser Kategorie sind zeitabhängig, schwer reproduzierbar und somit schwer zu lokalisieren. Es wird in diesem Zusammenhang zwischen den folgenden Problemklassen unterschieden:

- *Verlorener Update*
- *Inkonsistente Analyse*
- *Unbestätigte Abhängigkeit*

Diese Problemklassen werden anhand von Beispielen demonstriert.

2.3.1.1.1 Verlorener Update

Es soll einer Variablen v von Prozeß 1 ein Wert s hinzuaddiert werden und von Prozeß 2 soll die Summe anschließend mit einem Wert m multipliziert werden:

$$\underbrace{\underbrace{(v+s)}_{\text{Prozeß 1} \rightarrow v'} \cdot m}_{\text{Prozeß 2}} \rightarrow v'' \tag{17}$$

Bei einem unkoordinierten Zugriff der beiden Prozesse sind vier Varianten denkbar:

- Beide Prozesse greifen auf den ursprünglichen Wert zu:
  - Prozeß 1 schreibt vor Prozeß 2 das Ergebnis zurück und wird somit von Prozeß 2 überschrieben. Damit ist das Resultat die reine Multiplikationsoperation.
  - Prozeß 2 schreibt vor Prozeß 1 das Ergebnis zurück und wird somit von Prozeß 1 überschrieben. Damit ist das Resultat die reine Additionsoperation.
- Beide Prozesse greifen nacheinander auf die Variable zu:
  - Prozeß 2 wird vor Prozeß 1 ausgeführt. Die Multiplikation wird auf den ursprünglichen Wert ausgeführt.
  - Prozeß 1 wird vor Prozeß 2 ausgeführt. Man erhält die korrekte Lösung!

2.3.1.1.2 Inkonsistente Analyse

Prozeß 1 subtrahiert von einer Variablen a einen Wert c, der zur Variablen b addiert wird. Prozeß 2 berechnet die Summe s der beiden Variablen a und b, die immer konstant sein muß:

$$\begin{aligned} a' &= a - c \quad ; \quad b' = b + c \\ s &= a' + b' = a + b \end{aligned} \tag{18}$$

Falls Prozeß 2 beide Variablen vor oder nach Beendigung von Prozeß 1 liest erhält man das korrekte Ergebnis. Für den Fall, daß für A der ursprüngliche Wert verwendet wird und für B der von Prozeß 1 modifizierte Wert oder umgekehrt ergeben sich fehlerhafte Summen. Dieser Fall spielt bei verteilten Datenbanken (z.B. bei Banküberweisungen) eine wichtige Rolle.

2.3.1.1.3 Unbestätigte Abhängigkeit

Eine Prozedur ändert globale Daten. Falls ein Fehler auftritt soll diese Prozedur die ursprünglichen Daten wiederherstellen (roll back). Eine zweite Prozedur, die auf diese Daten zugreift, verarbeitet falsche Daten, falls sie zwischen Änderung und Wiederherstellung Daten liest.

2.3.1.2 Verklemmungen

Nach Bräunl [2] wird eine *Deadlock*-Verklemmung folgendermaßen definiert:

„Eine Gruppe von Prozessen wartet auf das Eintreten einer Bedingung, die nur durch Prozesse der Gruppe selbst hergestellt werden kann (wechselseitige Abhängigkeit)."

Während beim Deadlock alle Prozesse im „Blockiert"-Zustand sind (siehe 2.1.2.2), handelt es sich beim *Livelock* um aktive Prozesse, die sich in Warteschleifen befinden.

2.3.2 Synchronisation und Zugriffsverwaltung

Um den Zugriff auf einen Speicherbereich oder Betriebsmittel für jeweils nur einen Prozeß freizugeben sind besondere Konstrukte erforderlich. Ein solcher Zugriff erfolgt in einem „*kritischen Abschnitt*".

2.3.2.1 Busy-Wait

Das Busy-Wait-Konzept basiert auf einer Warteschleife, die überprüft, ob der kritische Abschnitt frei ist und erst dann terminiert. Beim Eintritt in diesen Abschnitt wird er über ein Flag für alle anderen Prozesse gesperrt. Ein Nachteil dieses Verfahrens sind die Warteschleifen, die Rechenzeit in Anspruch nehmen.

2.3.2.2 Semaphore

Die Semaphore können wie Signalmasten im Eisenbahnwesen gedeutet werden. Dijkstra führte sie 1965 ein. Die Operationen zum Stellen des Signal werden mit P („Passieren") und V („Verlassen") bezeichnet. Der kritische Abschnitt beginnt bei jedem Prozeß mit einer P- und endet mit einer V-Anweisung. Den Anweisungen wird eine Variable zugeteilt, die für diesen kritischen Abschnitt spezifisch ist und das Bindeglied zwischen den zusammengehörenden kritischen Abschnitten verschiedener Prozesse darstellt. Die P- und V-Anweisung selbst muß atomaren Charakter aufweisen: Während ihrer Ausführung darf im System kein anderer Prozeß „dazwischenfunken".

2.3.2.3 Monitore

Ein Monitor ist ein abstrakter Datentyp, der sowohl Daten enthält, die vor gleichzeitigem Zugriff zu schützen sind als auch Zugriffs- und Synchronisationsmechanismen. Die zu synchronisierenden Prozesse rufen Monitor-„Entries“ mit dazugehörigen „Conditions“ auf. Der Monitor besteht aus den statischen, lokalen Monitor-Daten, die zwischen zwei Aufrufen konstant bleiben und auf die nur über die Entries zugegriffen werden kann. Die Entries entsprechen Monitor-Prozeduren.

2.3.3 Lastbalancierung

Die Lastbalancierung (load balancing) kann große Auswirkungen auf die Effizienz einer MIMD-Anwendung haben. Falls auf einen Prozeß gewartet werden muß, ist für diese Zeit die Leistungsfähigkeit des Mehrprozessorensystems auf ein Einprozessorensystem herabgesetzt. Mit Scheduling-Modellen wird versucht, die Rechenbelastung auf die einzelnen PEs zu verteilen. Hierbei unterscheidet man zwischen statischem und dynamischen Lastausgleich.

2.3.3.1 Statischer Lastausgleich

Bei einer statischen Verteilung der Prozesse auf die PEs findet zur Laufzeit keine Verlagerung von einem zugeteilten Prozeß auf ein anderes PE statt. Die Prozesse werden den PEs vor der Ausführung zugewiesen. Der Verwaltungsaufwand ist hierbei gering.

2.3.3.2 Dynamischer Lastausgleich

Die dynamische Verteilung der Prozesse während der Laufzeit durch Umgruppieren bereits zugeordneter Prozesse (process migration) ermöglicht es, während des Programmlaufes auf Leistungsengpässe einzelner PEs zu reagieren. Die Prozessorlast wird ermittelt und ein Schwellwert (threshold) bestimmt, ob eine Überlastung vorliegt. Für die Steuerung dieser dynamischen Verteilung sind folgende Methoden möglich:

- *Empfänger-Initiative*:
  PEs fordern bei geringer Auslastung Prozesse an.
- *Sender-Initiative*:
  PEs versuchen bei zu hoher Auslastung Prozesse abzugeben.
- *Hybride Methode*:
  Empfänger- und Sender-Initiative ist möglich.

Während die Empfänger-Initiative für hohe Systemlast geeignet ist verwendet man die Sender-Initiative besser bei niedriger Systemlast. Hybride Methoden sind in der Lage sich der jeweiligen Situation anzupassen.

2.3.3.3 Wertung

Die Prozessorenauslastung wird durch den Lastausgleich verbessert und damit auch die Effizienz der Parallelität. Der Lastausgleich erfordert z.B. zur Feststellung der Prozessorlast zusätzlichen Verwaltungsaufwand, der nur durch die Verbesserung der Prozeßbearbeitung

gerechtfertigt ist. Wird diese Verbesserung nicht erreicht, wirkt sich dieser Mehraufwand negativ auf die Effizienz aus. Beim dynamischen Lastausgleich muß sichergestellt werden, daß keine zirkuläre „process migration“ stattfindet, bei der die Prozesse von einem zum nächsten PE nur weitergereicht werden. Die Umgruppierung von Prozessen ist ein erheblicher Aufwand und kann nur bei grobkörnigen Prozessen effizient sein. Die Rentabilität dieser Operation kann aber während der Laufzeit vor ihrer Ausführung nicht bestimmt werden. Die Lastbalancierung reagiert erst dann, wenn die Prozessorlast bereits den Schwellwert übersteigt. Dieser späte Zeitpunkt ermöglicht nur die Reaktion auf einen Mißzustand. Eine vorausschauende Vergabe ist ohne genaue Kenntnis über das Laufzeitverhalten der Prozesse und der Last der einzelnen PEs nicht möglich. Falls das System überlastet ist, kann der Lastausgleich nur einen Mißstand verwalten, aber keine Verbesserung erzielen und schaden durch den zusätzlichen Verwaltungsaufwand.

2.3.4 Message-Passing Programmiermodell

Für verteilte Systeme ist das Message-Passing Modell am weitesten verbreitet. Die unabhängigen, asynchron laufenden Prozesse kommunizieren durch den Austausch von Nachrichten (Message-Passing). Falls der Prozeß nicht auf Nachrichten wartet, ist er aktiv. Das Konzept ist prinzipiell unabhängig von der Hardware und auch auf leistungsschwächeren Netzwerken, abhängig von der Anwendung, noch sinnvoll.

2.3.4.1 Basisfunktionalität

Neben der Möglichkeit Prozesse auf anderen PEs zu generieren, muß das System hauptsächlich die Kommunikation zwischen den Prozessen synchronisieren und steuern. Auch die Möglichkeit im Programm die Charakteristika des verteilten Systems abzufragen ist oft notwendig. Diese Funktionalität kann durch verschiedene Ansätze erreicht werden:

- Parallele Programmiersprachen, die diese Funktionalität enthalten (Ada, Modula-2, Occam)
- Spracherweiterungen für bestehende Programmiersprachen (MIMD-Fortran)
- Compilerdirektiven
- Unterprogrammbibliotheken, die bestehende Programmiersprachen ergänzen. (PVM, MPI)

2.3.4.2 Interprozeßkommunikation

Die Kommunikation zwischen den einzelnen Prozessen kann synchron oder asynchron erfolgen.

Bei der *synchronen, vollständigen Kommunikation* (*Rendezvous-Technik*) wartet der Sender solange, bis der Empfänger empfangsbereit ist. Erst dann wird gesendet. Damit synchronisiert jeder Sendevorgang die beteiligten Prozesse. Durch das Warten wird Rechenzeit verschenkt und die Effizienz herabgesetzt.

Die *asynchrone, unvollständige Kommunikation* (*Send-no-Wait-Technik*) puffert die Nachricht eines Senders, der sofort weiterarbeiten kann. Der Empfänger holt sich die Nachricht

dann später ab. Die Pufferungstechnik ist aufwendig und benötigt einen „Startup“. Es wird aber keine Rechenzeit verschenkt.

2.3.4.3 Master-Slave Konzept

Das Master-Slave Konzept (bzw. Host-Node Modell) ähnelt einem Hauptprogramm, das parallele Nebenprogramme aufruft. Der „Master“-Pozeß erzeugt „Slave“-Prozesse. Die „Slaves“ kommunizieren mit dem „Master“ und eventuell auch untereinander. Bei Shared-Memory-Systemen wird der Nachrichtenaustausch durch den gemeinsamen Speicher ersetzt. Im Unterschied zum Hauptprogramm, das ein Unterprogramm aufruft, kann der Master nach dem Senden der Aufgabe an den Slave parallel weiterarbeiten. Als Beispiel eines Master-Slave Konzeptes dient die trivial-parallele Anwendung einer Integration. Zwischen den einzelnen Prozessen bestehen Abhängigkeiten, die Datenaustausch und Kommunikation erfordern. Die Netzplantechnik verwendet Ende-Anfangsbeziehungen, um zeitliche Abhängigkeiten von Prozessen zu beschreiben. Die EA-Beziehung ist in diesem Modell mit der benötigten Zeit für die Kommunikation gleichzusetzen. Ausgehend von diesen Zusammenhängen kann, analog zu Vorgehensweisen der Baubetriebslehre, ein kritischer Weg bestimmt werden und der Prozeß-Ablauf optimiert werden. Problematisch sind hierbei die Werte für die Bearbeitungs- und Kommunikationszeiten, die besonders auf einem Workstationcluster sehr stark schwanken. Eine automatisierte Analyse ist nach einem Programmlauf mit Hilfe von „Trace“-Dateien möglich, die Daten über den Programmlauf und insbesondere die Interprozeßkommunikation beinhalten.
In Abbildung 10 und Abbildung 11 wird anhand eines Netzplanes und der alternativen Darstellung in einem Balkenplan gezeigt, wie die einzelnen Prozesse einer parallelen Integration zusammenhängen. (vgl. Anhang)

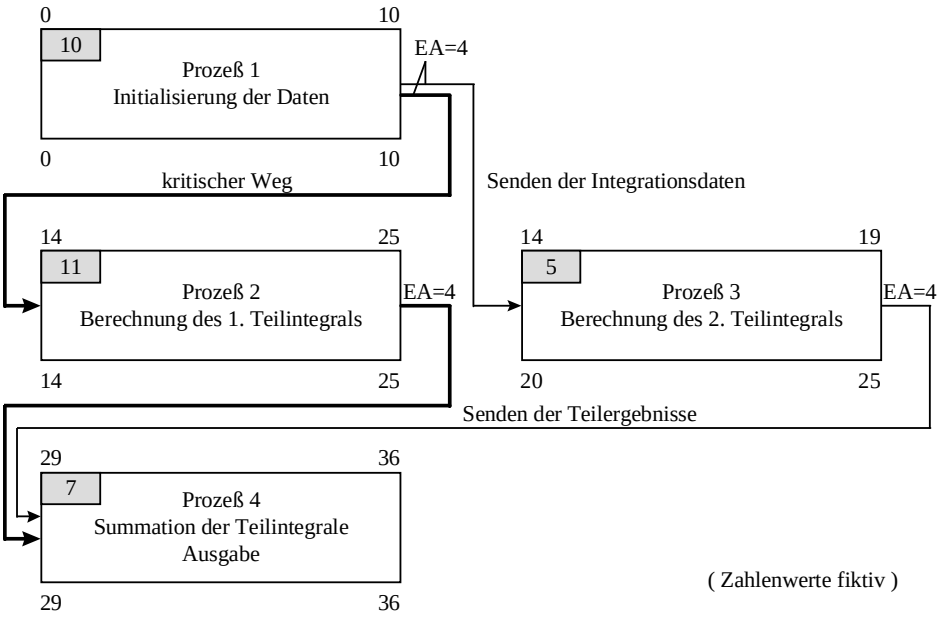


Abbildung 10: Netzplan einer parallelen Integration

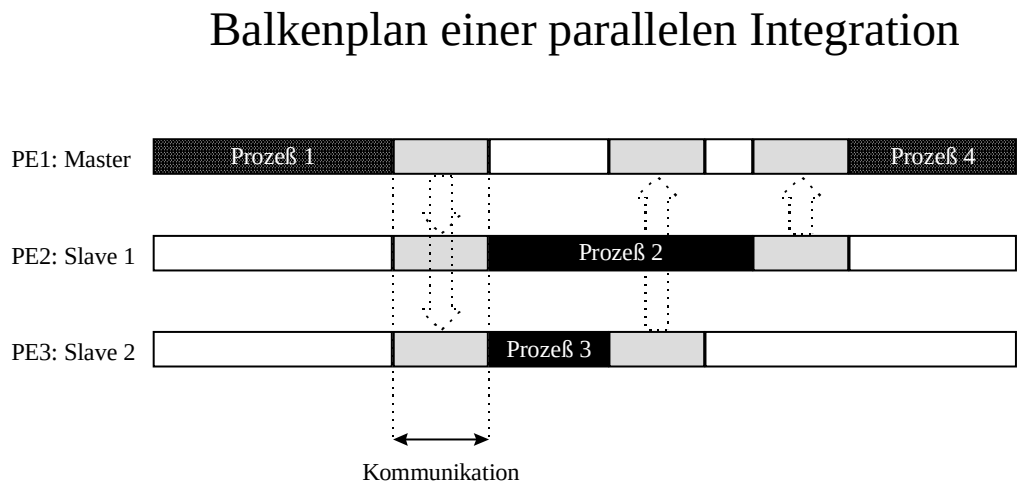


Abbildung 11: Balkenplan zu Abbildung 10

2.3.4.4 SPMD Konzept

Bei dem in 2.1.1.2 beschriebenen SPMD-Konzept sind alle Tasks gleichberechtigt. Ein Task kann die zusätzlichen Aufgaben, wie z.B. das User-Interface, übernehmen.

2.3.5 Workstation-Cluster

Vernetzte Workstations sind weit verbreitet und stellen somit einen größeren Markt wie die speziellen Parallelrechner dar. Die Kapazitäten der einzelnen Rechner sind oft nicht voll ausgenutzt. Die Betrachtung eines Workstation-Netzwerks als parallelen Rechner ist faktisch möglich (siehe Abbildung 12). Es müssen allerdings Unterschiede zwischen dieser Rechnerkategorie und den Parallelrechnern beachtet werden.

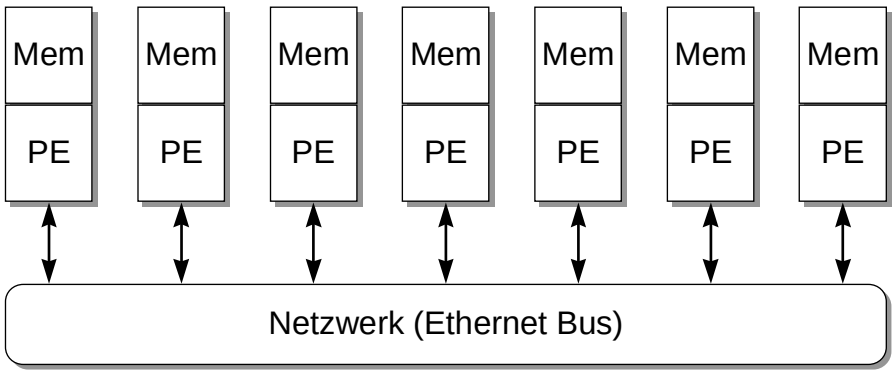


Abbildung 12: Schematische Darstellung eines Workstationclusters

2.3.5.1 Vernetzung

Workstationcluster sind meist in lokale Netzwerke eingebunden, deren Leistungsfähigkeit weit unter den spezialisierten Netzwerken eines Parallelrechners liegen. Bussysteme treten an Stelle von Punkt-zu-Punkt Verbindungen (siehe 2.2). Insbesondere wird, aufbauend auf Protokollen wie TCP/IP, keine Optimierung in Bezug auf die Latency (siehe 2.2) betrieben.

Für Netzdienste wie die Übertragung von Dateien oder das Remote Login ist die Anlaufzeit für die Übertragung auch nicht so wichtig wie für das Message-Passing. Es sollte also selten und dann in großen Blöcken kommuniziert werden. Die Granularität der Prozesse sollte damit möglichst grobkörnig sein.

2.3.5.2 Dedizierter Betrieb versus allgemeiner Betrieb

Beim dedizierten Betrieb wird der Workstationcluster zu einem Zeitpunkt entweder als Paralles System oder für den skalaren Betrieb eingesetzt. Das kann erreicht werden, wenn nur auf dem Gateway-Rechner interaktiver Betrieb zugelassen wird. Damit ähnelt der Cluster einem Parallelrechner mit Front-End/Back-End Betrieb. Da eine verteiltes Betriebssystem für einen Workstationcluster noch nicht entwickelt wurde, fehlt die globale Übersicht über die Prozessorlast der einzelnen Knoten. Eine Lastbalancierung nach 2.3.3 sollte vorgenommen werden.

Wird beim Workstationcluster im nicht-dedizierten Betrieb die parallele Nutzung mit der regulären Nutzung der einzelnen Rechner vermischt, sind die Verhältnisse unvorhersehbar. Insbesondere kann die Prozessorenlast nicht a priori bestimmt werden. Dadurch ist eine angepaßte, benutzergesteuerte Lastbalancierung, die am effizientesten ist, schwer verwirklichbar. Am sinnvollsten erscheint eine dynamische Lastverteilung (siehe 2.3.3.2) mit dem damit verbundenen erhöhten Kommunikationsbedarf. Die geringe Kommunikationsleistungsfähigkeit, die von Workstationclustern bereitgestellt wird, wirkt sich einschränkend aus. Die Scheduler der einzelnen Rechnerbetriebssysteme unterscheiden nicht zwischen parallelen und lokalen Prozessen und können somit nicht in den Lastverteilungsprozeß miteinbezogen werden. Die Parallelverarbeitung ist für dieses Umfeld problematisch!

3 Rechnernetze

3.1 Netztypen

Workstations werden im LAN (Local Area Network = lokales Netzwerk) üblicherweise mit den in Kapitel 2.2.2 beschriebenen busartigen Verbindungen auf Basis von Ethernet, Token-Ring, FDDI oder ATM vernetzt. Sie unterscheiden sich in Leistungsfähigkeit und Preis. Einen Überblick über die Übertragungsgeschwindigkeiten gibt Tabelle 4:

Netztyp:	Übertragungsrate [Mbps]:
Ethernet	10÷15
Token-Ring	10
FDDI	100
ATM	<620

Tabelle 4: Übertragungsraten der LAN-Netzwerktypen

Am weitesten verbreitet ist der preisgünstige Ethernetbus. Hier ergeben sich, aufgrund der geringeren Übertragungsraten Einschränkungen für parallele Anwendungen.

3.2 OSI-Referenzmodell

Das „Open Systems Interconnection reference model“ stellt das ISO[*]/IEC[†] -Referenzmodell für offene Systeme dar. Dieses Modell für geschichtete Kommunikationssysteme definiert Basiskonzepte und Terminologie. Es besteht aus den in Tabelle 5 dargestellten sieben Schichten mit jeweiliger Funktionalität. Mit Ausnahme der Bitübertragungsschicht werden die Daten in jeder Schicht in ein Protokoll eingebettet (Encapsulation). Somit erhält man geschachtelte „Protocol Data Units“ (PDUs) und bei jeder Schicht einen Protokoll-Header mehr. Ein dazwischengeschaltetes System entschlüsselt eine Nachricht bis zur Vermittlungsschicht und verpackt die Nachricht anschließend wieder, um sie weiterzusenden. Im einfachen Workstationcluster ist ein solches „*Routing*“ nicht nötig. Die einzelnen Schichten sind voneinander abgegrenzt und rufen sich über definierte Schnittstellen auf. So muß die Applikationsschicht nichts über den eigentlichen Transport- und Übertragungsvorgang wissen. Die Komplexität und die klare Abgrenzung in eine Vielzahl von Schichten hat allerdings den Nachteil, daß Anwendungen verlangsamt werden und schwerer zu realisieren sind.

[*] International Standardization Organisation (http://www.iso.ch/)
[†] International Electrotechnical Commission (http://www.iec.ch/)

	End system	Peer to Peer Protokol	End system	
7	Anwendungsschicht (Application Layer)		Anwendungsschicht (Application Layer)	7
6	Darstellungsschicht (Presentation Layer)		Darstellungsschicht (Presentation Layer)	6
5	Kommunikationssteuerungsschicht (Session Layer)		Kommunikationssteuerungsschicht (Session Layer)	5
4	Transportschicht (Transport Layer)		Transportschicht (Transport Layer)	4
3	Vermittlungsschicht (Network Layer)		Vermittlungsschicht (Network Layer)	3
2	Sicherungsschicht (Data Link Layer)		Sicherungsschicht (Data Link Layer)	2
1	Bitübertragungsschicht (Physical Layer)		Bitübertragungsschicht (Physical Layer)	1
		Medium		

Tabelle 5: OSI-Referenzmodell

3.2.1 Bitübertragungsschicht (OSI-Layer 1)

Die Bitübertragungsschicht spiegelt die Übertragungsmedien wie Koax- oder Glasfaserkabel wieder. Auch die Computerhardware, wie Netzwerkkarten, gehört dieser Schicht an. Es wird nur Sorge dafür getragen, daß ein Bitstrom übertragen wird. Die Übertragung kann analog oder digital erfolgen.

3.2.2 Sicherungsschicht (OSI-Layer 2)

Die Sicherungsschicht kontrolliert die Übertragung. Der L1-Dienst stellt sicher, daß Reihenfolgefehler im Bitstrom verhindert werden, während der L2-Dienst eine zuverlässige, effiziente Datenübertragung zwischen zwei benachbarten, direkt verbundenen Rechner gewährleistet. Zu diesem Zweck baut auf der Datenübertragung eine Fehler- und Flußkontrolle, sowie ein Konfigurationsmanagement auf. Während die Fehlerkontrolle für fehlerfreies Senden zuständig ist, sorgt die Flußkontrolle über eine Rückkoppelung von Sender und Empfänger für eine Synchronisation.

Bei den L2-Dienstklassen wird zwischen einem verbindungslosem und verbindungsorientiertem Dienst unterschieden. Beim bestätigten verbindungslosen Dienst wird durch die Bestätigung des Empfängers ein Verlust von Datenpaketen verhindert. Das PAR (Positive Acknowledgement with Retransmit) -System kann aber zu Reihenfolgefehlern und doppelten Paketen führen. Es existiert also keine Flußkontrolle. Beim verbindungsorientierten Dienst hingegen werden diese Fehler durch eine Flußkontrolle verhindert. Dazu ist eine dreiphasige Kommunikation nötig:

- Verbindungsaufbau
- Datentransfer
- Verbindungsabbau

Die verwendeten Protokolle können zeichen- , zähler- oder bitorientiert sein, wobei die Mehrzahl der heutigen Protokolle bitorientiert sind. Dies ermöglicht eine codeunabhängige Übertragung, wobei als Blockbegrenzung ein Flag (01111110) verwendet wird. Dieses Flag darf nicht in den zu übertragenden Daten enthalten sein. Aus diesem Grund werden die zu sendenden Daten durch „Bit stuffing“ modifiziert: Nach fünf aufeinanderfolgenden „1“-Bits wird ein „0“-Bit eingefügt und beim Empfänger wieder entfernt.

Das „Stop-and-Wait“-Protokoll mit PAR und Sequenznummern (siehe Abbildung 13) ist ein Beispiel für einen bestätigten verbindungslosen L2-Dienst. Die Nachricht wird in Teilnachrichten (Frames) aufgeteilt, die jeweils durch eine eindeutige Sequenznummer identifizierbar sind. Der Empfänger bestätigt ein empfangenes fehlerfreies Frame mit einer Positivantwort (ACKnowledgement) oder im Falle eines Übertragungsfehlers mit einer Negativantwort (Negative AcKnowledgement). Geht eine Bestätigung verloren, wird der Sender dazu veranlaßt dieses Frame nochmals zu senden. Die Sequenznummer ermöglicht dem Empfänger die nunmehr doppelten Daten auszusortieren. Der Vorteil dieses Protokolls ist ein geringer Pufferbedarf. Zudem ist es für einfache Endgeräte geeignet. Dem gegenüber steht eine schlechte Kanalauslastung und ein geringer Datendurchsatz.

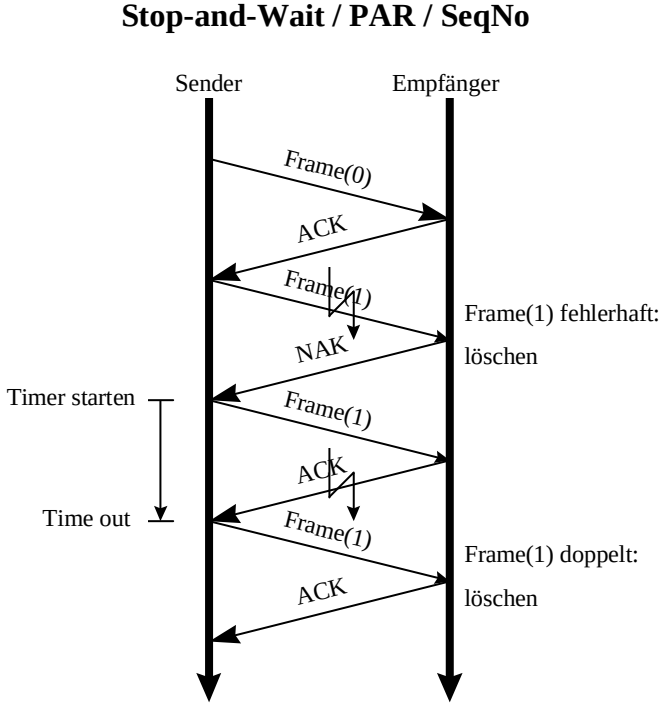


Abbildung 13: Stop-and-Wait/PAR/SeqNo Protocol

Ein weiteres Protokoll, das „Sliding Window“-Protokoll bietet bei hohem Pufferbedarf einen hohen Datendurchsatz. Einzelheiten hierzu sind in [8] nachzulesen.

3.2.2.1 Ethernet

Das Ethernet-LAN richtet sich nach dem IEEE* 802.3 Standard, der in der Bitübertragungs- und Sicherungsschicht des OSI-Referenzmodells angesiedelt ist. Dieser Standard beschreibt das 1-persistent CSMA/CD (Carrier Sense Multiple Access with Collision Detection) - System, das Übertragungsraten von 1-10 Mbps erreicht. Beim CSMA hört die sendewillige Station den Kanal ab und sendet nur, wenn er frei ist. Bei der 1-persistent Variante wartet die Station bis der Kanal frei ist um dann zu senden. Für den Fall einer Kollision während

* Institute of Electrical and Electronics Engineers (http://www.ieee.org/)

der Übertragung muß vor einem erneuten Versuch eine Zufallszeit abgewartet werden. Bei konstanter Wartezeit würden beide Sender immer wieder gleichzeitig senden und weitere Kollisionen auslösen. Die „Collision Detection“ erweitert das Modell so, daß die sendende Station eine Übertragung unterbricht, sobald sie eine Kollision feststellt. Damit wird Zeit und Bandbreite eingespart. Dies entspricht dem in Kapitel 2.2.2.1.1 beschriebenen Random-Access Zugriffsmechanismus.
Die Ethernet-Hardware verwendet Transceiver, um die Netzwerkkarte des Computers über ein Transceiverkabel mit dem Bus-Ethernetkabel zu verbinden. Im Transceiver steckt eine Elektronik für die Träger- und Kollisionserkennung. Das Ethernetkabel ist in verschieden Varianten erhältlich: Thin- oder Thick-Ethernet. Falls die Verbindungsleitungen zu lang sind, können bis zu vier Repeater zur Signalverstärkung zwischengeschaltet werden.

3.2.3 Vermittlungsschicht (OSI-Layer 3)

Die Vermittlungsschicht dient dazu, Rechner zu verbinden, die keine direkte Verbindung besitzen (Routing). Ein Netzwerk kann so aus vielen Subnetzen aufgebaut werden. Der L3-Dienst dieser Schicht kann als verbindungsloser oder verbindungsorientierter Dienst ausgeführt werden. Prinzipiell entsprechen diese Dienste den L2-Diensten der Sicherungsschicht. Innerhalb eines Netzes hat OSI-Layer 3 allerdings keine große Bedeutung.

3.2.4 Transportschicht (OSI-Layer 4)

Die Aufgabe der Transportschicht ist die Bereitstellung eines zuverlässigen und kostengünstigen Datentransports zwischen zwei Systemen. Die Eigenschaften des verwendeten Netzwerkes bleiben dabei unberücksichtigt.

3.2.5 Kommunikationssteuerungsschicht (OSI-Layer 5)

Der organisierte, synchronisierte Datenaustausch steht im Mittelpunkt dieser Schicht. Hierfür muß zuerst eine Verbindung aufgebaut werden, dann kann der synchronisierter Datenaustausch stattfinden und abschließend muß die Verbindung ordentlich abgebaut werden. Eine solche Sitzung wird über Tokens gesteuert.

3.2.6 Darstellungsschicht (OSI-Layer 6)

Hier wird die Datenkonvertierung in heterogenen Systemen vorgenommen. Kommunikationssteuerungsdienste werden durchgereicht. Komplexe Datenstrukturen können definiert und die Daten selbst in eine globale Darstellung konvertiert werden.

3.2.7 Anwendungsschicht (OSI-Layer 7)

Grundlegende Anwendungen wie Remote Operationen, Dateizugriffe oder Electronic Mail werden hier bereitgestellt.

3.3 TCP/IP

Die ARPA* begann 1969 mit einem kleinen Experimentiernetzwerk das ARPANET aufzubauen. Ziel war es, wissenschaftliche und militärische Einrichtungen der Vereinigten Staaten zu verbinden. Aus diesem Netz entwickelte sich das Internet. Als Protokoll für die Kommunikation zwischen den Netzen wurde TCP/IP entwickelt und 1983 genormt. Das Internet bestand am Anfang aus militärischen Netzen, Satellitennetzen und Universitäts-LANs. Die TCP/IP-Technologie bildet die Grundlage des Betriebssystems UNIX, was zu einer weiteren Verbreitung führte.
Das IP-Modell hat im Vergleich zum OSI-Referenzmodell weniger Schichten. Damit ist es schneller, direkter in der Ausführung und einfacher zu implementieren. Dies mag auch ein Grund dafür sein, warum sich das OSI-Referenzmodell noch nicht durchsetzen konnte. Die fünf Schichten der Internet-Architektur sind in Abbildung 14 dargestellt. Das „Physical Net“

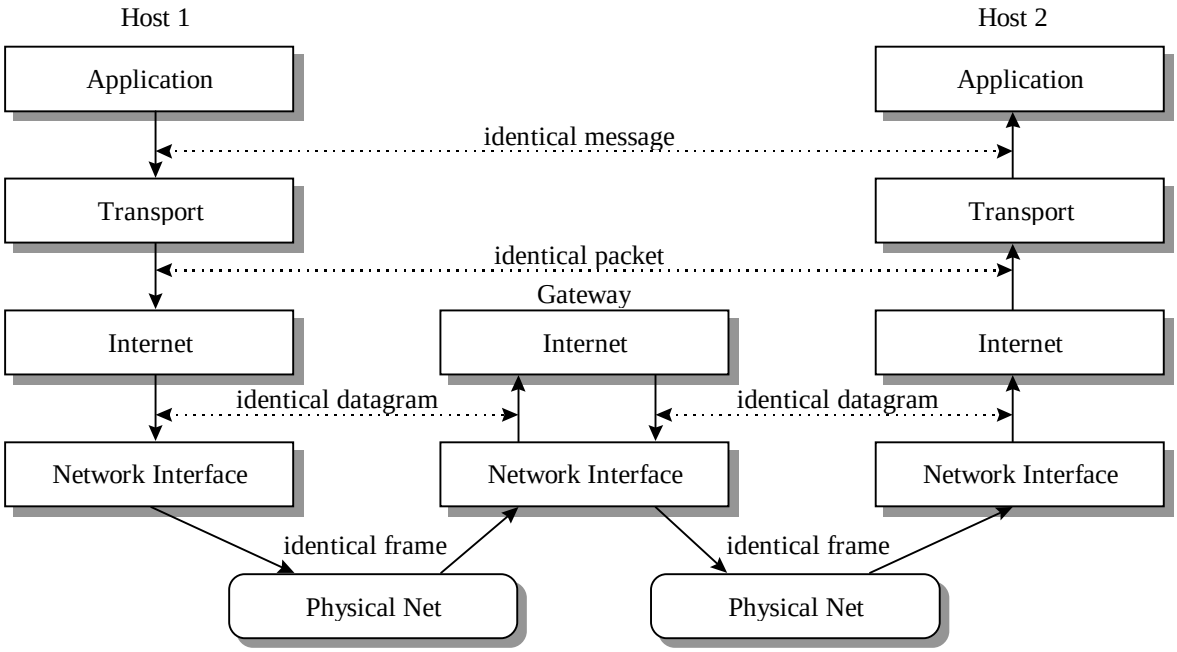


Abbildung 14: Internet Architektur

entspricht der OSI-Bitübertragungsschicht (L1), das „Network Interface“ kann mit der Sicherungsschicht (L2) verglichen werden und die „Internet“-Schicht entspricht der Vermittlungsschicht (L3). „Transport“ und „Application“ sind gleichnamig im OSI enthalten. Die OSI-Schichten 5 und 6 (Kommunikation und Darstellung) sind in der Internet-Architektur nicht zu finden.

* Advanced Research Projects Agency des U.S. Department of Defense (DoD)

Den einzelnen Schichten sind bestimmte Protokollarten zugeordnet. Der Aufbau der verschieden Internet-Protokolle wird in Tabelle 6 aufgeführt.

SMTP Simple Mail Transfer Protocol	FTP File Transfer Protocol	TELNET Remote Login Protocol		NFS Network File System
TCP Transmission Protocol				UDP User Datagram Protocol
IP Internet Protocol				
LLC Logical Link Control			MAC Media Access Control	
Physical				

Tabelle 6: Internet Protokolle

3.3.1 Internet Protocol (IP)

Das IP der Vermittlungsschicht übernimmt basierend auf dem verbindungslosen Dienst die Adreßauflösung und das Routing. Die physikalische Netzadresse der Adapterkarte wird auf eine Internetadresse abgebildet. Falls sich das Zielsystem im gleichen Subnetz befindet, kann die Auflösung im Quellsystem erfolgen (direct routing), ansonsten wird die Nachricht an das Gateway weitergeleitet. Dort wird die physikalische Adresse entweder einer gespeicherten Tabelle entnommen oder über ein Address Resolution Protocol (ARP) erfragt und in einem Cache zwischengespeichert. Das IP kontrolliert nicht, ob Daten fehlerfrei übermittelt werden. Dies bleibt höheren Schichten vorbehalten. Das TCP und UDP der Transportschicht baut auf dem IP auf.

3.3.2 Transmission Control Protocol (TCP)

TCP stellt die verbindungsorientierte Variante der Kommunikation im Internetmodell dar. Analog zum Telefon wird die Verbindung aufgebaut, Daten ausgetauscht und die Verbindung wieder abgebaut. Die Anzahl der Anschlüsse, bzw. Sockets ist hierbei begrenzt. Vom TCP wird eine Fehlerkontrolle und -behebung bereitgestellt und bei Zeitüberschreitung ein Fehler ausgelöst. Somit ist es ein „sicheres“ Protokoll.

3.3.3 User Datagram Protocol (UDP)

UDP unterstützt eine verbindungslose Kommunikation. Das kann mit dem Versenden von Briefen mit der Post verglichen werden. Die Nachricht wird in einzelne Pakete (Frames) zerteilt, unabhängig voneinander verschickt und wieder zusammengebaut. Es erfolgt allerdings keine Fehlerkontrolle. Reihenfolgefehler und Fehler durch Duplikate oder verlorene Pakete müssen von der Applikationsschicht behoben werden. Auch Zeitüberschreitungen werden nicht geprüft. Der Verwaltungsaufwand für dieses „unsichere“ Protokoll ist aber gering und es wird insgesamt nur ein Socket benötigt.

4 PVM (Parallel Virtual Machine)

4.1 Übersicht

PVM ist eine über „netlib“ frei verfügbare Programmbibliothek für C- und Fortran-Programme. Entwickelt wird diese Software vom Oak Ridge National Laboratory, der University of Tennessee, der Emory University und der Carnegie Mellon University. Das System ermöglicht es, mehrere vernetzte Computer als einen (virtuellen) Parallelrechner zu verwenden. PVM baut auf dem Message-Passing Modell auf und ist somit für lose gekoppelte heterogene verteilte Systeme mit grobgranularen Anwendungen geeignet. Die einzelnen Rechner können von unterschiedlicher Bauart sein: PCs, Workstations oder auch Mehrprozessorrechner arbeiten in einer Konfiguration zusammen. Die Verbindungsstruktur ist beliebig. Eine Heterogenität ist nicht nur bei der Architektur der Einzelrechner, sondern auch beim zugrundegelegten Datenformat, der Rechenleistung, sowie der jeweiligen Prozessor- und Netzwerklast möglich. Da das System für die unterschiedlichsten Plattformen existiert, ist die Portabilität sehr hoch. Um den Einsatz in einem heterogenen Netz zu gewährleisten, sind die Portierungen für verschiedene Architekturen voll kompatibel zueinander. Bei der Programmentwicklung wurde außerdem auf die Skalierbarkeit geachtet. Die Anzahl der Einzelrechner des virtuellen Parallelrechners ist in der verwendeten Version 3.3 mit 4095 „Hosts“ limitiert. Diese Schranke wird im praktischen Einsatz nicht erreicht. Die Konfiguration des virtuellen Parallelrechners kann während des Programmlaufs dynamisch verändert werden. Durch Hinzufügen oder Herausnehmen von Einzelrechnern ist die Konfiguration auf die Bedürfnisse der Anwendung anpassbar. Es liegt in der Hand des Programmierers auch fehlertolerante Anwendungen mit Hilfe von PVM zu erzeugen. Hierzu müssen Rückgabewerte der Funktionen, sowie PVM-Nachrichten über verlorengegangene Prozesse ausgewertet werden. Prozesse können auch Gruppen zugeordnet werden, die sich dynamisch während des Programmlaufs ändern lassen. Spezielle Gruppenfunktionen, wie der Broadcast oder die Barrierefunktion, sind auf eine spezielle Gruppe anwendbar. Ein Prozeß kann hierbei auch mehreren Gruppen angehören. Ein von UNIX übernommenes Feature ist das Senden von Signalen an Prozesse. So kann ein Prozeß z.B. mit einem Kill-Signal terminiert werden. Für die Kommunikation werden Pufferspeicher verwendet, wobei mehrere Puffer möglich sind. Hierauf wird im folgenden noch näher eingegangen. Zum Debugging können gängige Debugger eingesetzt und über Optionen ein Tracing des Programms erzeugt werden.

4.2 Funktionsweise

4.2.1 Kommunikation

Bei Message-Passing Systemen steht die Kommunikation zwischen den Prozessen im Vordergrund. PVM steuert die Prozeßkommunikation mit Hilfe eines Dämons, der auf jedem Einzelechner gestartet wird. Ein Dämon ist ein im Hintergrund ablaufendes Programm ohne Benutzerschnittstelle. Über Dämonen werden auch Dienste wie Electronic-Mail, Netzwerk-Druckmanagement und verteilte Dateisysteme gesteuert. Der PVM-Dämon kommuniziert mit den jeweiligen Tasks des Computers über eine TCP-Verbindung. Die Dämonen untereinander hingegen verwenden UDP-Verbindungen, um Nachrichten auszutauschen. Dies wird in Abbildung 15 verdeutlicht. Die Gründe hierfür ist in den Charakteristika der Proto-

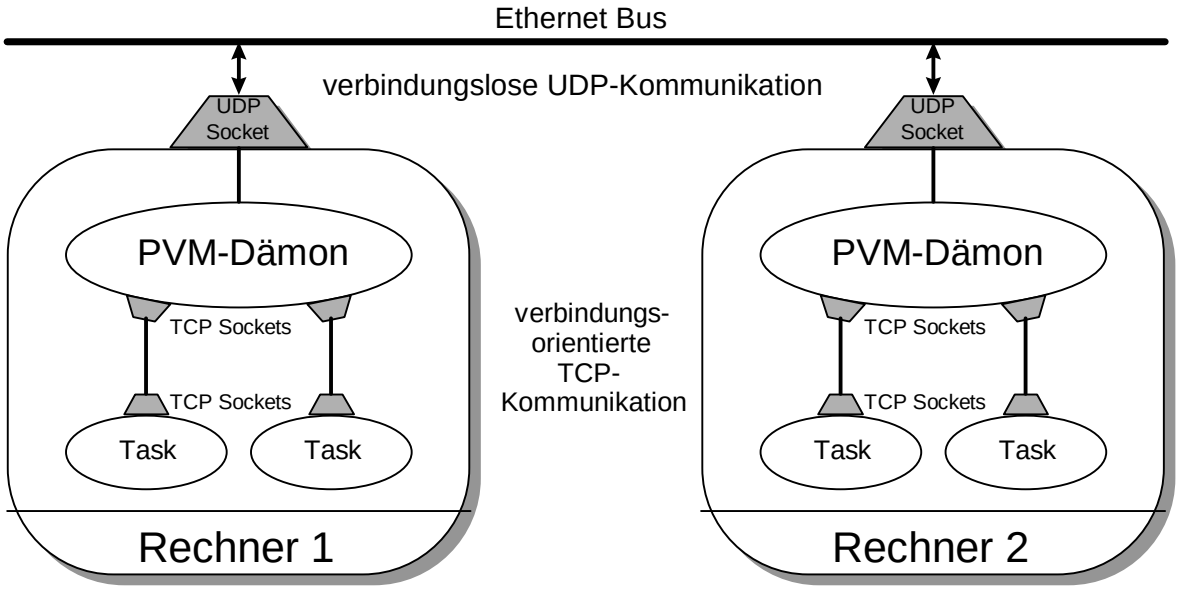


Abbildung 15: PVM Kommunikation

kolle (siehe 3.3) zu suchen. Die Internet-Protokoll-Familie ist sehr weit verbreitet und in den UNIX-Systemen sogar als Bestandteil des Betriebssystems vorhanden. Deshalb wurden diese Protokolle verwendet, um die Portierung auf verschiedene Platformen zu erleichtern.
Für die Kommunikation zwischen den PVM-Dämonen würde die Verwendung von TCP-Verbindungen folgende Einschränkungen mit sich bringen:

- Für jede TCP-Verbindung wird vom Dämon ein „File descriptor“ oder „Socket“ verbraucht. Aus diesem Grund erhält man bei einigen Architekturen ein Limit von 64 TCP-Verbindungen, die zugleich offen sein können. Dies schränkt die Skalierbarkeit des Systems ein. Für einen Workstationcluster mit weniger als 32 Rechner ergeben sich zwar noch keine Probleme, Mehrprozessorrechner sprengen aber diese Grenzen.
- Ein virtueller Parallelrechner, bestehend aus n Rechnern würde bis zu $\frac{n(n-1)}{2}$ Verbindungen benötigen (siehe2.2.1.2), um alle Dämonen miteinander zu verbinden.

- Um Fehlertoleranz zu ermöglichen, muß der Absturz von Tasks oder Rechnern festgestellt werden. Hierzu muß eine Zeitschranke auf Protokollebene gesetzt werden. Die Umsetzung mit der TCP „keep alive“-Option brachte Schwierigkeiten mit sich (siehe [10]).

Für UDP-Verbindungen hingegen benötigt man nur einen Socket pro Rechner und Verbindungen werden nicht aufgebaut. Da UDP als unsicheres Protokoll keine Gewährleistung für die korrekte Übertragung der Daten bietet, mußte eine zusätzliche Fehlerkontrolle und -behebung seitens PVM implementiert werden. Hierfür werden vom Empfänger Bestätigungsnachrichten zurückgesendet. Die gesendeten Pakete, wie auch die Bestätigungsnachrichten, sind durch Sequenznummern eindeutig identifizierbar. Diese Methode kann mit Abbildung 13 verglichen werden, wobei ein sequentielles Versenden der Pakete aufgrund der Sequenznummern der „ACKnowledgement“-Nachrichten nicht zwingend ist.
Diese Strategie setzt sich für die Verbindung der Tasks zum PVM-Dämon nicht fort. Hier wird das Transmission Control Protokoll eingesetzt. Da TCP ein sicheres Protokoll darstellt und nicht wie UDP sogar innerhalb eines Rechners Pakete verliert, muß keine zusätzliche Fehlerbeseitigungsinstanz eingebaut werden. Allerdings ist eine Konvertierung zwischen dem TCP-Datenformat und den UDP-Datenpaketen nötig. Das bereits angesprochene Limit für die Anzahl gleichzeitig verwendbarer TCP-Verbindungen aufgrund der begrenzten Anzahl an Sockets wirkt sich nicht auf praktische Anwendungen aus: Es ist aus Effizienzgründen nicht sinnvoll zu viele Tasks auf einem Rechner im Multitaskingbetrieb abarbeiten zu lassen.

4.2.1.1 Broadcast

Ein echter Broadcast (siehe 2.2.2.1.1) ist in PVM nicht verwirklicht. Das simultane Senden von Daten an mehrere Rechner wird sequentiell, kurz hintereinander durchgeführt. Eine optimierte Abarbeitung gewährleistet bei der Broadcast-Funktion Geschwindigkeitsvorteile gegenüber dem separaten Senden der Nachricht.

4.2.1.2 Direkte Verbindung

Diese beschriebene Struktur stellt die Standardkommunikation unter PVM dar. In der aktuellen Version kann durch Setzen spezieller Parameter auch eine direkte TCP-Verbindung zur Kommunikation aufgebaut werden. Diese Verbindung existiert auch nach der Datenübertragung noch bis zum Programmende oder dem Erreichen des Limits für die Anzahl der TCP-Sockets. Bei häufiger Kommunikation zwischen bestimmten Rechnern oder einer kleinen Konfiguration muß so die TCP-Verbindungen nicht neu aufgebaut werden. Die verbindungsorientierte direkte Kommunikation ist hier eine Alternative zum traditionellen PVM-Modell.

4.2.1.3 Nachrichtenpuffer

Daten, die verschickt werden sollen, werden in einen Pufferbereich (message buffer) kopiert. Damit wird zum einen sichergestellt, daß bis zum Versenden der Nachricht keine Än-

derungen mehr vorgenommen werden und zum anderen ermöglicht, daß eine Nachricht aus unterschiedlichen Datenpaketen zusammengepackt werden kann. Dies kann durch die „InPlace“ Option umgangen werden, was allerdings keine nennenswerten Geschwindigkeitssteigerungen bringt.

4.2.1.4 Datenformat

Sollen Daten zwischen Rechnern mit unterschiedlicher interner Darstellung von Daten ausgetauscht werden, ist eine Konvertierung nötig. Im Internet-Modell fehlt die Darstellungsschicht (siehe 3.2.6) des OSI-Modells. Aus diesem Grund verwendet PVM das XDR (eXternal Data Representation standard) -Format als übergeordnete Darstellung. Falls die an der Datenübertragung beteiligten Rechner keine Datenkonvertierung nötig machen, kann dieser Schritt umgangen und somit eine Geschwindigkeitssteigerung erreicht werden.

4.2.1.5 Empfangen von Nachrichten

PVM unterstützt unterschiedliche Arten des Nachrichtenempfangs:

- *Blockierendes Empfangen*
- *Nichtblockierendes Empfangen*
- *Blockierendes Empfangen mit Zeitbeschränkung*

Beim blockierenden Empfangen wird ein Programm solange gestoppt, bis die erwartete Nachricht eingetroffen ist. Nichtblockierendes Empfangen hingegen fragt ab, ob die Nachricht eingetroffen ist. Ist dies der Fall kann empfangen werden, ansonsten werden andere Instruktionen abgearbeitet. Hiermit kann unter Umständen die Effizienz gesteigert werden, da beim Warten auf Nachrichten Rechenleistung verschenkt wird. Das blockierende Empfangen mit Zeitbeschränkung stellt ein Bindeglied der beschriebenen Arten dar. Bei sehr langen Zeitbeschränkungen entspricht es dem blockierenden, bei sehr kurzen dem nichtblockierenden Empfangen. In fehlertolerante Anwendungen kann also vermieden werden, daß ein blockierendes Empfangen einen Deadlock verursacht, indem nach einer gewissen Zeit mit einem Fehlerstatus abgebrochen und der Fehler abgefangen wird.

4.2.2 Konfiguration

Es ist möglich, die einzelnen Dämonen auf jedem Einzelrechner von der Kommandozeile aus zu starten. Einfacher ist es auf die dynamischen Konfigurationsmöglichkeiten in PVM zurückzugreifen. Ein spezieller Dämon der Konfiguration, der Master-Dämon (auf dem Master-Host), wird zu Steuerungsaufgaben verwendet. Von dieser Steuerungszentrale aus können mit Hilfe von „remote shell“-Diensten weitere „Slave-Dämonen“ gestartet werden. Der Master-Dämon übergibt aus Timing-Gründen diese Aufgabe dem „Shadow-Dämon“, einem Programm, das auf dem selben Rechner gestartet wird. Soll während des Programmablaufs von einem Task ein zusätzlicher Rechner dynamisch zur Konfiguration aufgenommen werden, sendet der Task über den lokalen Dämon eine Anweisung an den Master-Dämon. Über einen Broadcast informiert der Master alle Dämonen über die veränderte Konfiguration.

Die Beendigung eines Dämons erfolgt analog. Der Dämon beendet alle ihm zugeordneten Tasks und versendet eine „Shutdown“-Nachricht bevor er terminiert. Falls ein Dämon den Kontakt zum Master-Dämon verliert, beendet er sich selbständig. Dies ist nötig, um zu vermeiden, daß sich die Konfiguration teilt und der abgespaltene Teil ein unkontrolliertes Eigenleben führt.

4.2.3 Resource manager (RM)

PVM verwendet ein spezielles Programm, welches das Verteilen von Tasks übernimmt: Vom „Resource Manager“ (RM) wird festgelegt, auf welchem Rechner ein neuer Task gestartet werden soll, falls kein spezieller Hostrechner spezifiziert wird. Ein RM ist immer dem Master-Dämon zugeteilt. Der Programmentwickler kann den Standard-RM durch eine eigene Implementierung ersetzen und somit die Lastbalancierung verbessern. Hierzu können auch zusätzliche Informationen, wie die Prozessorlasten der Einzelrechner, miteinbezogen werden. Jedem Dämon kann ein eigenständiger RM zugeordnet werden. Ansonsten ist der Master-Dämon RM zuständig. Ein neuer Task erbt den RM vom Elterprozeß.

4.2.4 Tasks

Tasks entsprechen auf UNIX-Rechnern Prozesse. Jedem Task wird von PVM eine eindeutige Identifikationsnummer (TID = Task IDentifier) zugeteilt. Jeder TID entspricht einer UNIX-Prozeßnummer (PID = Process Identifier). Der lokale PVM-Dämon, der für die Tasks zuständig ist, speichert und verwaltet sie in Listen. Eine Liste wird nach den TIDs geordnet und eine weitere nach PIDs. Alternativ zur Datenstruktur Liste ist für spätere Programmversionen eine Umsetzung als AVL-Baum in Erwägung gezogen worden.

4.3 Anwendung

4.3.1 PVM-Konsole

Die PVM-Konsole ist ein mitgeliefertes Programm mit dem ein PVM-Parallelrechner erzeugt und modifiziert werden kann. In einer Konfigurationsdatei legt man die zum virtuellen Parallelrechner gehörenden Rechner fest und konfiguriert einzelne Parameter, wie z.B. die Pfade zu den ausführbaren Programmen (siehe 4.3.1.2). Falls diese Parameter nicht angegeben werden, verwendet PVM Standardeinstellungen. Das Konsolenprogramm startet oder beendet die einzelnen Dämonen. Das ist zwar dynamisch von der Applikation aus möglich, es ist aber vor allem bei nicht-dedizierten Workstationclustern (siehe 2.3.5.2) vorteilhaft, dem Benutzer die Festlegung der Konfiguration zu überlassen. Außerdem ist es den Applikationen nicht möglich, die oben erwähnten Parameter zu ändern. Auf die von der Konsole gestartete PVM-Konfiguration kann von jeder PVM-Applikation zugegriffen werden.

Die Konsole wurde durch ein Skriptfile „pvms“ ergänzt, welches mit den Optionen „start“, „stop“ und „restart“ ein einfaches Starten und Beenden von PVM ermöglicht. Standardmäßig wird von einer Verzeichnisstruktur nach Abbildung 16 ausgegangen, wobei die Konfigu-

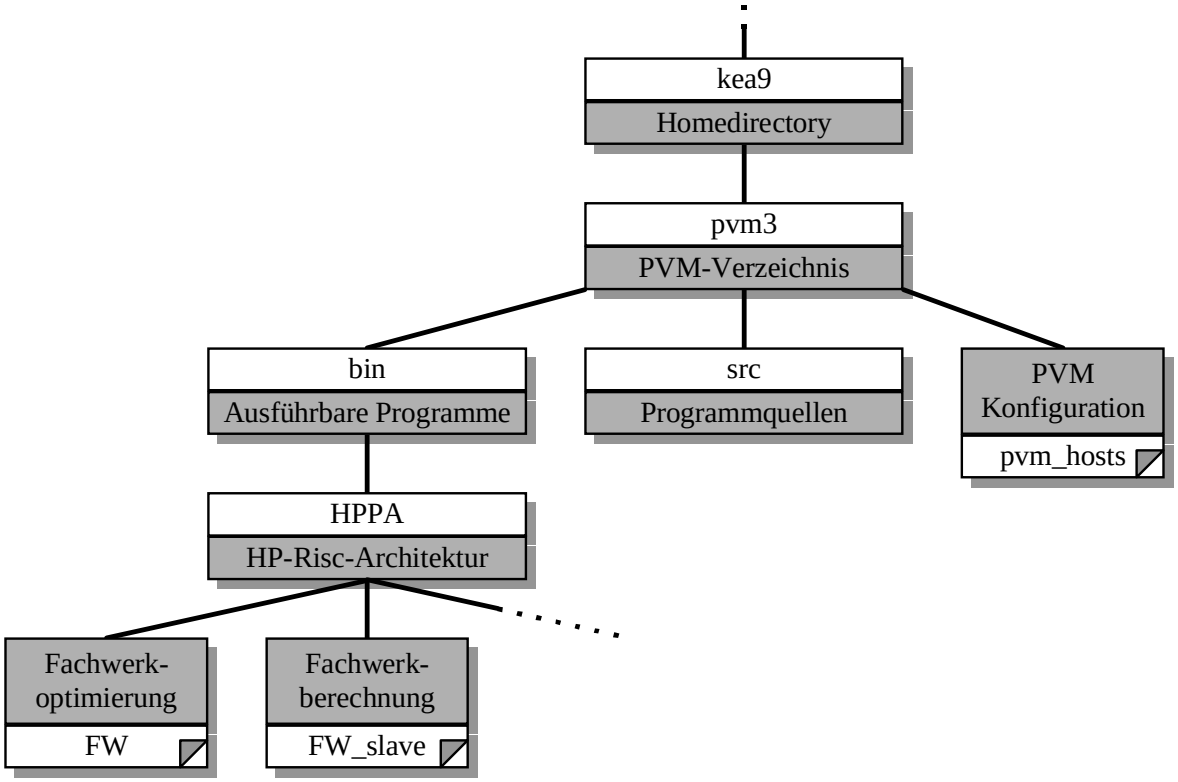


Abbildung 16: Beispiel einer PVM-Entwickler-Verzeichnisstruktur

ration des virtuellen Parallelrechners in der Datei „pvm_hosts“ definiert wird. Diese Datei wird von „pvms“ automatisch generiert, falls sie nicht vorgefunden wird. Die ausführbaren Programme sind nach Architektur getrennt in verschiedenen Verzeichnissen abgespeichert, um Namenskonflikte zu umgehen. Dies ist vor allem dann sinnvoll, wenn verschiedene Rechnerarchitekturen bei verteilten Dateisystemen (z.B. über NFS) auf dieselben Speichermedien zugreifen.

4.3.1.1 Umgebungsvariablen

Informationen, wie z.B. der Pfad zu den PVM-Programmen sind in Umgebungsvariablen abgelegt. Eine kurze Übersicht ist in Tabelle 7 enthalten:

PVM-Umgebungsvariable:	Beschreibung:
PVM_ROOT	PVM Installationsverzeichnis
PVM_ARCH	Rechnerarchitektur (→pvmgetarch)
PVM_DPATH	Pfad zum PVM-Dämon
PVM_EXPORT	zu exportierende Umgebungsvariablen

Tabelle 7: PVM Umgebungsvariablen

4.3.1.2 Konfigurationsdatei

Die folgende PVM-Konfigurationsdatei dient als Beispiel:

```
# pvm_hosts Beispieldatei
# Standard-Definitionen:
* ep=/bau/stat15/u/statik/kea9/pvm3/bin/HPPA
########################################
# CIP-Pool
bcip2 wd=/bau/stat15/u/statik/kea9/pvm3/bin/HPPA
bcip3
bcip4
bcip5
bcip6
bcip7
bcip8
bcip9
bcip10
bcip11
bcip12
bcip13
bcip14
bcip15
# Rechner, die noch hinzugefügt werden können
&bcip1
```

Die Parameteroptionen sind in Tabelle 8 aufgelistet. Falls keine Optionen angegeben werden, wird auf Standardeinstellungen zurückgegriffen.

Option:		Bedeutung:
#	(am Anfang der Zeile)	Kommentarzeile (Leerzeilen werden ignoriert)
*	(am Anfang der Zeile)	Standardbelegung für Optionen definieren
&	(am Anfang der Zeile)	Deklaration eines Rechners, Def. der spez. Optionen
dx =	location_of_pvmd	PVM-Dämon mit Pfadangabe
ep =	paths_to_user_executables	Suchpfade für PVM-Tasks
wd =	working_directory	Verzeichnis, in dem ein Task ausgeführt wird.
bx =	location_of_debugger	Debugger mit Pfadangabe
so =	pw	Passwortabfrage auf remote hosts (rexec)
so =	ms	Dämonen manuell starten
sp =	value	relative Geschwindigkeit setzen
		(wird momentan noch nicht von PVM verwendet)

Tabelle 8: PVM Hostfile Optionen

Für die Beispielkonfigurationsdatei wird von „pvms“ über die PVM-Konsole ein virtueller Parallelrechner mit den Einzelrechnern bcip2 - bcip15 erzeugt. Die Applikationsprogramme sind in dem ep-Verzeichnis zu suchen. Diese Definition gilt für alle Rechner. Für die bcip2 wird noch das aktuelle Verzeichnis beim Start eines Tasks auf das wd-Verzeichnis gewechselt.
Die Informationen zu jedem Rechner speichert PVM in der „hosttable“ und verwendet sie für die Konfiguration.

4.3.1.3 PVM-Logfiles

Meldungen und nähere Details zu Fehlern werden von den Dämonen in besondere Dateien geschrieben: den Logfiles. Sie befinden sich im /tmp-Verzeichnis und sind mit „pvml.uid.hostname“ bezeichnet. Hierbei steht „uid“ für die Unix-UserID und „hostname“ für den jeweiligen Rechner. Da der HP*-Workstation-Cluster auf ein gemeinsames Dateisystem zugreift, ist es wichtig, die Dateien der einzelnen Dämonen namentlich zu trennen. Bei der PVM-Standardinstallation wird von lokalen Dateisystemen der einzelnen Rechner ausgegangen und der hier unnötige Anhang „.machine“ weggelassen. Deshalb muß bei der Installation auf dem Cluster die „SHAREDTMP“-Option verwendet werden.
Im /tmp Verzeichnis existieren auch noch „pvmd.uid.hostname“ Dateien, in denen die Sokketadresse gespeichert ist, über die PVM-Tasks auf den lokalen Dämon zugreifen.

4.4 Programmierung

In diesem Kapitel wird ein Einblick in die Programmierung mit Hilfe der PVM-Bibliothek gegeben. Hierzu wird der Aufbau PVM-Master-Slave-Applikation dargestellt, sowie wichtige PVM-Befehle erläutert. Anschließend werden diese Komponenten anhand eines Beispiels verdeutlicht.

4.4.1 Aufbau einer Master-Slave-Applikation

Beim Master-Slave Konzept (siehe 2.3.4.3) übernimmt der Master alle Steuerungs- und die Slaves untergeordnete Berechnungsaufgaben. Dies ist vergleichbar mit Haupt- und Unterprogrammen in der sequentiellen, strukturierten Programmierung. Es gibt prinzipiell zwei Möglichkeiten der Umsetzung:

- Es existieren getrennte ausführbare Programme für den Master und den Slave.
- Das Master- und das Slave-Programm ist in einem ausführbaren Programm vereint.

Für den Fall, daß ein explizites Master-Programm existiert, wird dieses vom Anwender nach der PVM-Initialisierung gestartet. Das Master-Programm startet selbständig die Slaveprogramme und übermittelt über Nachrichten die Berechnungsdaten an den Slave. Die Ergebnisse sendet der Slave an den Master zurück.
Falls Master und Slave in einem Programm-Modul zusammengefaßt sind, muß der TID des Elterprozesses ermittelt werden. Falls kein Elter existiert, läuft das Programm im Master-Modus, ansonsten im Slave-Modus. Dies wird im Programm abgefragt und entsprechend verzweigt. Im Master-Modus startet das Programm sich selbst auf anderen Rechnern, wobei es dann im Slave-Modus abgearbeitet wird. Das Vorgehen kann mit dem „fork-join“-Konstrukt in UNIX oder mit SPMD-Programmen verglichen werden. Falls im Master- und Slaveprogramm gemeinsamer Code verwendet wird, ergeben sich Vereinfachungen in der Programmierung. Ansonsten muß beim Starten des Slaves unnötiger Code mitgeladen werden und die Trennung der Module ist vorteilhafter.

* Hewlett Packard, Palo Alto, California, USA (http://www.hp.com/)

4.4.2 Programmbibliotheksfunktionen

Im folgenden wird eine knappe Übersicht über die wichtigsten PVM-Funktionen gegeben werden. Eine ausführlichere Dokumentation ist in [10] zu finden. Bei korrekter Einbindung können Hilfestellungen in den „man-pages“ nachgelesen werden. Die folgenden Beispiele sind in C verfaßt. Entsprechende Fortran-Funktionen sind meist gleich aufgebaut und enthalten ein „f“ statt „_“ im Funktionsnamen.

4.4.2.1 Allgemeine Funktionen

```
int tid=pvm_mytid()
```

PVM-Programme können mit dieser Funktion ihre Identität ermitteln. Der erste PVM-Befehl initialisiert zusätzlich den aktuellen Task.

```
int tid=pvm_parent()
```

Die Identität des Elterprozesses wird festgestellt. Falls „tid=PvmNoParent“ existiert kein Elter und der Prozeß ist somit der Master-Prozeß.

```
int info=pvm_exit()
```

„pvm_exit“ meldet den Task beim virtuellen Rechner ab und stellt somit den letzten PVM-Befehl in einem Programm dar.

```
int numt=pvm_spawn(char *task, char **argv, int flag, char
    *where, int ntask, int *tids)
```

Um „ntask“ Slave-Tasks auf anderen Rechnern zu starten wird der Befehl „pvm_spawn“ verwendet. Mit „flag“ wird festgelegt, ob der Resource-Manager (siehe 4.2.3) bestimmt, wo die Tasks gestartet werden oder man in Verbindung mit „where“ die Rechner oder auch die Rechnerarchitektur vorgeben will.

```
int oldval=pvm_setopt(int what, int val)
```

Diese allgemein gehaltene Funktion setzt verschiedene Optionen. Herauszugreifen ist die Option „what=PvmRoute“, mit der durch das Setzen von „val=PvmRouteDirect“ direkte TCP-Verbindungen bei der Kommunikation verwendet werden.

4.4.2.2 Kommunikation

```
int bufid=pvm_initsend(int encoding)
```

Vor dem Senden der Daten werden diese in einem Puffer zwischengespeichert (siehe 4.2.1.3). Dieser Speicher wird mit „pvm_initsend“ bereitgestellt. Mit „encoding“ wird festgelegt, ob die XDR-Konvertierung durchgeführt werden soll (siehe4.2.1.4).

```
int info=pvm_pkint(int *np, int nitem, int stride)
int info=pvm_pkdouble(double *dp, int nitem, int stride)
```

Die Sendedaten „np“ bzw. „dp“ müssen in den Puffer kopiert werden. Hierfür existieren Funktionen für verschiedene Datentypen, von denen die Integer und Double-Varianten exemplarisch aufgelistet sind. „nitem“ gibt die Anzahl an und entspricht bei einem Feld dem Produkt der Dimensionen. „stride“ stellt eine Schrittweite dar und wird üblicherweise auf 1 gesetzt.

```
int info=pvm_send(int tid, int msgtag)
```

Die eigentliche Senderoutine „pvm_send“ schickt die Daten im Puffer zum Prozeß „tid“. Jeder Nachricht kann eine Identifikationsnummer „msgtag“ mitgegeben werden. Die entsprechende Empfangsfunktion kann somit auf diese Nachricht speziell zugreifen. Um eine klare Zuteilung dieser Identifikationsnummern zu bestimmten Nachrichten zu erreichen, wurden mit der „#define“-Anweisung in einem Headerfile NachrichtenIDs definiert und in den Master- und Slaveteil eingebunden. „M2Sxxx“ symbolisiert Nachrichten vom Master an einen Slave und „S2Mxxx“ vom Slave zum Master.

```
int info=pvm_psend(int tid, int msgtag, void *vp, int cnt,
    int type)
```

„pvm_psend“ stellt eine optimierte Variante des Sendens von Variablenfeldern dar. Hierbei ist das Initialisieren des Puffers, das Packen der Daten und die Senderoutine in einer Funktion zusammengefaßt.

```
int info=pvm_mcast(int *tids, int ntask, int msgtag)
```

Ein Broadcast ermöglicht das Senden der Daten an mehrere Prozesse gleichzeitig (siehe 4.2.1.1).

```
int bufid=pvm_recv(int tid, int msgtag)
int bufid=pvm_nrecv(int tid, int msgtag)
int bufid=pvm_trecv(int tid, int msgtag, struct timeval
    *tmout)
```

Diese Funktionen stellen die Empfangsfunktionen nach Kapitel 4.2.1.5 zur Verfügung. „pvm_recv“ ermöglicht blockierendes Empfangen, „pvm_nrecv“ nichtblockierendes Empfangen und „pvm_trecv“ das blockierende Empfangen mit Zeitbeschränkung. „msgtag“ de-

finiert die bei den Sendefunktionen erläuterte Nachrichten-Identifikationsnummer dar. Für „msgtag=-1“ wird jede empfangene Nachricht akzeptiert.

```
int info=pvm_upkint(int *np, int nitem, int stride)
int info=pvm_upkdouble(double *dp, int nitem, int stride)
```

So wie die Daten mit „pvm_pkxxx“ in den Sendepuffer gepackt wurden, so werden sie mit den entsprechenden „pvm_upkxxx“-Funktionen vom Empfangspuffer in die Variablen kopiert.

```
int bufid=pvm_precv(int tid, int msgtag, void *vp, int cnt,
    int type, int *rtid, int *rtag, int *rcnt)
```

Entsprechend „pvm_psend“ wird für „pvm_precv“ eine optimierte Empfangsfunktion für Felder bereitgestellt, bei der Empfangen und Entpacken zusammengefaßt werden.

4.4.3 Beispiel

Im Beispiel „test1“ wird ein Vektor an Slaveprozesse übertrage. Der Slave negiert die Werte des Vektors und schickt ihn an den Master zurück. Um auf einfache Weise festzustellen, von welchem Slaveprozeß gerade empfangen wird, versendet der Master zudem eine Identifikationsnummer an den Slave, die dieser zusätzlich zum Vektor zurücksendet.
In der Datei „test1_glbdef.h“ werden die Nachrichtenidentifikationsnummern für Master

```
/* test1_glbdef.h */
/* Martin Bernreuther, July 1996 */

#define VECDIM 100

/* PVM definitions */
#define SLAVEPRGNAME "test1_slave"
#define NPROC 10

#define M2S_Vec 0
#define S2M_Vecback 1
```

Listing 1: test1_glbdef.h

und Slave definiert. Diese zentrale Verwaltung soll eine eindeutige Definition gewährleisten.

„test1.c“ stellt das Masterprogramm des Beispiel dar und wird zu „test1“ compiliert:

```
/* test1.c */
/* Martin Bernreuther, July 1996 */
/* Send&Receive Vector */

#include <stdio.h>          /* using printf    */
#include "pvm3.h"           /* using PVM-calls */
#include "test1_glbdef.h"   /* MessageID definition */

/* MAIN
     ======================================================================*/
int main(void)
{
  double Vec[VECDIM];                     /* Vector to send&receive */
  int    i,iproc;                         /* loop variables */
  int    mytid,ntask,slave_tid,info,bufid; /* PVM variables */

  mytid=pvm_mytid(); /* determine own task identifier, start PVM */

  for (i=0;i<VECDIM;i++) Vec[i]=(double)i; /* Initialize vector */

/* Create slave processes and send vector ------------------------------------
     */
  for (iproc=1;iproc<=NPROC;iproc++)
  {
    /* start slave task */
    ntask = pvm_spawn(SLAVEPRGNAME,(char**)0,PvmTaskDefault,"",1,&slave_tid);
    /* initialize&pack sendbuffer */
    info=pvm_initsend(PvmDataDefault);
     info=pvm_pkdouble(Vec,VECDIM,1);
     info=pvm_pkint(&iproc,1,1);
    /* send message M2S_Vec to slave_tid */
    info=pvm_send(slave_tid,M2S_Vec);
    (void) printf ("Vector %d sent to t%d\n",iproc,slave_tid);
  }

/* Receive vector */
  for (iproc=1;iproc<=NPROC;iproc++)
  {
    bufid=pvm_recv(-1,S2M_Vecback);   /* blocking receive, accept any sender */
     info=pvm_upkdouble(Vec,VECDIM,1);   /* unpack Vector */
     info=pvm_upkint(&i,1,1);
    (void) printf ("Vector %d received (%d more)\n",i,NPROC-iproc);
  }

/* Exit PVM & stop program */
  pvm_exit();
  return 0;
}
```

Listing 2: test1.c

„test1_slave.c“ codiert den Slaveprozeß des Beispiels in einem seperaten Programm, das unter „test1_slave“ als ausführbares Programm vorliegt:

```
/* test1_slave.c */
/* Martin Bernreuther, July 1996 */

#include "pvm3.h"             /* using PVM functions  */
#include "test1_glbdef.h"     /* MessageID definition */

/* MAIN ============================================================*/
int main(void)
{
  double Vec[VECDIM];    /* Vector to send&receive */
  int vecid;             /* vector number */
  int i;                 /* loop variable */
  int ptid,bufid,info;   /* PVM data */

  ptid=pvm_parent();     /* determine parent task identifier (master) */

  /* receive vector data from master */
  bufid=pvm_recv(ptid,M2S_Vec);
   info=pvm_upkdouble(Vec,VECDIM,1);
   info=pvm_upkint(&vecid,1,1);

  /* do something */
  for (i=0;i<VECDIM;i++) Vec[i]=-Vec[i];

  /* Pack&send result to master */
  pvm_initsend(PvmDataDefault);
   pvm_pkdouble(Vec,VECDIM,1);
   pvm_pkint(&vecid,1,1);
  pvm_send(ptid,S2M_Vecback);

  /* exit PVM */
  pvm_exit();
  return 0;
}
```

Listing 3: test1_slave.c

Die Kommunikation zwischen Master und Slave kann mit diesem Beispiel nachvollzogen werden. Auf eine fehlertolerante Programmierung wurde verzichtet. Die kombinierte Master-Slave-Version ist unter „test2.c“ auf Seite 42 ausgedruckt. Wichtig ist die Ermittlung der Elterprozeßnummer „ptid“, mit der festgestellt wird, ob das Programm im Master oder Slave-Modus läuft. Ein weiteres Demonstrationsbeispiel einer parallelen Integration findet sich im Anhang.

```
/* test2.c */
/* Martin Bernreuther, July 1996 */
/* Send&Receive Vector */

/* DEFINITIONS
     ================================================================*/
#define VECDIM 100

/* PVM definitions */
#define SLAVEPRGNAME "test2"
#define NPROC 10

#define M2S_Vec 0
#define S2M_Vecback 1

#include <stdio.h>          /* using printf    */
#include "pvm3.h"           /* using PVM-calls */

/* MAIN
     ================================================================*/
int main(void)
{
  double Vec[VECDIM];                          /* Vector to send&receive */
  int    vecid;                                /* vector number */
  int    i,iproc;                              /* loop variables */
  int    mytid,ptid,ntask,slave_tid,info,bufid; /* PVM variables */

  mytid=pvm_mytid(); /* determine own task identifier, start PVM */
  ptid=pvm_parent();     /* determine parent task identifier (master) */

  if (ptid==PvmNoParent)
  {
/*  MASTER
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
    for (i=0;i<VECDIM;i++) Vec[i]=(double)i; /* Initialize vector */

    /*  Create slave processes and send vector ------------------------------
     */
    for (iproc=1;iproc<=NPROC;iproc++)
    {
      /* start slave task */
      ntask = pvm_spawn(SLAVEPRGNAME,(char**)0,PvmTaskDefault,"",1,&slave_tid);
      /* initialize&pack sendbuffer */
      info=pvm_initsend(PvmDataDefault);
       info=pvm_pkdouble(Vec,VECDIM,1);
       info=pvm_pkint(&iproc,1,1);
      /* send message M2S_Vec to slave_tid */
      info=pvm_send(slave_tid,M2S_Vec);
      (void) printf ("Vector %d sent to t%d\n",iproc,slave_tid);
    }
    /* Receive vector */
    for (iproc=1;iproc<=NPROC;iproc++)
    {
      bufid=pvm_recv(-1,S2M_Vecback);  /* blocking receive, accept any sender
     */
       info=pvm_upkdouble(Vec,VECDIM,1);   /* unpack Vector */
       info=pvm_upkint(&i,1,1);
      (void) printf ("Vector %d received (%d more)\n",i,NPROC-iproc);
    }
  }
  else
  {
/* SLAVE
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
    /* receive vector data from master */
    bufid=pvm_recv(ptid,M2S_Vec);
     info=pvm_upkdouble(Vec,VECDIM,1);
     info=pvm_upkint(&vecid,1,1);

    /* do something */
    for (i=0;i<VECDIM;i++) Vec[i]=-Vec[i];

    /* Pack&send result to master */
    pvm_initsend(PvmDataDefault);
     pvm_pkdouble(Vec,VECDIM,1);
     pvm_pkint(&vecid,1,1);
    pvm_send(ptid,S2M_Vecback);
  }

/* Exit PVM & stop program */
  pvm_exit();
  return 0;
}
```

Listing 4.1: test2

4.5 Leistungsuntersuchung des Netzwerks

4.5.1 Hintergrund

Zur Untersuchung der Leistungsfähigkeit des Ethernet-Netzwerks, auf das der HP-Workstationpool des Instituts aufbaut, dient folgender Test. Ein Vektor mit doppelt genauen Werten wird vom Master- zum Slaveprozeß geschickt, der ihn sofort zurücksendet (vgl. 4.4.3). Da der Abstand zwischen zwei PEs bei Bussystemen immer zwei beträgt, macht es keinen Unterschied, welche Knoten verwendet werden. Das in C programmierte PVM-Programm variiert die Vektorgröße ebenso wie die Übertragungsart. PVM verwendet grundsätzlich vier Übertragungsarten:

1. „PvmDataDefault“: Daten werden in Sendepuffer kopiert, in das XDR-Format konvertiert, verschickt, rückkonvertiert und vom Empfangspuffer in die Variable kopiert.
2. „PvmDataRaw“: Im Gegensatz zu „PvmDataDefault“ wird auf eine XDR-Konvertierung verzichtet.
3. „PvmDataInplace“: Im Gegensatz zu „PvmDataRaw“ verschickt PVM direkt die Variablenwerte, ohne sie in den Sendepuffer zu kopieren. Im Puffer wird nur die Größe und eine Referenz der Variablen gespeichert, auf die beim Sendevorgang direkt zugegriffen wird.
4. „psend/precv“: Routinen, die für das Versenden von Feldern optimiert wurden. Das Senden bzw. Empfangen wird von einer Funktion erledigt und baut auf dem „PvmDataInplace“-Konzept auf.

Außerdem kann statt der UDP-Verbindung zwischen Knoten mit der Option „PvmRouteDirect“ eine verbindungsorientierte TCP-Kommunikation gewählt werden. Die Skalierbarkeit ist aufgrund der limitierten Sockets zwar eingeschränkt, für den Workstationcluster aber ausreichend.

Dies führt zu acht Kommunikationsarten, die verglichen werden. In einem Multiuser-Workstationcluster streuen Ergebnisse je nach momentaner Auslastung der Maschinen und des Netzwerks. Eine besondere Belastung für das Netzwerk stellt das verteilte Dateiensystem (NFS) dar. Aufgrund der Schwankungen wurden die Vektoren jeweils 1000x verschickt und empfangen. Nach 100 Durchläufen wurde die Zeit gemessen und die 10 somit ermittelten Werte gemittelt und auf eine Sende-/Empfangsoperation umgerechnet.

4.5.2 Diskussion

Die Ergebnisse sind in Abbildung 17 dargestellt:

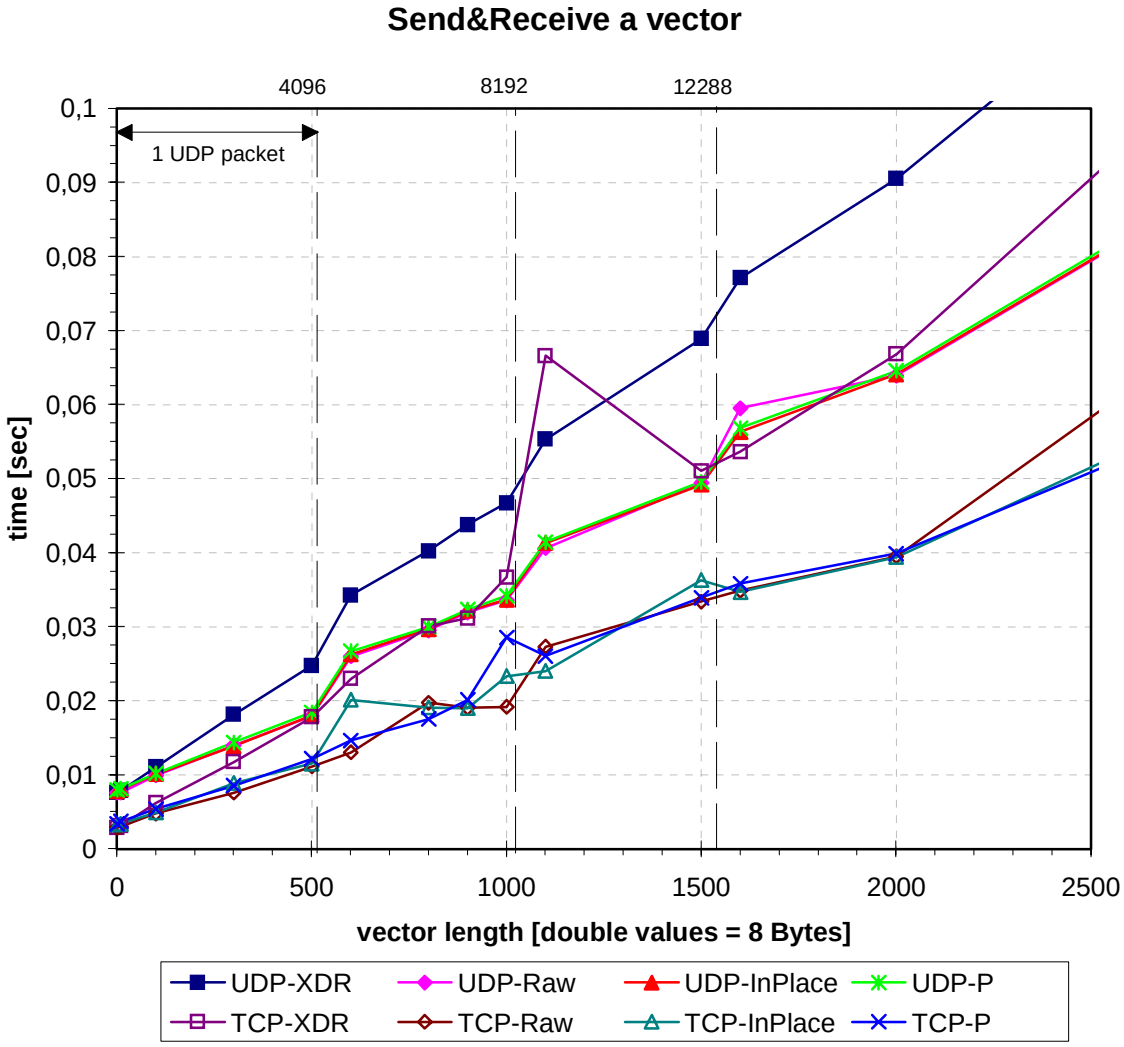


Abbildung 17: Senden/Empfangen von Vektoren

Es ist zu beachten, daß PVM bereits aufgebaute TCP-Verbindungen nicht nach jedem Sendevorgang automatisch beendet. Bei wiederholtem Senden wird dadurch die Latency verfälscht und eine kürzere Zeit ermittelt. Beim Messen eines einzigen Sende- und Empfangs-Vorgangs ist das Ergebnis bei kleinen Vektoren aufgrund der sehr kurzen Zeit allerdings sehr ungenau. Das Experiment sollte aber weniger der Ermittlung exakter Zahlen als vielmehr der Tendenzen und Schlußfolgerungen dienen. Zudem wird üblicherweise in Anwendungen auch mehrmals kommuniziert.

Bei UDP-Verbindungen werden Datagramme verschickt. Die Größe dieser Pakete ist bei der HP-Architektur auf 4096 Bytes limitiert. Da die Werte doppelter Genauigkeit mit 8 Bytes dargestellt werden, entspricht das 512 Werten. Hier ergibt sich ein Sprung, da bei weiteren Daten ein zusätzliches Paket verschickt werden muß. Zwischen den Sprüngen er-

gibt sich ein linearer Verlauf. Der größte Faktor stellt die XDR-Konvertierung dar. Die Steigung der Kurven im Vergleich zum unkonvertierten Versenden wird deutlich steiler, was sich vor allem bei größeren Vektoren auswirkt. Die weiteren Optimierungen fallen dagegen nicht ins Gewicht.
Auch bei TCP-Verbindungen bedeutet der Verzicht auf die XDR-Konvertierung die größte Geschwindigkeitssteigerungen. Die für UDP vorhandenen Grenzen aufgrund der Paketlänge sind nicht vorhanden. Deshalb sind lineare Kurven zu erwarten. Grundsätzlich ist zu beobachten, daß im Test Daten mit TCP- schneller als mit UDP-Verbindungen übertragen werden. Dieses Ergebnis ist aber nur für eine Vielzahl an Kommunikationsvorgängen aussagekräftig (s. o.).
Diese Ergebnisse können im Allgemeinbetrieb sehr stark schwanken. Einzelne Ausreißer können so weit vom Mittelwert abweichen, daß dieser stark verändert wird. So wurde in einem Fall bei einer erwarteten Übertragungszeit von 0.04 sec ein Wert von 2 sec gemessen! So läßt sich der Ausreißer der „TCP-XDR“-Kurve für den 1100-Werte Vektor erklären.
Für andere Computerarchitekturen kann sich das Bild ändern. Bei MPP (Massively Parallel Processing) -Systemen stellen Computerhersteller auf den Rechner optimierte Kommunikationsroutinen zur Verfügung, die im PVM „psend/precv“ -Befehl eingebunden sind. Somit kann auf diesen Systemen mit diesen Befehlen nach Beguelin et al. [13] eine größere Geschwindigkeitssteigerung erreicht werden.
Zusammenfassend kann gesagt werden, daß auf diesem System eine Datenübertragung ohne Konvertierung mit einer TCP-Verbindung die besten Ergebnisse zeigt. Allerdings kann keine Aussage über Broadcast-Nachrichten gemacht werden, bei denen für UDP Vorteile zu erwarten sind.

5 Lineare Algebra auf MIMD-Systemen

Die Lösung von linearen Gleichungssystemen, Ausgleichsproblemen und Eigenwertproblemen gehört in die Gruppe der Probleme, die in der linearen Algebra auftauchen. Eine besondere Rolle spielen die linearen Gleichungssysteme, die im Bauingenieurwesen z.B. beim Verschiebungsgrößenverfahren vorkommen. Für die Grundgleichung

$$A \cdot x = b \tag{19}$$

kann für A die globale Steifigkeitsmatrix und für b der Lastvektor eingesetzt werden, um für x die gesuchten Verschiebungen zu ermitteln. Der „Rechte-Seite-Vektor“ b wird für mehrere Lastfälle zu einer Matrix erweitert. Verschiedene Strukturmerkmale der globalen Steifigkeitsmatrix können bei der Lösung der Gleichung ausgenutzt werden. Bei den häufig vorkommenden Stabwerken nach Theorie 1. Ordnung ist diese Matrix symmetrisch und positiv definit. Spezialisierte Lösungsalgorithmen nutzen diese Eigenschaften aus, um gegenüber der Lösung des allgemeinen Problems schneller oder genauer zu sein. Auch können z.B. bei symmetrischen oder bandartigen Strukturen durch ein spezielles Speicherformat Einsparungen an Speicherressourcen erzielt werden. Durch parallele Ansätze soll es möglich gemacht werden, auch sehr große, komplexe Probleme in annehmbarer Zeit zu lösen.

5.1 ScaLAPACK

5.1.1 Übersicht

ScaLAPACK (Scalable LAPACK) ist eine Programmbibliothek für Routinen der linearen Algebra. Sie überträgt die weit verbreitete LAPACK-Bibliothek auf lose gekoppelte MIMD Computer. Diese Bibliotheken sind wie PVM über „netlib“ frei verfügbar.
Die sequentielle LAPACK-Bibliothek wurde für Speicherzugriffe im Hinblick auf die interne Speicherstruktur optimiert, im besonderen auf die Ausnutzung der schnellen Prozessorregister und des Memory-Cache. Bei eng gekoppelten Parallelrechnern mit gemeinsamen Speicher sind die Daten so auf verschiedene Speicherbänke zu verteilen, daß möglichst wenig Zugriffskonflikte auftreten. Für ScaLAPACK, das auf lose gekoppelte Systeme mit verteiltem Speicher aufbaut, ist hingegen die Kommunikationsminimierung ausschlaggebend. Eine Parallelisierung der LAPACK-Routinen durch Versenden der notwendigen Daten an die PEs wird für eine größere Anzahl Knoten ineffizient, da der Kommunikationsaufwand rapide ansteigt. Dieser Ansatz ist somit nicht skalierbar. Deshalb werden in ScaLAPACK die Matrizen auf die einzelnen Knoten verteilt.

5.1.2 Aufbau

ScaLAPACK baut auf mehreren Paketen auf, die Teilfunktionalitäten zur Verfügung stellen. Auf die einzelnen Programmpakete wird im folgenden eingegangen. Die gegenseitigen Abhängigkeiten zeigt Abbildung 18:

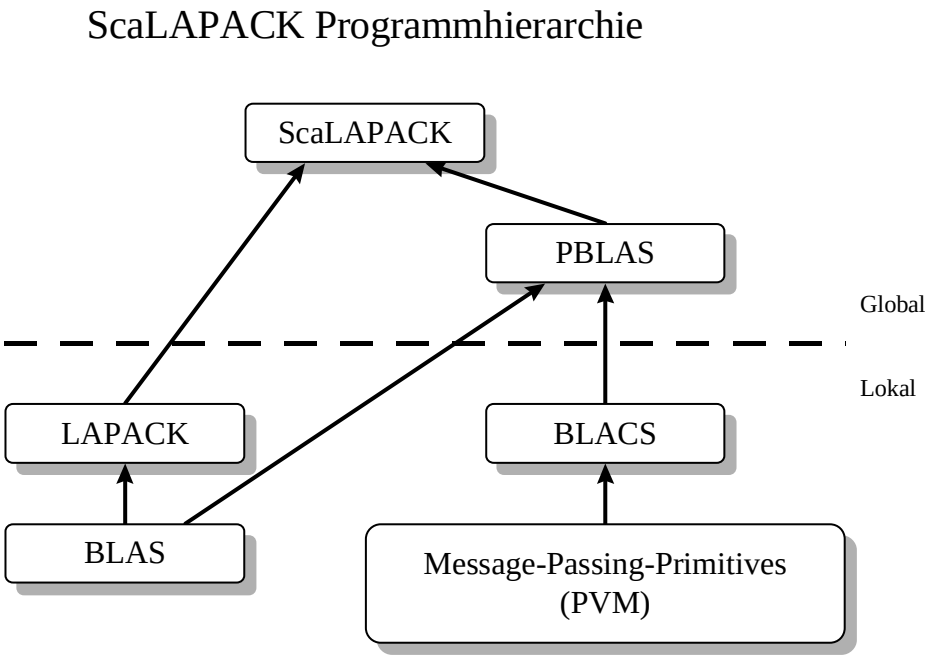


Abbildung 18: ScaLAPACK Struktur

5.1.2.1 BLAS

Die „Basic Linear Algebra Subprograms“ stellen Basisoperationen für Vektoren und Matrizen bereit. Durch eine Standardisierung der Funktionsaufrufe wird die Portabilität der aufbauenden Applikationen gewährleistet. Auf zahlreichen Platformen existieren maschinenoptimierte Implementierungen mit denen die Ausführungsgeschwindigkeit dieser, in der linearen Algebra oft verwendeten Grundfunktionen, deutlich gesteigert wird. BLAS wird in drei verschiedene Stufen (Levels) eingeteilt:

- Level 1: Operationen mit einer inneren Schleife, wie z.B. Vektor-Vektor-Operationen.
- Level 2: Operationen mit zwei inneren Schleifen, wie z.B. Matrix-Vektor-Operationen.
- Level 3: Operationen mit drei inneren Schleifen, wie z.B. Matrix-Matrix-Operationen.

5.1.2.2 LAPACK

Das „Linear Algebra PACKage“ wurde bereits in der Übersicht angesprochen und stellt Routinen zur Lösung linearer Gleichungssysteme, Ausgleichsrechnungen und Eigenwertprobleme für SISD-Systeme bereit. Dabei existieren auch spezialisierte Versionen für besondere Matrixtypen. LAPACK verwendet BLAS-Routinen und kann mit Hilfe von Performance-Parametern an den jeweiligen Rechner speziell angepaßt werden.

5.1.2.3 BLACS

Die „Basic Linear Algebra Communication Subprograms“ stellen eine Message-Passing Bibliothek speziell für die Lineare Algebra dar. Dabei wird auf ein Message-Passing System, wie es PVM enthält, aufgebaut. Prozesse werden auf einer zweidimensionalen Gitterstruktur abgebildet. Hiermit kann eine physikalische Struktur wie die in Kapitel 2.2.1.3 dargestellte Gitterstruktur abgebildet werden. Jeder Prozeß speichert folglich Teilmatrizen und -vektoren. Das Versenden dieser Teilmatrizen zwischen Prozessen oder die Berechnung globaler Reduktionsoperationen (Summe, Maxima, Minima) sind die Aufgaben von BLACS. Die Topologie der Verbindungen zwischen PEs bzw. Prozessen bestimmt die Übertragungsgeschwindigkeiten und muß bei der Verteilung der Matrizen beachtet werden. Für das Bussystem mit konstanten Abständen läßt sich die Verbindungstopologie nicht auf das zweidimensionale Raster übertragen, das auf einer Gitterstruktur aufbaut.

5.1.2.4 PBLAS

„Parallel BLAS“ stellt die parallele Erweiterung von BLAS aufbauend auf BLACS dar. Die in 5.1.2.1 beschriebenen Lineare Algebra Basisfunktionen werden mit PBLAS für verteilte Systeme zur Verfügung gestellt. Der Aufbau entspricht der sequentiellen Version, so daß sich die Funktionsaufrufe stark gleichen.

5.1.3 Verteilen einer Matrix

Die Parallelisierung von ScaLAPACK beruht auf einer zweidimensionalen, statisch blockzyklischen Verteilung der Matrizen auf eine Gitterstruktur. Dies wird anhand einer 9x9 Matrix demonstriert. Zunächst teilt man die Matrix in Blöcke auf. In Tabelle 9 ist die Partitionierung in 2x2 Blöcke dargestellt. Die Blockgröße ist frei wählbar. Bei zunehmender Blockgröße muß tendenziell weniger kommuniziert werden, aber die Lastverteilung wird dabei ungleichmäßiger. Die optimale Blockgröße muß damit für jede Konfiguration spezifisch bestimmt werden. Diese Blöcke werden nun auf die einzelnen Prozessoren blockzyklisch verteilt. Die Verteilung auf ein 2x3 Prozeßraster stellt Tabelle 10 dar.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}	a_{49}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}	a_{59}
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{68}	a_{69}
a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}	a_{79}
a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}	a_{89}
a_{91}	a_{92}	a_{93}	a_{94}	a_{95}	a_{96}	a_{97}	a_{98}	a_{99}

Tabelle 9: 9x9 Matrix in 2x2 Blöcke partitioniert

	0				1			2	
0	a_{11}	a_{12}	a_{17}	a_{18}	a_{13}	a_{14}	a_{19}	a_{15}	a_{16}
	a_{21}	a_{22}	a_{27}	a_{28}	a_{23}	a_{24}	a_{29}	a_{25}	a_{26}
	a_{51}	a_{52}	a_{57}	a_{58}	a_{53}	a_{54}	a_{59}	a_{55}	a_{56}
	a_{61}	a_{62}	a_{67}	a_{68}	a_{63}	a_{64}	a_{69}	a_{65}	a_{66}
	a_{91}	a_{92}	a_{97}	a_{98}	a_{93}	a_{94}	a_{99}	a_{95}	a_{96}
1	a_{31}	a_{32}	a_{37}	a_{38}	a_{33}	a_{34}	a_{39}	a_{35}	a_{36}
	a_{41}	a_{42}	a_{47}	a_{48}	a_{43}	a_{44}	a_{49}	a_{45}	a_{46}
	a_{71}	a_{72}	a_{77}	a_{78}	a_{73}	a_{74}	a_{79}	a_{75}	a_{76}
	a_{81}	a_{82}	a_{87}	a_{88}	a_{83}	a_{84}	a_{89}	a_{85}	a_{86}

Tabelle 10: Verteilung der Matrix ausTabelle 9 auf 2x3 Prozesse

Für das Bussystem des Workstationclusters wurde ein eindimensionales Prozeßraster gewählt. Die blockzyklische Aufteilung der Matrix wird von einer selbst implementierten Funktion übernommen. Hierbei muß beachtet werden, daß das ScaLAPACK-Paket in Fortran programmiert ist und Fortran im Gegensatz zu C eine spaltenweise Abspeicherung der Matrizen vornimmt. Trotz zahlreicher Schwierigkeiten, auf die nicht näher eingegangen werden soll, war es möglich diese Fortran-Bibliothek in ein C-Programm einzubinden. Die Verteilung der Matrizen wird in einem SPMD-Programm mit Hilfe von BLACS-Routinen bewerkstelligt.

5.1.4 Einteilung der Funktionen

Grundsätzlich sind ScaLAPACK-Funktionen in drei Gruppen eingeteilt:

- „Treiberfunktionen“ stellen eine komplette Lösung eines Problems dar.
- „Rechenfunktionen“ bearbeiten spezielle Rechenoperationen
- „Zusatzfunktionen“ wie Erweiterungen zu PBLAS.

Falls geeignete Treiberfunktionen zur Verfügung stehen, ist es am einfachsten, diese zu verwenden. Sie stehen in spezialisierter Form für unterschiedliche Aufgaben zur Verfügung.

5.1.4.1 Namenskonvention für ScaLAPACK-Funktionen

Die Namen der Treiber- und Rechenfunktionen sind nach dem Schema Pxyyzzz aufgebaut:

- Der 1. Buchstabe P steht für „parallel“ und soll Namenskonflikte mit der sequentiellen LAPACK-Bibliothek ausschließen.
- Die Routinen sind für unterschiedliche elementare Datentypen vorhanden. Fortran kennt keine „Templates“ wie C++ und so mußte für jeden Datentyp eine eigene Funktion implementiert werden. Der 2. Buchstabe (x) zeigt im Funktionsname den Datentyp an. Für das Bauingenieurwesen sind vor allem die mit „D“ beginnenden „Double precision“ Routinen von Bedeutung.

- Eine Auswahl für die Einteilung von Matrizen durch ScaLAPACK wird in Tabelle 11 gegeben. Der aus zwei Buchstaben bestehende Code findet sich in den ScaLAPACK-Funktionsnamen an Stelle 3 und 4 (yy) wieder.

Code	Beschreibung
GE	general: allgemeine Matrix (auch unsymmetrisch oder rechteckig)
GB	general banded: Matrix mit Bandstruktur
GT	general tridiagonal: tridiagonale Matrix
OR	orthogonal: orthogonale Matrix
SY	symmetric: symmetrische Matrix
ST	symmetric tridiagonal: tridiagonale symmetrische Matrix
PO	symmetric or Hermitian positive definite: symmetrische, positiv definite Matrix
PB	symmetric or Hermitian positive definite band: symmetrische, positiv definite Matrix mit Bandstruktur
PT	symmetric or Hermitian positive definite tridiagonal: symmetrische, positiv definite, tridiagonale Matrix

Tabelle 11: ScaLAPACK-Matrizenklassifikation

- Die letzten drei Positionen 5 bis 7 (zzz) definieren die Funktion. „SV“ steht z.B. für die „solver“-Treiberfunktionen zum Lösen eines linearen Gleichungssystems.

Die Funktion „PDPBSV“ stellt z.B. die Treiberfunktion zum Lösen eines linearen Gleichungssystems nach Gleichung (19) mit einer symmetrischen, positiv definiten Matrix A, die eine Bandstruktur aufweist, dar.

6 Evolutionsstrategien

6.1 Einleitung

Die Natur diente schon oft als Vorbild für die Lösung technischer Probleme. Ingenieure versuchen hierbei durch Nachahmung, Kopieren und Übertragen neue Ansätze zu finden. Daraus entstand die Bionik als Forschungsgebiet. Am Institut für Baustatik der Universität Stuttgart wurde im Zuge des Sonderforschungsbereichs 230 (Natürliche Konstruktionen) zahlreiche Arbeiten zu diesem Themengebiet verfaßt.
Organismen wurden im Verlaufe der Zeit optimal an ihre Umgebung angepaßt oder starben aus. Doch hat die Natur nicht nur optimierte Organismen hervorgebracht. Auch die Art wie sich die Organismen fortpfanzen und dabei anpassen wurde optimiert. Es fand eine Evolution der Evolution statt. Aus diesem Gesichtspunkt heraus lassen sich auf dem Gebiet der Optimierung in der Natur ebenfalls viele Ansatzpunkte finden. 1859 begründete Darwin mit seinem Werk „On the Origin of Species by Means of Natural Selection“ die Evolutionstheorie, die heute die wichtigste und fundierteste Theorie der modernen Biologie darstellt. Die biologische Evolution ist die Grundlage für die Evolutionsstrategie, die zur Lösung von Optimierungsproblemen im mathematisch-technischen Bereich dient. Rechenberg spricht sich in [15] gegen eine genaue Kopie der Natur in der Technik aus, da die Randbedingungen für die Objekte nicht die gleichen sind. Unterschiedliche Materialien sind ebenso zu beachten wie die Tatsache, daß man oft nur Teilfunktionen des Vorbilds aus der Natur übernehmen will. Die Bionik soll nur an die Lösung heranführen. Die Evolutionsstrategien (ES) wurden durch Rechenberg [15] und Schwefel [16] begründet. Etwa zeitgleich entstand unabhängig hiervon das „Evolutionary Programming“ (EP) und die „Genetic Algorithms“ (GA), die auf denselben Grundlagen aufbauen. Während die EP heute den ES gleichen unterscheiden sich die GA in den Ansätzen. Nach einem kurzen Einblick in die biologische Evolution wird hier auf die Evolutionsstrategien eingegangen.

6.2 Biologische Evolution

Die Evolution wird geprägt durch die Vorstellung, daß sich Individuen mit vorteilhaften Eigenschaften durchsetzen. Nach Darwin wird dies durch eine natürliche Auslese (Selektion) erreicht („survival of the fittest“). Es werden mehr Nachkommen produziert, als überleben. Der wachsenden Population steht ein begrenzter Lebensraum und Nahrung zur Verfügung, was zu einem „Selektionsdruck“ führt.
Der Augustinerabt Gregor Johann Mendel entdeckte um 1865 die nach ihm benannten Vererbungsgesetze: das Uniformitätsgesetz, das Spaltungsgesetz und das Rekombinationsgesetz. Sie beschreiben, wie Merkmale an Nachkommen weitergegeben werden. Bei Kreuzung zweier reinerbiger, sich in einem oder mehreren Merkmalen unterscheidenden Vorfahren der

Ursprungsgeneration (Parentalgeneration), treten nach dem *Uniformitätsgesetz* nur einheitliche Nachkommen in der ersten Nachfolgegeneration (Filialgeneration 1) auf. Bei Betrachtung eines sich unterscheidenden Merkmals xx und yy der Eltern entsteht xy oder yx. Mendel führte auch den Begriff der *dominanten* und *rezessiven Erbmerkmale* ein. Das dominante setzt sich durch. Dadurch ergibt sich ein Verhältnis 3:1 für eine dominante Ausprägung. Die rezessive Ausprägung kommt nur durch ein reinerbiges, rezessives Merkmal zustande. Bei *intermediären* Erbgängen ergibt sich für die Ausprägung ein Mittelwert der einzelnen Merkmale. Das Verhältnis liegt hier bei 1:2:1. Zwei intermediäre Ausprägungen stehen den jeweils reinerbigen Ausprägungen gegenüber. Nach dem *Spaltungsgesetz* tritt nach Kreuzungen der ersten Nachfolgegeneration eine Aufspaltung der Merkmalsausprägung auf. Die Kreuzung von xy bzw. yx -Merkmalen ergibt xx, xy, yx oder yy. Es kann also auch ein Nachkomme mit rezessivem reinerbigen Merkmal vorkommen. Das wichtigste Gesetz für die Evolution ist das Gesetz der freien *Rekombination* des Erbgutes. Es sagt aus, daß das Erbgut in allen möglichen Kombinationen neu zusammengestellt werden kann. Die einzelnen Merkmale sind also völlig unabhängig voneinander.

Die beim Menschen vorkommenden eukaryotischen Zellen enthalten im Gegensatz zu den Prokaryoten (z.B. Bakterien) einen Nukleus (Zellkern). In dieser Steuerzentrale sind im Karyoplasma (Kernplasma) die fadenförmigen *Chromosomen* als Träger der Erbsubstanz enthalten. Die Chromosomen bestehen aus Desoxyribonukleinsäuren (DNS) und Proteinen, wobei die DNS die Erbinformationen linear verschlüsselt enthält. Eine Erbinformationseinheit wird als Gen bezeichnet. Beim Menschen tritt die DNS als Doppelstrang von einigen Millionen Nukleotiden auf. Ein *Nukleotid* bestehen aus einem Phosphatrest, einem Zucker und einer Base. Es existieren nur vier Nukleotidbasen: Adenin (A), Guanin (G), Cytosin (C) und Thymin (T). Die Basen von jeweils drei aufeinanderfolgenden Nukleotiden (Codons) kodieren eine Aminosäure. Somit stellt ein Codon stellt die kleinste Informationseinheit dar. Es sind 4^3=64 verschiedene Codons möglich, während nur 20 verschiedene Aminosäuren existieren, d. h. der genetische Code ist degeneriert. Einige spezielle Codons fungieren als Stopzeichen. Die DNS-Information wird mittels Transkription und Translation durch Ribonukleinsäuren (RNS) in das zu synthetisierende Protein umgesetzt. Beim Menschen ist die Information auf einen Chromosomensatz von 46 Chromosomen verteilt und bei geschätzten 10^9 Codons ergeben sich $20^{1.000.000.000}$ Möglichkeiten.

Die Vermehrung von Zellen und damit das Wachstum des Organismus beruht auf Zellteilung. Das Erbgut wird so auf die Tochterzellen verteilt, daß alle die vollständige Erbinformation erhalten. Mit *Mitose* bezeichnet man die ungeschlechtliche, erbgleiche Zellteilung. Falls keine Störungen auftreten besitzen die Tochterzellen identisches Erbgut. Dieser Vorgang läuft in Minuten ab. Niedere Organismen pflanzen sich durch Mitose fort. Anders verhält es sich mit der *Meiose*, die bei der Fortpflanzung höherer Organismen vorkommt. Die Meiose stellt also eine Optimierung der Fortpflanzung dar. Bei der Meiose werden die Chromosomen durch ein „crossing-over“ miteinander rekombiniert und zufällig auf die Keimzellen verteilt. Hierbei wird das Erbgut gemischt und der doppelte Chromosomensatz auf einen einfachen reduziert (Reduktionsteilung). Es entstehen *haploide* Keimzellen. In der

befruchteten Eizelle ist dann wieder ein doppelter Chromosomensatz vorhanden und die Zelle ist somit *diploid*. Die Meiose benötigt oft Tage, Wochen oder gar Jahre (z.B. Mensch).

6.3 Technisch-mathematische Umsetzung der biologischen Evolution in der ES

Technisch-mathematische Probleme lassen sich oft durch Parameter beschreiben. Diese Parameter können z.B. die Variablen einer Funktion oder die Querschnittswerte eines Fachwerks sein. Diese Parameter stellen den *Genotyp*, also die Gen-Schaltstellungen im Evolutionstheorie dar. Bei experimentellen Arbeiten muß dieser Genotyp in den *Phänotyp*, das zu testende Modell, umgewandelt werden. Für Computersimulationen ist diese Unterscheidung nicht notwendig. Auf den Phänotyp wirkt die Umwelt ein. Eine Last auf ein zu optimierendes Fachwerk würde dieser Einwirkung entsprechen. Auch die Vitalität des Individiums muß bestimmt werden. Dies wird anhand einer *Fitneßfunktion* durchgeführt. Sie spiegelt die Bewertung des Phänotyps wieder. Der Funktionswert einer zu minimierenden Funktion oder das Gesamtgewicht des zu optimierenden Fachwerks sind solche Bewertungskriterien.

Alle Parameter faßt man in einem Vektor zusammen. Das durch die Parameter definierte Individium wird durch die Fitneßfunktion bewertet. Hierbei spielt es keine Rolle, ob diese Fitneß anhand eines experimentellen Versuchs, eines Computerberechnungsmodells oder einer Funktion ermittelt wird. Die Erzeugung neuer Individuen erfolgt durch den Rekombinations- und Mutationsoperator.
Nach Bäck [17] ist das Optimierungsproblem mit

$$\hat{f} := f\left(\hat{\vec{x}}\right) = \min\left\{ f\left(\vec{x}\right) \,\middle|\, \vec{x} \in M \subseteq \Re^n \right\} \tag{20}$$

$$M = \left\{ \vec{x} \in \Re^n \,\middle|\, g_j \geq 0 \;\forall\; j \in \left\{1,\ldots,q\right\} \right\}; \quad g_j : \Re^n \to \Re \tag{21}$$

beschreibbar. Diese Darstellung enspricht der Notation in Kapitel 1.1, wobei die Ungleichheits-Nebenbedingungen g im Gegensatz zu Bletzinger [1] unterschiedlich (negativ) definiert ist. Die Individuen a sind im „Entwurfsraum“ I definiert und beinhalten neben den Parametern x eventuell noch eine Strategieparametermenge S (siehe6.6):

$$\vec{a} \in I; \quad I = \Re^n \times S \tag{22}$$

Eine Population P besteht aus μ Elter- und λ Kindinividuen:

$$P^{(t)} = \left\{ \vec{a}_1,\ldots,\vec{a}_k \right\} \in I^k; \quad k \in \left\{ \mu, \lambda \right\}; \quad \mu, \lambda \in \mathrm{N} \tag{23}$$

6.4 Grundelemente der Evolutionsstrategie

6.4.1 Mutation

Die *Mutation* verursacht zufällige Veränderungen am Individium x. Diese Änderungen werden beim Mutationsoperator durch die Addition eines normalverteilten Mutationsvektors z erzeugt:

$$mut: \quad I \to I \tag{24}$$

$$mut(\vec{x},\vec{\sigma}) = \vec{x} + \vec{z}(\vec{\sigma}) \tag{25}$$

6.4.1.1 Gauß'sche Normalverteilung

Die Gauß'sche Normalverteilung ist durch die Wahrscheinlichkeitsdichtefunktion p mit der Zufallsgröße z, dem Erwartungswert ξ und der Standardabweichung σ definiert, wobei sich die Gesamtwahrscheinlichkeit zu 1 ergeben muß:

$$p(z,\xi,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(z-\xi)^2}{2\sigma^2}\right) \tag{26}$$

$$\int_{-\infty}^{\infty} p(z,\xi,\sigma)\,dz = 1 \tag{27}$$

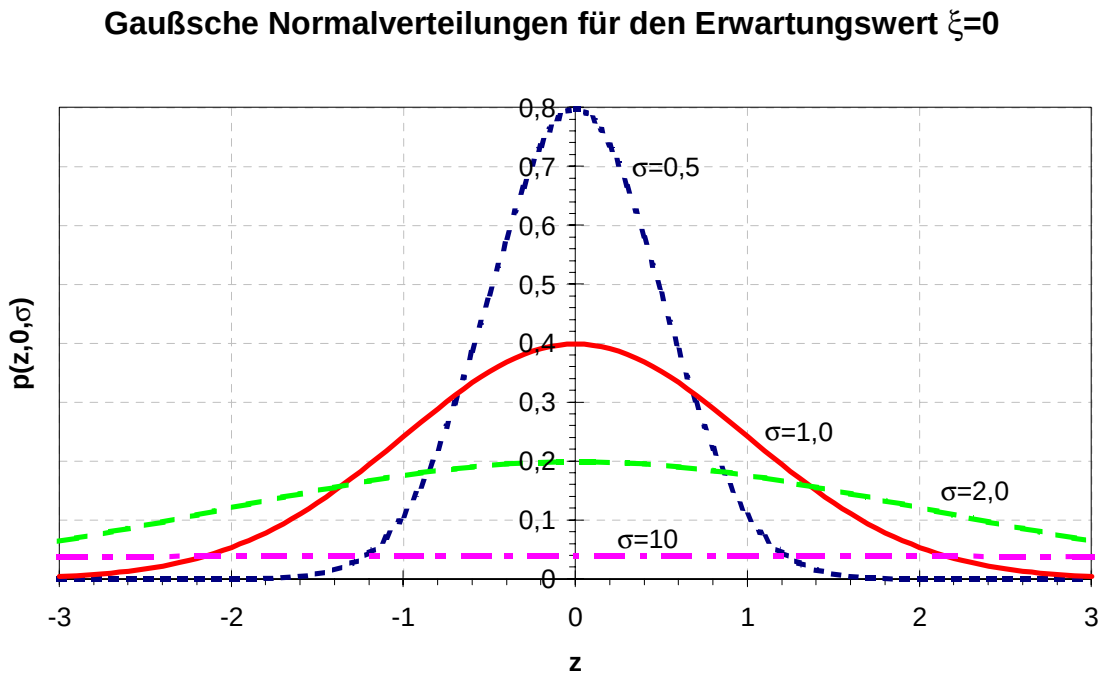


Abbildung 19: Gauß'sche Normalverteilungen

Für die Erzeugung normalverteilter Zufallszahlen Z aus zwei im Intervall (0,1] gleichverteilten Zufallszahlen Y wird von Schwefel [16] auf eine Transformationsregel von Box und Muller verwiesen:

$$Z_1' = \sqrt{-2 \cdot \ln Y_1} \cdot \sin\left(2 \cdot \pi \cdot Y_2\right) \tag{28}$$

$$Z_2' = \sqrt{-2 \cdot \ln Y_1} \cdot \cos\left(2 \cdot \pi \cdot Y_2\right) \tag{29}$$

Die Transformation der N(0,1)-Zufallszahlen, d. h. normalverteilte Zufallszahlen mit dem Erwartungswert 0 und der Standardabweichung 1, auf N(ξ,σ) erfolgt mit:

$$Z_i = \sigma_i \cdot Z_i' + \xi \tag{30}$$

Für diskrete Probleme müssen die Werte noch gerundet werden.

6.4.1.2 Binominalverteilung

Eine weitere Möglichkeit normalverteilte Zufallszahlen speziell für ganzzahlige Werte anzunähern, ist die Umsetzung des Galtonbretts (siehe Abbildung 20) in einen Algorithmus.

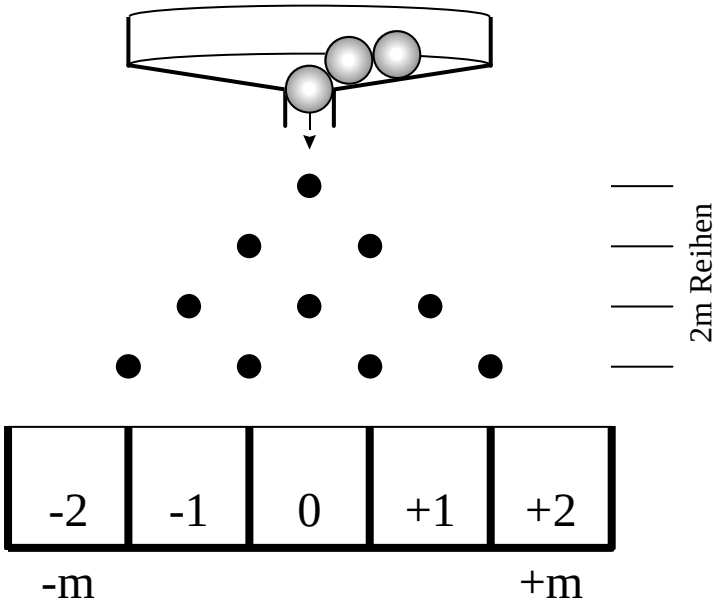


Abbildung 20: Galton-Brett

Ausgehend von gleichverteilten Zufallzahlen wird ein Münzwurf simuliert, wobei den zwei Münzseiten die Zahlenwerte -½ und ½ zugewiesen werden. Beim Aufsummieren von 2m Würfen erhält man ganzzahlige normalverteilte Zufallszahlen im Intervall [-m,m]. Die 2^{2m} Möglichkeiten verteilen sich binominal auf 2m+1 Endresultate. In der folgenden Formel ergibt sich aus der Anzahl der Ziehungen mit ½ das resultierende Ergebnis z:

$$z = k \cdot \tfrac{1}{2} - (2m - k) \cdot \tfrac{1}{2} = k - m; \quad 0 \le k \le 2m \tag{31}$$

Die Binominalverteilung ist definiert durch

$$p_k^{(n)} = P\left(X^{(n)} = k\right) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}; \quad 0 \le k \le 2m \tag{32}$$

Für dieses Problem ist die Anzahl der Ziehungen n mit 2m gegeben, die Anzahl des Eintretens des Ereignisses k bereits definiert, wobei die Wahrscheinlichkeit des Eintretens dieses Ereignisses p = ½ ist. Die möglichen Zufallsgrößen $X^{(n)} = 0,1,\ldots,n$ werden durch Gleichung (31) transformiert, um das Resultat z zu erhalten.
Der *Binominalkoeffizient* ist gegeben durch

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}; \quad 0 \le k \le n \tag{33}$$

Hieraus ergibt sich

$$p_k^{(2m)} = \binom{2m}{k} \cdot 2^{-2 \cdot m} = \frac{(2m)!}{k! \cdot (2m-k)!} \cdot 2^{-2 \cdot m}; \quad 0 \le k \le 2m \tag{34}$$

Die Wahrscheinlichkeitsverteilung für z

$$p(z,m) = p_{z+m}^{(2m)} \tag{35}$$

ist aus Abbildung 21 ersichtlich:

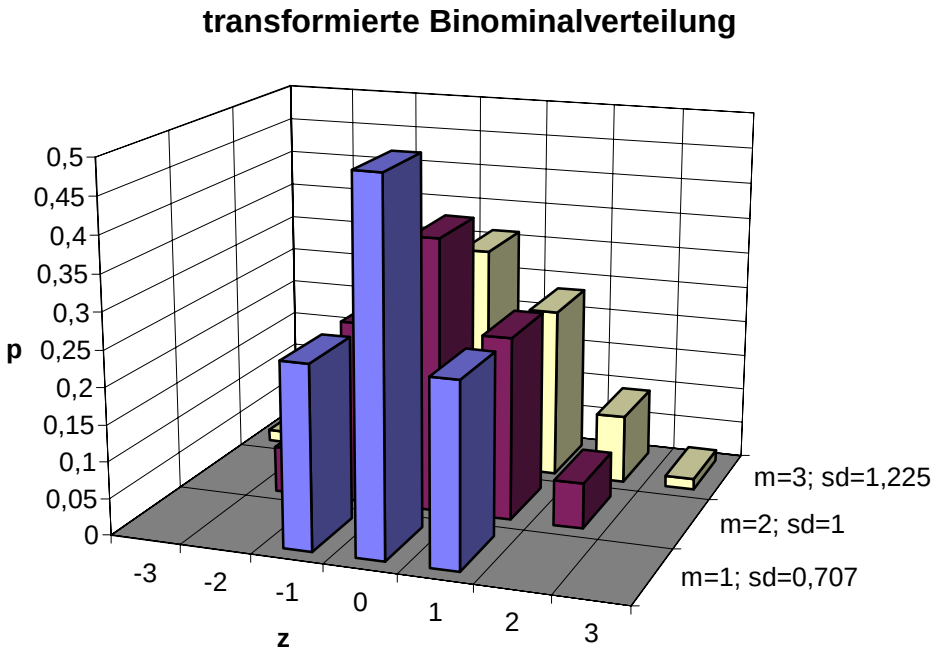


Abbildung 21: Wahrscheinlichkeitsverteilung des Galton-Algorithmus

Mit dem diskreten Erwartungswert einer Binominalverteilung von

$$EX = \sum_{k=0}^{n} k \cdot \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} = n \cdot p \tag{36}$$

läßt sich der Erwartungswert des Resultats z aus Gleichung (31) berechnen:

$$Ez = EX - m = 2m \cdot \tfrac{1}{2} - m = 0 \tag{37}$$

Der Erwartungswert 0 ist notwendig für die Mutation.

Die Varianz der Binominalverteilung wird mit

$$DX = \sum_{k=0}^{n} (k - n \cdot p)^2 \cdot \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} = n \cdot p \cdot (1-p) \tag{38}$$

und die Standardabweichung damit mit

$$\sigma = \sqrt{n \cdot p \cdot (1-p)} \tag{39}$$

angegeben. Hieraus ergibt sich

$$\sigma = \sqrt{2m \cdot \frac{1}{2} \cdot \left(1 - \frac{1}{2}\right)} = \sqrt{\frac{m}{2}} \tag{40}$$

Dieser einfache Algorithmus ist für die Ermittlung binominalverteilten ganzzahligen Zufallszahlen geeignet, die für diskrete Probleme der Evolutionsstrategie verwendbar sind.

Die gleichverteilten Zufallszahlen können nach [14] für Zufallszahlengeneratoren, die auf linearen Kongruenzgeneratoren basieren, dadurch verbessert werden, daß man die erzeugten Zufallszahlen in einem Feld zwischenspeichert, einen zufällig gewählten Wert entnimmt und durch eine neue Zufallszahl ersetzt.

6.4.1.3 Mutationsvarianten

Bei der Mutation wird der Erwartungswert $\xi = 0$ gesetzt. Die größte Wahrscheinlichkeit soll bei keinen oder kleinen Änderungen liegen. In Abbildung 19 ist ersichtlich, daß die Streubreite der Werte durch die der Gauß'schen Normalverteilung zugrundegelegte Standardabweichung σ variiert:
Mit Hilfe der Standardabweichung kann der Streuradius gesteuert werden, in dem sich das mutierte Individium im Vergleich zum ursprünglichen Individium befindet. Wird für jeden Parameter eine unabhängige Standardabweichung verwendet, erhält man die in Abbildung 22 dargestellten Wahrscheinlichkeitsellipsoide für den Aufenthaltsort der mutierten Individuen.

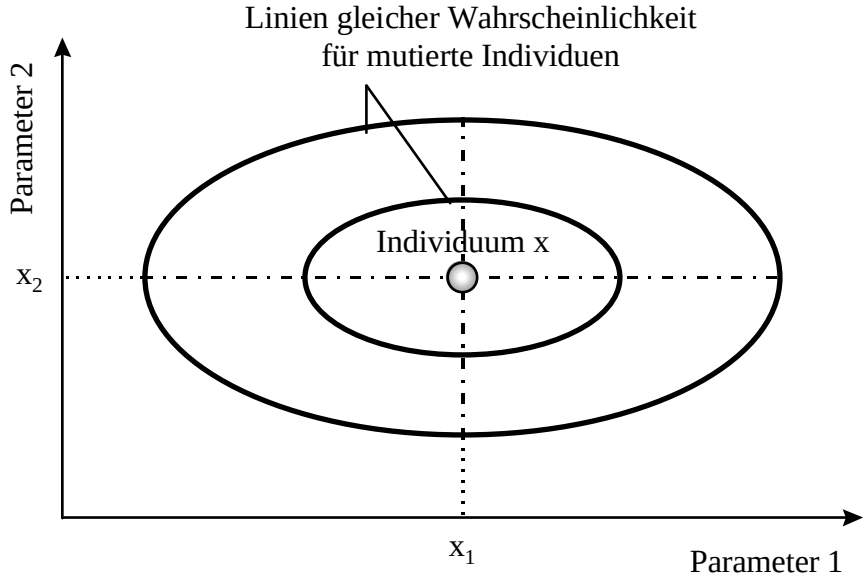


Abbildung 22: Wahrscheinlichkeitsellipsoid für Mutation

Die Ellipsoidachsen sind bei dieser Vorgehensweise parallel zu den Koordinatenachsen. Eine Verdrehung dieser Achsen kann durch Korrelationen ausgedrückt werden. Der Mutationsvektor z in Gleichung (25) wird hierbei mit einem durch die Korrelationen definierten Rotationstensor multipliziert. Während im n-dimensionalen Parameterraum für das Ellipsoid n Standardabweichungen existieren, sind $\frac{n(n-1)}{2}$ Korrelationen vorhanden. Die Standardabweichungen und Korrelationen steuern die Mutation und werden als Strategieparameter (siehe 6.6) bezeichnet.

6.4.2 Rekombination

Für die *Rekombination* stützt sich auf die in Kapitel 6.2 beschriebenen Modelle der biologischen Evolutionstheorie, wie z.B. den Mendel'schen Vererbungsgesetzen oder den Rekombinationsvorgängen bei der Meiose. Allgemein betrachtet wird durch Rekombination aus zwei oder mehreren Eltern ein Kindindividuum erzeugt.

$$rec: \quad I^{\mu} \rightarrow I \tag{41}$$

Jedes Merkmal (Parameter) der beteiligten Individuen wird hierbei getrennt betrachtet. Es sollen Rekombinationsoperatoren für zwei Eltern betrachtet werden. Die Modelle können auf mehrere Eltern erweitert werden, indem man für jedes Merkmal den zweiten Elter zufällig bestimmt und das Kindindividium somit Merkmalsausprägungen mehrerer Eltern besitzt. Ein Elter wird auch bei mehreren miteinbezogenen Eltern üblicherweise fest gewählt. Die einzelnen Rekombinationsmöglichkeiten für einzelne Merkmale sind im folgenden aufgeführt:

- Bei der *Crossover-Rekombination* wählt man zufällig den Parameterwert von Elter 1 oder Elter 2. Dieser Wert wird dem Parameter des Kindindividuums zugeteilt:
  $$x_i^K = x_i^{E1} \cup x_i^{E2} \tag{42}$$
- Die *intermediäre Rekombination* bildet jeweils den Mittelwert der Parameterwerte der Eltern für das Kindindividuum.
  $$x_i^K = \frac{x_i^{E1} + x_i^{E2}}{2} \tag{43}$$
- Die *zufällig intermediäre Rekombination* bestimmt den Kind-Parameterwert zufällig durch die gleichverteilte Zufallszahl z im Intervall zwischen den Elternwerten.
  $$x_i^K = z \cdot x_i^{E1} + (1-z) \cdot x_i^{E2} \quad z \in [0,1] \tag{44}$$

In Abbildung 23 werden die Rekombinationstypen im zweidimensionalen Parameterraum gezeigt:

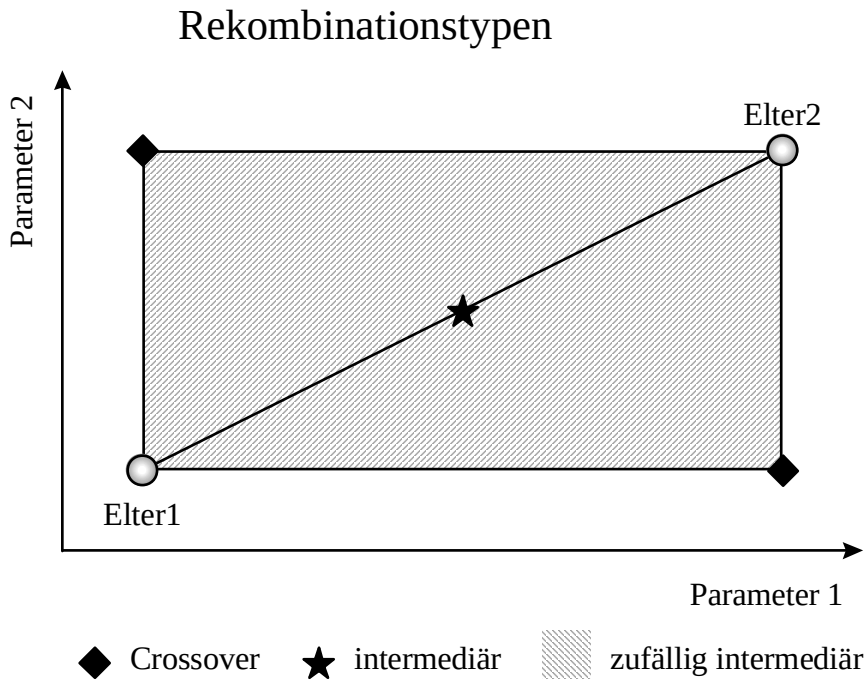


Abbildung 23: Rekombinationstypen

6.4.3 Selektion

Durch *Selektion* werden die Eltern einer neuen Generation bestimmt. Eltern erzeugen durch die Rekombination und Mutation Kinder, die zusammen die Generation einer Population bilden. Bei der Selektion sind grundsätzlich zwei Strategien zu unterscheiden:

- *Plus(+)-Strategie*: Die Eltern der kommenden Generation werden aus der gesamten Generation (Eltern + Kinder) selektiert.
- *Komma(,)-Strategie*: Die Eltern der kommenden Generation werden aus den Kindern selektiert. Die Eltern sterben nachdem Kinder erzeugt wurden.

Der Selektionsoperator kann folgendermaßen beschrieben werden:

$$sel_{\mu}^{k}: \quad I^{k} \rightarrow I^{\mu}; \quad k \in \{\lambda, \mu+\lambda\} \tag{45}$$

In dieser Diplomarbeit wurde eine modifizierte Plus-Strategie verwendet, mit der auch die Komma-Strategie simuliert werden kann. Hierzu wird die Fitneß der Eltern nach jeder Generation mit einem konstanten Strafterm P verschlechtert. Je nach Wahl des Strafterms entspricht diese Strategie für P = 0 der Plus-, und für einen sehr großen Wert für P, der Komma-Strategie.

Die Selektion sortiert aus der Ausgangsmenge die besten Individuen aus. Ausschlaggebend hierfür ist die Fitneßfunktion. Die Anzahl der auszusortierenden Individuen ist vorgeben.

Plus- und Kommastrategien unterscheiden sich wesentlich in ihren Eigenschaften. Plusstrategien lassen im Gegensatz zur Komma-Strategie keine Verschlechterungen zu. Die in Kapitel 6.6 erläuterte Selbstanpassung ist wiederum nur für die Komma-Strategie möglich.

6.5 Evolutionsstrategien mit einer Population

Eine Generation und damit ein Optimierungsschritt beinhaltet das Erzeugen der λ Kinder durch Rekombination und Mutation und eine anschließende Selektion der μ neuen Eltern. Bäck [17] gibt folgende mathematischen Formeln für diesen Vorgang an:

$$opt_{ES}: \quad I^{\mu} \rightarrow I^{\mu} \tag{46}$$

$$opt_{ES}\left(P^{(t)}\right) = sel_{\mu}^{k}\left(\cup_{i=1}^{\lambda}\left\{mut\left(rec\left(P^{(t)}\right)\right)\right\} \cup Q\right); \quad \{k, Q\} \in \left\{\{\lambda, 0\}, \left\{\mu + \lambda, P^{(t)}\right\}\right\} \tag{47}$$

Eltern und Kinder, die durch diese Abhängigkeiten zusammenhängen gehören einer Population an.

6.5.1 (1+1)-Evolutionsstrategie

Die einfachste Evolutionsstrategie stellt die eingliedrige (1+1)-ES dar. Sie wurde von Rechenberg [15] als Mutations-Selektions-Verfahren vorgestellt und kommt einem stochastischen Gradientenverfahren gleich. Biologische Parallelen können in der einfachen Fortpflanzung durch Mitose bei niedrigen Lebensformen gefunden werden (siehe6.2).
Es existiert bei diesem Modell nur ein Elter. Dieser wird geklont und das Duplikat durch Mutation verändert. Das mutierte Duplikat stellt das Kind dar, das mit dem Elter in Bezug auf seine Fitneß verglichen wird. Das bessere Individuum wird zum Elter der nächsten Generation. Die Strategie kann nach Bäck [17] mit folgender Formel beschrieben werden:

$$opt_{(1+1)-ES}\left(\{\vec{a}\}\right) = sel_{1}^{2}\left(\left\{mut(\vec{a})\right\} \cup \{\vec{a}\}\right) \tag{48}$$

wobei

$$sel_{1}^{2}\left(\left\{\vec{a}, \tilde{\vec{a}}\right\}\right) = \begin{cases} \left\{\tilde{\vec{a}}\right\} & , \text{für } f\left(\tilde{\vec{x}}\right) \leq f\left(\vec{x}\right) \\ \{\vec{a}\} & , \text{für } f\left(\tilde{\vec{x}}\right) > f\left(\vec{x}\right) \end{cases} \tag{49}$$

6.5.2 (μ‡λ)-Evolutionsstrategie

Bei mehrgliedrigen Evolutionsstrategien existieren ein oder mehrere Eltern, die mehrere Kinder erzeugen. μ gibt die Anzahl der Eltern an und λ die Anzahl der Kinder. Die Schreibweise (μ/ρ#λ) spezifiziert mit ρ die Anzahl der Eltern, wobei # als Platzhalter für + oder , steht. Im weiteren soll der Fall ρ=2 für zwei Eltern betrachtet werden. Nachdem die Eltern durch Rekombination die Kinder erzeugt haben, diese anschließend mutiert werden, müssen die Eltern für die nächste Generation selektiert werden. Hier unterscheiden sich die Plusstrategie, bei der aus allen Individuen gewählt wird, und die Kommastrategie, die nur Kinder als neue Eltern zuläßt (siehe6.4.3).

Bäck [17] notiert diese Strategien folgendermaßen:
(μ+λ)-Strategie:

$$opt_{(\mu+\lambda)-ES}\left(P^{(t)}\right) = sel_{\mu}^{\mu+\lambda}\left(\bigcup_{i=1}^{\lambda}\left\{mut\left(rec\left(P^{(t)}\right)\right)\right\} \cup P^{(t)}\right) \tag{50}$$

(μ,λ)-Strategie:

$$opt_{(\mu,\lambda)-ES}\left(P^{(t)}\right) = sel_{\mu}^{\lambda}\left(\bigcup_{i=1}^{\lambda}\left\{mut\left(rec\left(P^{(t)}\right)\right)\right\}\right) \tag{51}$$

Man wählt

$$\mu \leq \lambda \tag{52}$$

Für Kommastrategien ist dies notwendig, damit die Population nicht ausstirbt. Der Quotient

$$s = \frac{\mu}{\lambda} \tag{53}$$

wird als *Selektionsdruck* bezeichnet. Übliche Werte liegen zwischen $\frac{1}{3}$ und $\frac{1}{7}$.

6.6 Anpassung durch Strategieparameter

Das Erzeugen der Kinder durch Rekombination und Mutation sowie die Selektion wird von Faktoren gesteuert. Um die Konvergenzgeschwindigkeit der Optimierung zu maximieren ist man bestrebt, diese Faktoren von Generation zu Generation so anzupassen, daß ein möglichst großer Fortschritt zu erwarten ist. In den Strategieparametern spiegelt sich die Metrik der konventionellen Optimierungsverfahren wieder. Noch „unentdeckte“ Gebiete des Entwurfsraums werden hauptsächlich durch die Mutation erschlossen. Auch die Schrittweite des Fortschritts bestimmt sich zu einem großen Teil durch die Mutation, die wiederum von den verwendeten Standardabweichungen und eventuell auch den Korrelationen abhängt. Diese Werte sind die Strategieparameter mit denen man den Fortschritt steuert. Hier sind unterschiedlich komplexe Ansätze möglich.

6.6.1 Einfache Schrittweitensteuerung

Bei der einfachsten Variante wird eine Standardabweichung für alle Individuen und alle Parameter verwendet. In Abbildung 22 würden sich Kreise für die „Wahrscheinlichkeitshöhenlinien“ ergeben, die bei allen Individuen identisch sind. Eine Anpassung dieser Standardabweichung mit einer Schrittweitensteuerung nach einer bestimmten Anzahl von Generationen verbessert die Konvergenzgeschwindigkeit. Hierzu kann die Erfolgsrate der Verbesserung über mehrere Generationen bestimmt werden, um die Schrittweite dementsprechend zu erhöhen oder zu erniedrigen. Von Rechenberg [15] wurde für die in Kapitel 6.5.1 beschriebene (1+1)-ES Konvergenzgeschwindigkeiten und optimale Schrittweiten mathematisch für bestimmte Grundmodelle hergeleitet. Ein Grundmodell ist das sogenannte Kugelmodell, bei dem die Fitneß proportional zum Abstand zu einem Kugelmittelpunkt abnimmt.

Hierauf stützt sich die Rechenberg'sche[15] *1/5-Erfolgsregel*(siehe [18]):

„Der Quotient aus den erfolgreichen Mutationen (also den Mutationen, die eine Verbesserung der Qualität bewirken) zu allen Mutationen sollte mindestens 1/5 betragen. Ist der Quotient größer als 1/5, so sollte die Streuung der Mutation erhöht werden; ist der Quotient geringer (also die Mutationen seltener erfolgreich), so sollte die Streuung verringert werden."

Folgender algorithmischer Ansatz, der die Änderung der Standardabweichung von einer Generation t_i zu nächsten t_{i+1} beschreibt, stützt sich auf diese Regel:

$$p > \frac{1}{5} \quad \rightarrow \quad \sigma\left(t_{i+1}\right) = \frac{\sigma\left(t_i\right)}{c} \tag{54}$$

$$p < \frac{1}{5} \quad \rightarrow \quad \sigma\left(t_{i+1}\right) = c \cdot \sigma\left(t_i\right) \tag{55}$$

$$p = \frac{1}{5} \quad \rightarrow \quad \sigma\left(t_{i+1}\right) = \sigma\left(t_i\right) \tag{56}$$

Für den Faktor c sollte ≈0,85 gewählt werden.

Um die Auswirkungen der Schrittweitensteuerung zu verdeutlichen wurde die dem Kugelmodell entsprechende zweidimensionale Funktion $f1(x,y)=x^2+y^2$ (siehe Abbildung 24) untersucht.

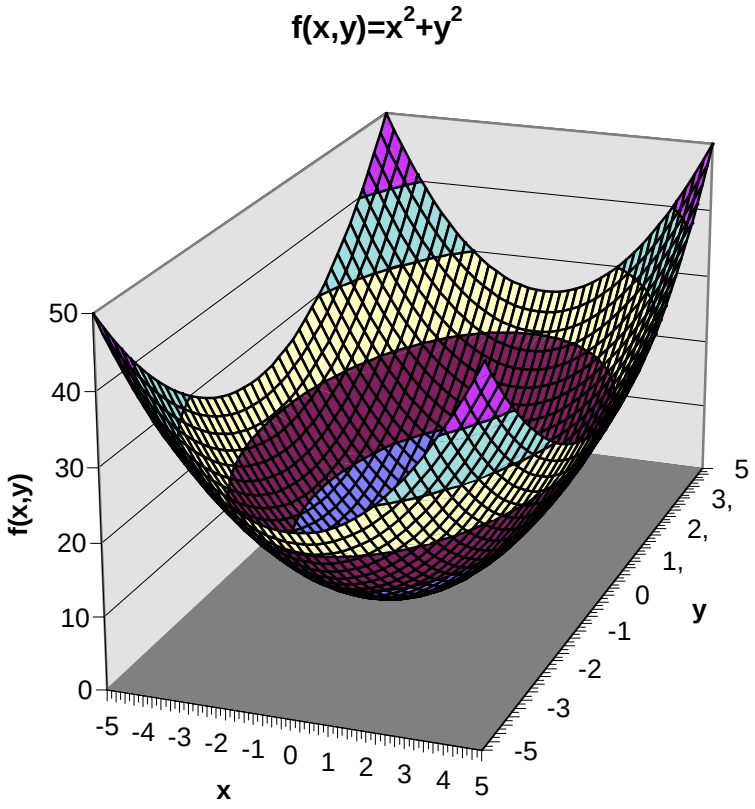


Abbildung 24: Testfunktion f1

Das für x = y = 0 vorhandene Minimum f1(0,0) = 0 soll mit einer (1,1)-ES mit einer Begrenzung auf 250 Generationen gefunden werden. Die Funktionsparameter x und y sind auf das Intervall [-5, 5] beschränkt. Die Standardabweichung für den Mutationsoperator wird mit 5.0 vorgegeben. Für den Testlauf mit Schrittweitensteuerung kam die 1/5-Regel zur Anwendung, wobei nach jeweils 10 Generationen eine Auswertung stattfand.
Auswertung:

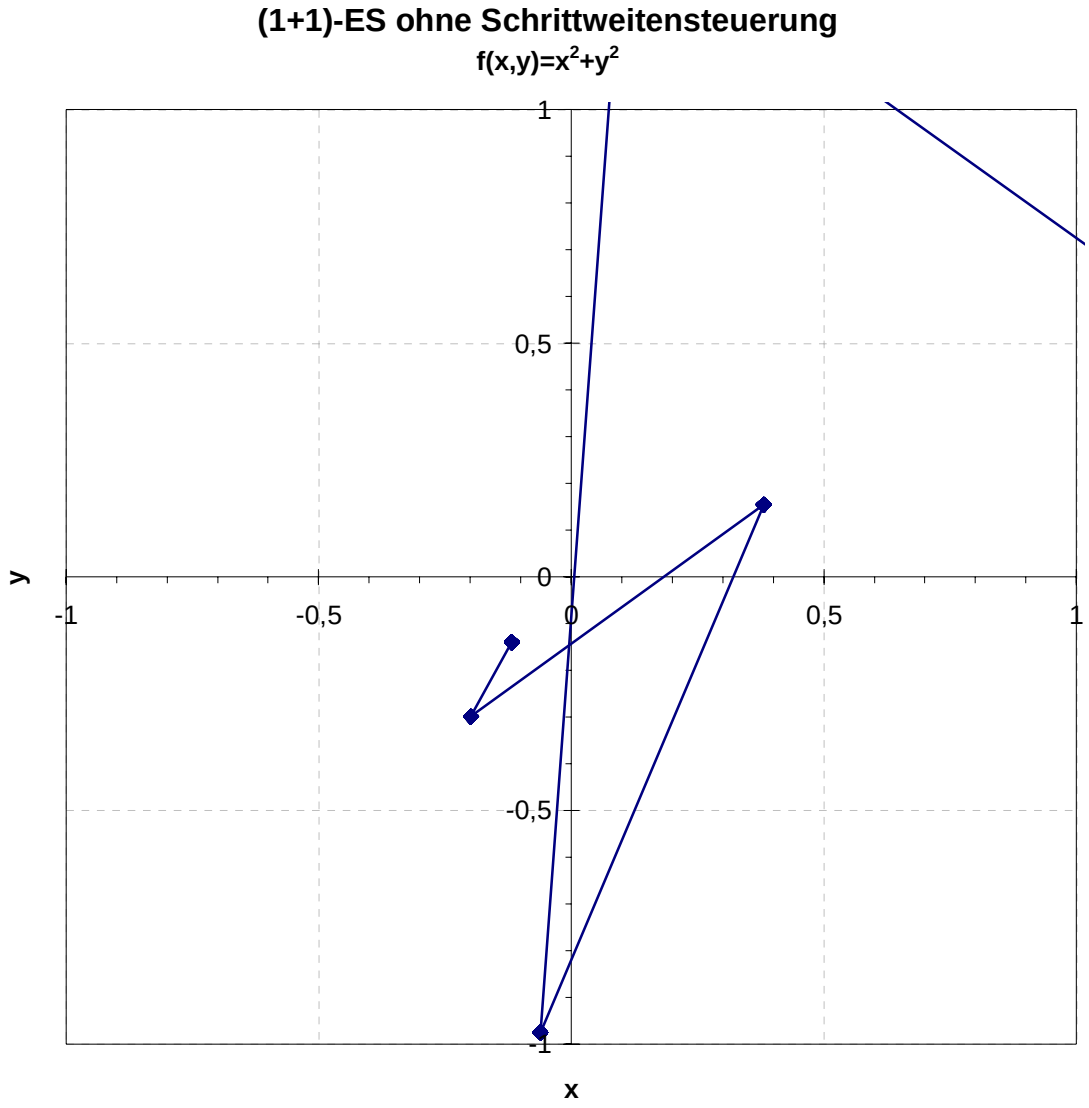


Abbildung 25: (1+1)-ES ohne Schrittweitensteuerung

Das Endresultat des Testlaufs ohne Schrittweitensteuerung ergab für x = -0,276 und y = 0,131 und somit einen Funktionswert von f = 0,093. Die ES ohne Schrittweitensteuerung zeigt zu Beginn gute Resultate, schnürt aber bei Erreichen des Minimums den Suchradius nicht genügend ein. Die durch die Standardabweichung von 5,0 bedingte Schrittweite ist für eine weitere Verbesserung des Ergebnisses zu groß und die Kindindividuen in einem zu

großen Durchmesser gestreut. Der Optimierungsprozeß bleibt somit stecken. Theoretisch würde für unendlich viele Generationen das optimale Ergebnis gefunden.
Mit Schrittweitensteuerung liegt das Endresultat bei $x = -3{,}8 \cdot 10^{-3}$ und $y = -7{,}0 \cdot 10^{-3}$ mit einem Funktionswert von $f = 6{,}4 \cdot 10^{-5}$. Dabei wurde die Standardabweichung im Verlauf der

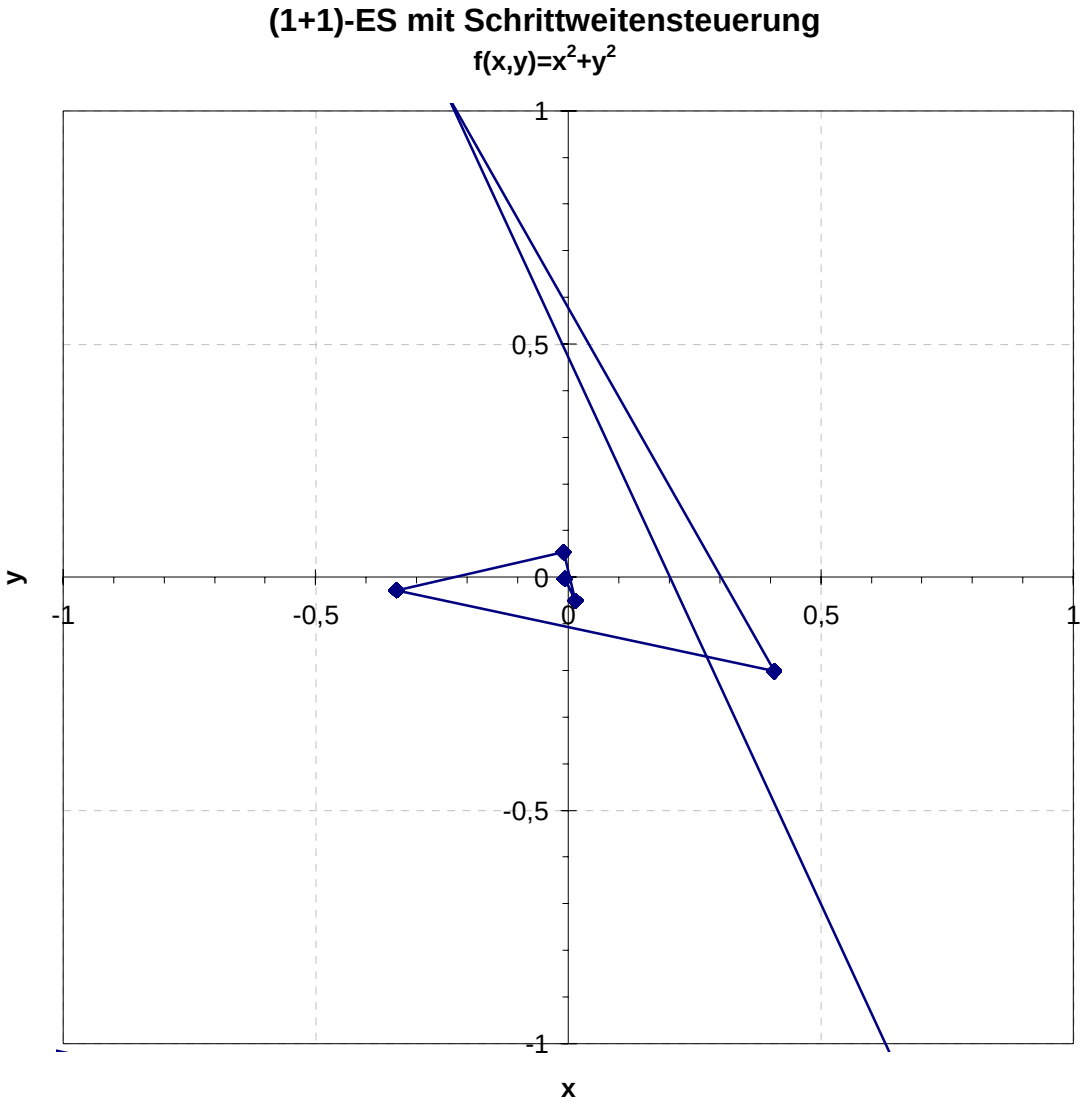


Abbildung 26: (1+1)-ES mit Schrittweitensteuerung

Suche von 5,0 auf $4{,}3 \cdot 10^{-2}$ gesenkt. Die kleinere Standardabweichung sorgt für kleinere Schrittweiten, die für eine weitere Verbesserung des Ergebnisses vorteilhaft sind. Deshalb wird mit der Schrittweitensteuerung ein besseres Resultat erzielt. Allerdings sind beim einfache Sphärenmodell keine lokalen Minima vorhanden, an denen der Optimierungsprozeß stecken bleiben könnte. Bei lokalen Minima wirkt sich die Einschnürung negativ aus, da die Wahrscheinlichkeit, das globale Minimum zu finden, herabgesetzt wird. Bei diskreten Pro-

blemen ist eine Einschnürung nur sinnvoll, solange noch andere Parameterwerte erreicht werden.

6.6.2 Selbstanpassung

Bei der in Kapitel 6.6.1 beschriebenen Schrittweitensteuerung müssen sowohl die Anfangsstandardabweichung, als auch der Faktor c für die Schrittweitensteuerung fest vorgegeben werden und können somit nicht problemabhängig bestimmt werden. Dies wird durch eine Selbstanpassung erreicht. Hierbei wird jedem Individuum neben den Phänotypparametern auch die Strategieparameter zugeteilt.

$$\vec{a} = \left[\underbrace{(x_1 \ldots x_n)}_{\vec{x}}, \underbrace{(\sigma_1 \ldots \sigma_{n_\sigma})}_{\vec{\sigma}}, \underbrace{(\alpha_1 \ldots \alpha_{n_\alpha})}_{\vec{\alpha}} \right] \in I; \quad 1 \le n_\sigma \le n; \quad n_\alpha = \left(n - \frac{n_\sigma}{2} \right) \cdot (n_\sigma - 1) \tag{57}$$

Das Individuum a bekommt hiermit noch seine Mutationseigenschaften zugeteilt. Es ist weiterhin möglich, eine Standardabweichung σ für alle Parameter zu verwenden. Die Erweiterung auf n Standardabweichungen σ_i verallgemeinert den kreis- oder kugelförmigen Suchraum zu einem elliptischen oder ellipsoiden (siehe 6.4.1). Optional können noch die Korrelationen α_i aufgenommen werden, die eine Verdrehung des Ellipsoides bewirken.
Die Strategieparameter werden ebenso wie die Parameter x rekombiniert und mutiert. Die somit erzeugten neuen Strategieparameter für das Kindindividuum werden für die Mutation des Kindes verwendet. Die Mutation der Strategieparameter muß also vor der Mutation der Parameter durchgeführt werden. Die Mutation der Strategieparameter wird üblicherweise nicht additiv, sondern multiplikativ mit Hilfe eines Exponentialterms, durchgeführt:

$$mut(\vec{\sigma}, \tau) = \vec{\sigma} \cdot e^{N(0,\tau)} \tag{58}$$

Der Grundgedanke der Selbstanpassung ist, daß sich auf lange Sicht nur gute Individien mit guten Mutationseigenschaften durchsetzen. Dies ist allerdings nur bei der Komma-Strategie möglich. Bei der Plusstrategie überlebt ein Individuum mit sehr hoher Fitneß, aber schlechten Mutationseigenschaften sehr lang und vererbt diese schlechte Eigenschaft den Kindern.

6.6.3 Anpassung durch globale Standardabweichung

Für mehrgliedrige ES wurde in dieser Diplomarbeit eine neue Strategie getestet. Hierfür werden n Standardabweichungen der Parameterwerte aller Eltern berechnet:

$$\sigma_i^E = \sqrt{\frac{\sum \left(x_i - \bar{x}_i \right)^2}{n}} = \sqrt{\frac{\sum x_i^2}{n} - \left(\frac{\sum x_i}{n} \right)^2} \tag{59}$$

Diese Standardabweichungen werden zur Mutation der Kinder verwendet. Die Elternindividuen bilden eine „Wolke“ im Parameterraum. Der Schwerpunkt ist durch die Mittelwerte bestimmt und die jeweiligen Ausdehnungen durch die in (59) berechneten Standardabweichungen. Die Eltern stellen die selektierten Individuen mit guten Eigenschaften dar. Falls sich die Eigenschaften bei allen Eltern ähneln, wird davon ausgegangen, daß dies der endgültigen Lösung nahekommt und der Suchradius wird eingeschnürt. Die Anzahl der Eltern

darf allerdings nicht zu gering sein. Bei multimodalen Problemen zieht sich die Wolke dagegen sehr schlecht zusammen.

6.7 Restriktionen und Nebenbedingungen

Die Parameterwerte müssen in der Definitionsmenge liegen. Es wird festgelegt, ob ganzzahlige oder auch reale Werte zulässig sind. Die Größe des Werts ist durch ein Intervall beschränkt, in dem der Parameterwert liegen darf. Diese Restriktionen werden direkt in den Algorithmus integriert. Es sind aber eine Vielzahl Nebenbedingungen denkbar, die Teile des Parameterraums für ungültig erklären und ausgrenzen. Eine Möglichkeit besteht darin, Individuen, die gegen diese Restriktionen verstoßen, sofort zu eliminieren. Eine weitere Behandlung der Nebenbedingungen ist die Einführung von Straftermen, die die Fitneß verschlechtern, um so diese Individuen indirekt „auszurotten".

$$\tilde{f}(\vec{x}) = \begin{cases} f(\vec{x}), & \text{für } g(x_j) \geq 0,\ j = 1 \ldots q \\ f(\vec{x}) + S, & \text{für } g(x_j) < 0,\ j \in [1, q] \end{cases} \tag{60}$$

6.7.1 Strafterme

Die Wirkungsweise von *Straftermen* soll an einem einfachen Beispiel verdeutlicht werden: Das Minimum der linearen Funktion f(x) = x soll im Intervall [0,3] bestimmt werden. Als Nebenbedingung ist für den Funktionswert $f(x) \geq f_{grenz} = 1$ vorgegeben, also $g(x) = f(x)-f_{grenz} = f(x)-1$ Verschiedene Ansätze für den Strafterm zeigen, wie die ursprüngliche Funktion so abgeändert wird, daß das Ergebnis x = 1 gefunden werden kann.
In einem ersten Versuch wird versucht einen konstanten, ausreichend großen Wert (S=2) als Strafterm zu addieren. In Abbildung 27 ist die modifizierte Funktion dargestellt.

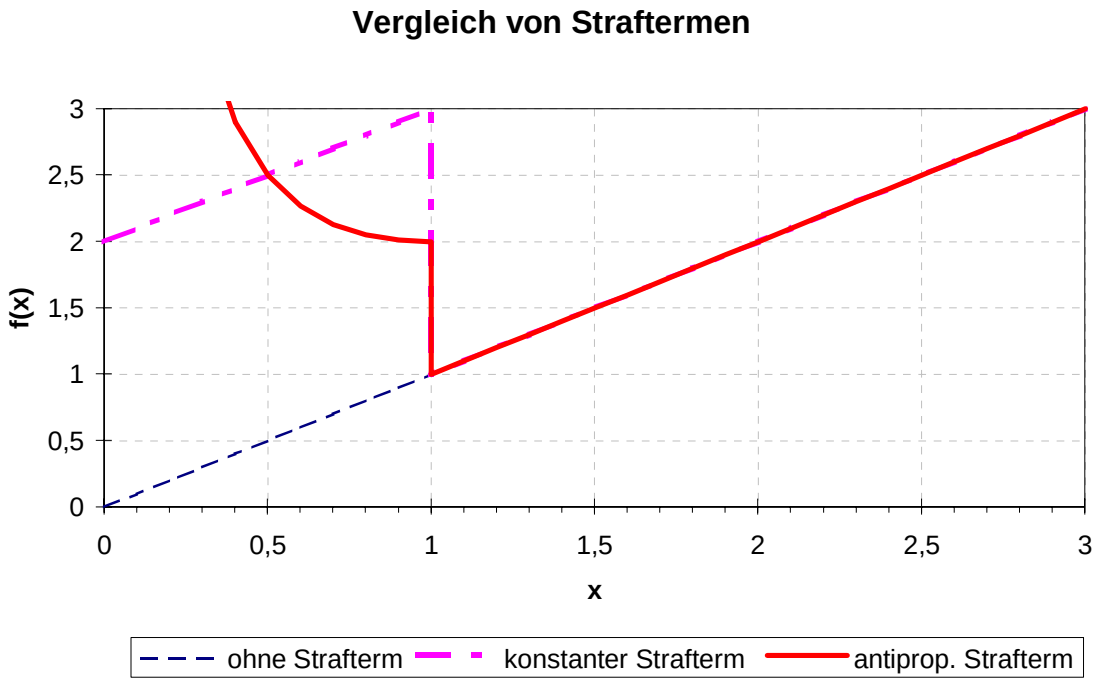


Abbildung 27: Vergleich von Straftermen

Das globale Minimum ist nun für x = 1 zu finden. Problematisch ist der Gradient, der im Interval [0,1] auf ein lokales Minimum für x = 0 zeigt. Ein Optimierungsalgorithmus wird somit zum falschen Wert geleitet. Für Evolutionsstrategien ist dies aber nur relevant, wenn die Mehrheit der Eltern die Nebenbedingungen verletzt. Der konstante Strafterm stellt in diesem Fall also eine ungünstige Variante dar.
Ein der Verletzung der Nebenbedingung antiproportionaler Strafterm kann durch folgende Formel gebildet werden:

$$S = -g(x_j) \cdot m + c, \quad \text{für } g(x_j) < 0 \tag{61}$$

Für den vorliegenden Fall wurden der Parameter m für x>0 mit

$$m = \frac{c}{f(x)} \tag{62}$$

so angepaßt, daß sich

$$S = \frac{f_{grenz}}{f(x)} \cdot c \tag{63}$$

ergibt. Mit c = 1 erhält man die Kurve nach Abbildung 27. Es ist leicht zu erkennen, daß sich hier der Gradient günstiger auswirkt.
Evolutionsstrategien sind stabil in Bezug auf die Stetigkeit und Differenzierbarkeit der zu optimierenden Ziel- oder Fitneßfunktion. Die durch die Strafterme gebildeten Ersatzfunktionen müssen deshalb nicht stetig oder differenzierbar sein.

6.8 Parallele Evolutionsstrategien

Die Natur bedient sich paralleler Vorgänge, um in möglichst kurzer Zeit eine gute Anpassung an die Umwelt zu erreichen. Eine Vielzahl von Individuen und Populationen existieren zur gleichen Zeit. Auch die Evolutionsstrategien eignen sich gut zur Parallelisierung.

6.8.1 Parallele Implementierung von Evolutionsstrategien mit einer Population

Der aufwendigste Teil der Evolutionsstrategien stellt in den meisten Fällen die Ermittlung der Fitneßwerte für die Individuen dar. Sowohl bei numerischen Berechnungen als auch bei experimentellen Versuchen sind diese einzelnen Werte in den hier betrachteten ES unabhängig voneinander. Es liegt nahe, diese Fitneßwerte nicht sequentiell, sondern parallel zu bestimmen. Bei einem MIMD Master-Slave Konzept verwaltet der Master die Population und verteilt über einen Scheduler die Fitneßberechnungen an die Slaves.

6.8.2 Evolutionsstrategien mit Multipopulationen

Es kann auch mit mehreren Populationen zugleich gearbeitet werden.
Im einfachsten Fall sind diese Populationen voneinander isoliert. Man spricht hierbei von einem *Inselmodell*. Der Speedup (siehe 2.1.4), der bei einer Parallelisierung mit einer Population pro PE erreicht wird, ist direkt proportional zur Anzahl der „Inseln“. Eine Kom-

munikation zwischen den PEs ist während der Berechnung nicht nötig. Die Konvergenzgeschwindigkeit wird nicht gesteigert. Die Sicherheit das globale Optimum zu finden, erhöht sich mit wachsender Populationsanzahl.
Dieses Konzept kann erweitert werden, indem die einzelnen Populationen Individuen austauschen. Hierbei ersetzen nach bestimmten Zeitabständen eine gewisse Anzahl der besten Individuen einer Population die schlechtesten Individuen einer anderen. Wird die Austauschrate zu hoch gewählt, erhält man nach kurzer Zeit identische Populationen.
Das *Netzwerk-Modell* baut feststehende Verbindungen zwischen Populationen, wie sie beim Insel-Modell vorkommen, auf. Diese Verbindungen sind ähnlich den in Kapitel 2.2.1 vorgestellten Punkt-zu-Punkt Verbindungen bei Mehrprozessorrechnern, wie die Ringstruktur oder eine vollständige Vernetzung. Hierbei steigt der Kommunikationsaufwand. Der Austausch kann synchron oder asynchron vorgenommen werden. Beim synchronen Austausch tauschen alle Populationen nach einer bestimmten Anzahl von Generationen gleichzeitig ihre Individuen aus, während der asynchrone Austausch zeitlich versetzt abläuft. Die statischen Verbindungen des Netzwerk-Modells sind auch während der Berechnung nach bestimmten Gesetzmäßigkeiten modifizierbar. Ein solches Modell, das die Populationen zum Austausch bestimmt, bezeichnet man als*Migrations-Modell*.

6.9 Untersuchung anhand einer Testfunktion

Anhand einer von Schwefel vorgeschlagenen Testfunktion (vgl. [18]), die in Abbildung 28 dargestellt ist, wird demonstriert, wie die Evolutionsstrategie das Minimum bei x = y = 420,9687 findet.

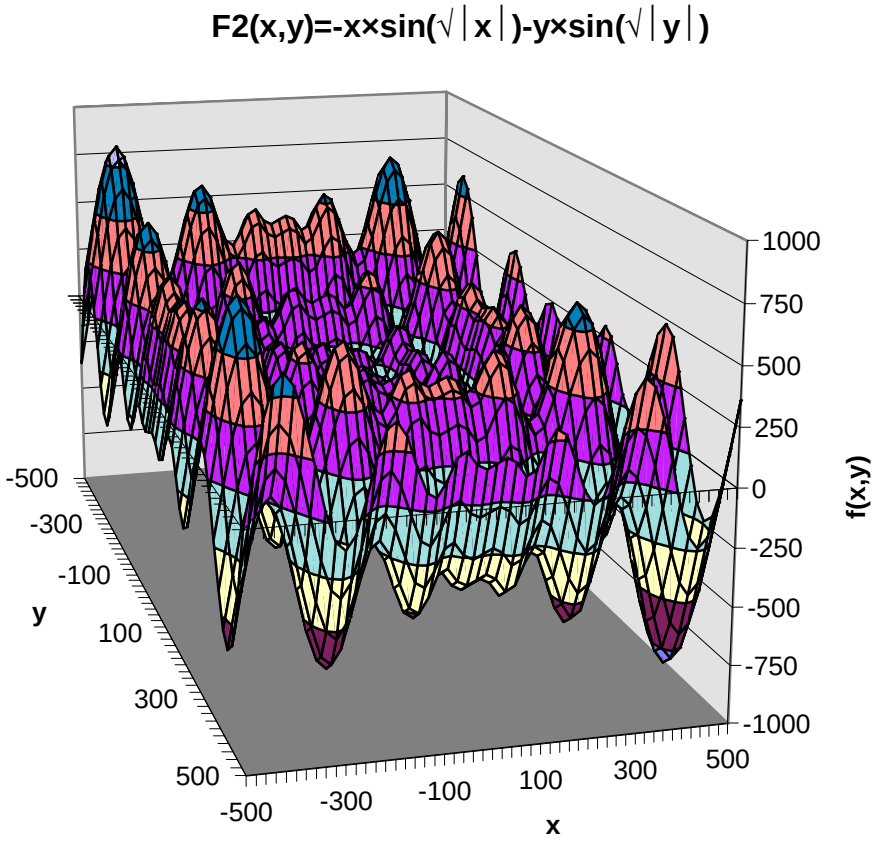


Abbildung 28: Testfunktion f2 von Schwefel

Das Problem wird mit einer (15,100)-ES angegangen. Für die Parameter wird eine einfache Crossover- und für die n Standardabweichungen eine intermediäre Rekombination gewählt. Für jede Generation eines Programmlaufs sind jeweils die Eltern in Abbildung 29 markiert. Die Funktion besitzt mehrere lokale Minima. Man erkennt deutlich wie sich nach wenigen Generationen die Eltern um verschiedene lokale Minima anhäufen. Das globale Minimum setzt sich durch und die Elternverteilung zieht sich um diesen Punkt zusammen.

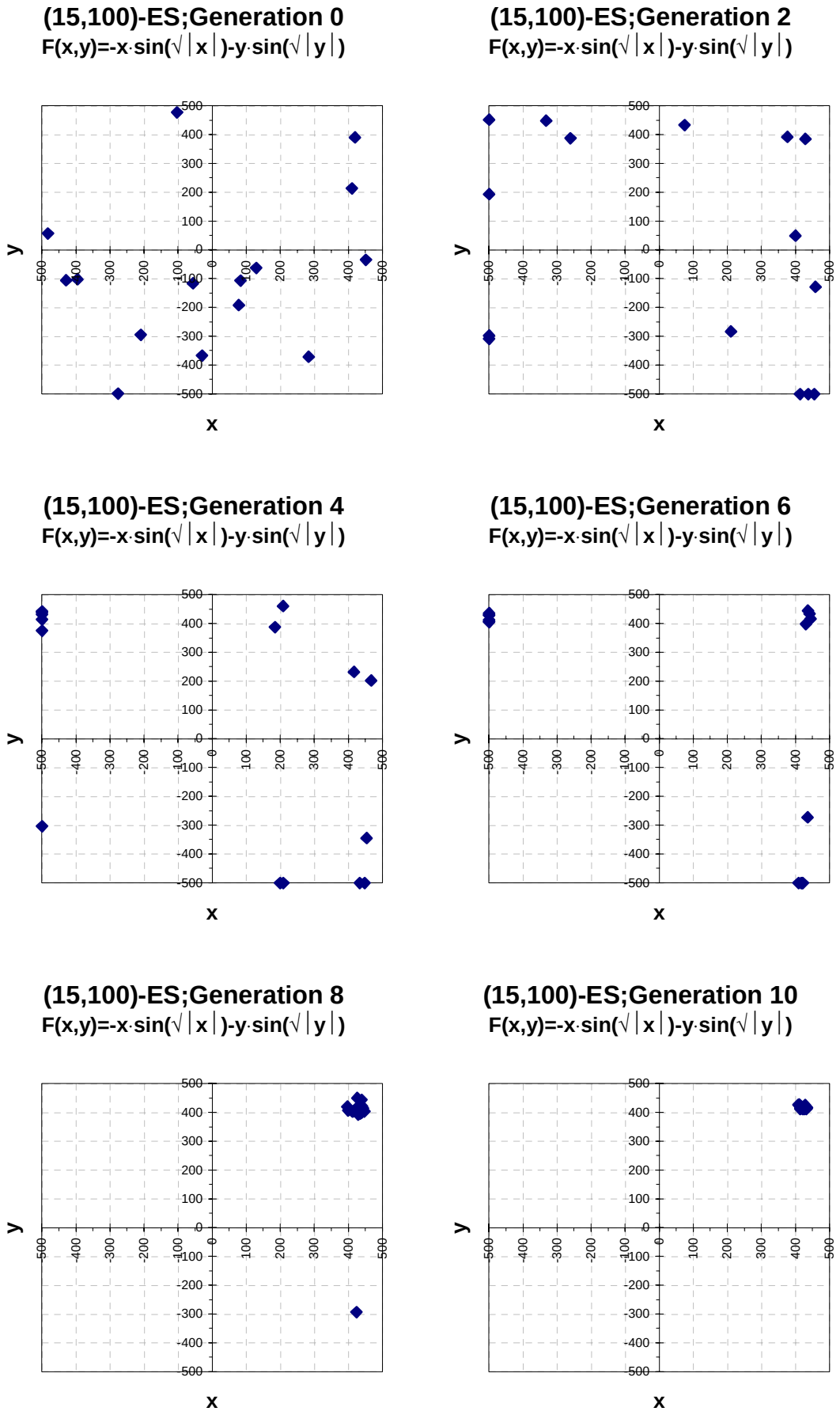


Abbildung 29: Elter-Individuen

7 Algorithmus zur parallelen Evolutionsstrategie

Das in der Programmiersprache C geschriebene Programm setzt unter Verwendung der PVM-Bibliothek die Evolutionsstrategie für die diskrete Optimierung von Fachwerken auf einem verteilten System um. Dabei ist das Gesamtgewicht des Fachwerks zu minimieren. Als Nebenbedingungen sind Grenzwerte für maximale Normalspannungen sowie maximale Verschiebungen einzuhalten.

Das Programmierkonzept entspricht dem klassischen Master-Slave Modell. Der Master verwaltet den Evolutionsstrategie-Algorithmus und verwendet die Slaves zur parallelen Berechnung der Fitneßfunktion für die einzelnen Individuen. Die Fitneßfunktion entspricht dem Gesamtgewicht G und hängt von der Länge l und der Querschnittsfläche A aller n Stäbe ab:

$$G = \sum_{i=1}^{n} \gamma \cdot l_i \cdot A_i \tag{64}$$

Die Wichte γ kann als konstanter Faktor vernachlässigt werden, was einer Optimierung des Materialvolumens gleichkommt. Die Fitneßfunktion wird bei Verletzung der Restriktionen mit Straftermen beaufschlagt. Hierfür muß eine statische Untersuchung des Fachwerks erfolgen, um die einzelnen Normalspannungen und Verschiebungen zu ermitteln, mit den Grenzwerten zu vergleichen und gegebenenfalls den Strafterm zu berechnen. Dieser Schritt erfolgt durch die einzelnen Slaves, die jeweils sequentiell arbeiten. Zur Lösung des dabei zu berechnenden linearen Gleichungssystems, werden LAPACK-Treiberfunktionen verwendet.

Das Programm verbindet ein klassisches Fachwerk-Stabwerksprogramm mit einer parallelen Evolutionsstrategie mit einer Population. Die Parameter sind ganzzahlige Werte, die die Querschnitte festlegen. Ein Satz von n Querschnitten legt ein Individuum fest. Alle Individuen sind in einem Feld gespeichert. Ein weiterer Vektor legt die einzelnen Fitneßwerte fest. Für die Selektion wird dieses Feld anhand der Fitneßwerte mit einem Quicksort-Algorithmus über einen Indexvektor sortiert.

7.1 Programmaufbau

Das Flußdiagramm in Abbildung 30 zeigt die Funktionsweise des Masterprogramms:

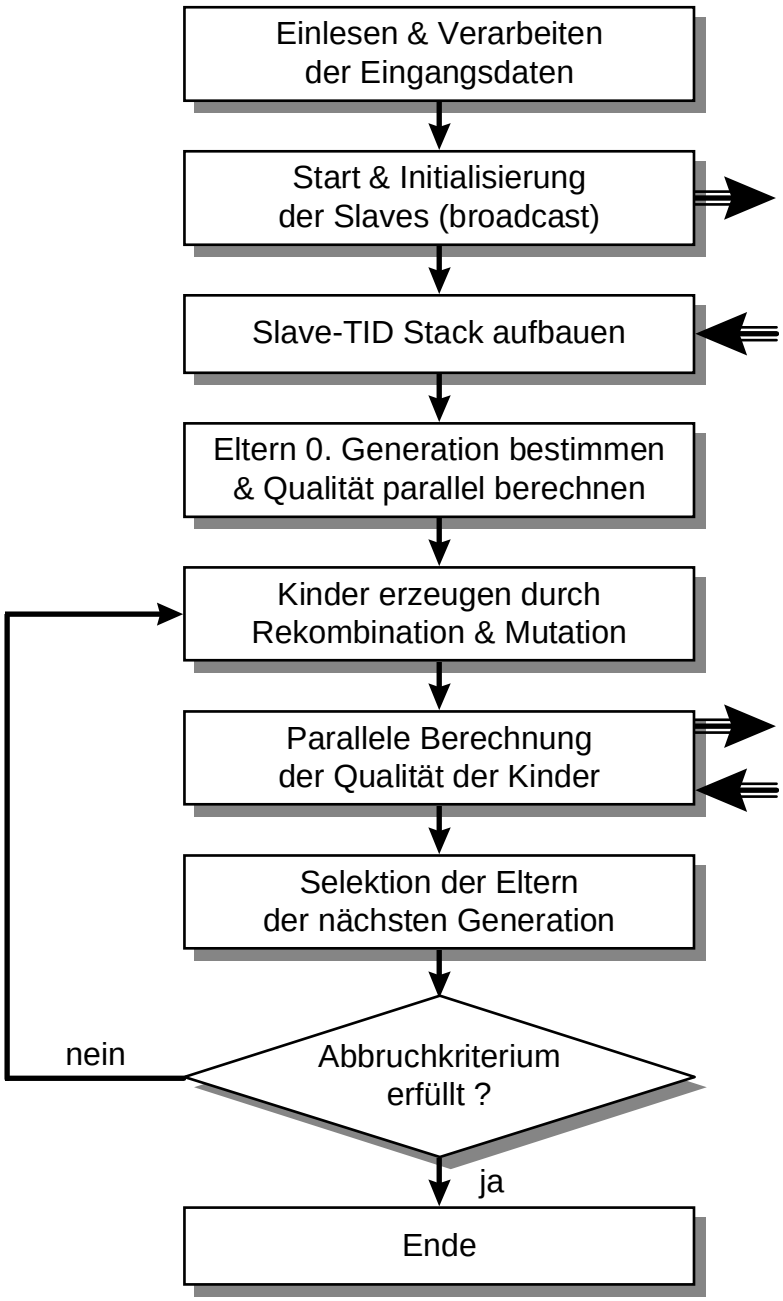


Abbildung 30: Flußdiagramm - Master

7.1.1 Eingangsdaten

Die Eingangsdaten liegen in Form einer Eingabedatei vor. Einzelne Eingabeblöcke sind durch Kenner getrennt. Im einzelnen sind das:

- NODES:
  - Knotengeometrie (Knotennummer und -koordinaten)

- SUPPORTS:
  - Auflagerbedingungen (Knotennummer und entsprechender Auflagercode)
- MATERIALS:
  - Werkstoffparameter (Werkstoffnummer, E-Modul und zul. Spannung)
- SECTIONS:
  - Querschnitte (Querschnittsnummer und Querschnitt)
  - die angegebenen Querschnitte stellen die Auswahlliste für die ES dar.
- ELEMENTS:
  - Stabtopologie (Anfangs- und Endknoten)
  - Zuordnung des Werkstoffs aus MATERIALS
  - Eine Zuordnung des Querschnitts ist nicht erforderlich, da es vom Programm selbst vorgenommen wird.
- LOADS:
  - Lastdaten (Knotennummer und Lastkomponenten)
- EVOLDAT:
  - Parameter für die Evolutionsstrategie. (Anzahl der zu berechnenden Generationen, Anzahl der Eltern und Kinder, Straftermparameter)

7.1.2 Initialisierung und Steuerung der Slaves

Die Topologie und Geometrie des Fachwerks soll während der Optimierung nicht verändert werden. Stablängen, -orientierung, sowie die globale Numerierung der Freiheitsgrade und die Zuordnung auf die lokalen Unbekannten bleiben konstant. Die Bestimmung und Berechnung dieser Werte erfolgt nur einmal im Masterprogramm.

In einer Initialisierungsphase versendet der Master diese Daten an die einzelnen Slaves über einen Broadcast. Nachdem die Slaves alle Initialisierungsdaten empfangen haben, senden sie als Bestätigung ihren Task-Identifier zurück. Der Master baut sich aus diesen TID in einem Feld einen Stack auf, der dem Scheduler als Grundlage dient. Nach der Initialisierungsphase warten die Slaves auf die Übermittlung der variablen Querschnittsdaten und liefern den Fitneßwert zurück. Den Querschnittsdaten wird eine Identifikationsnummer beigefügt, die der Slave unverändert zurückschickt. Dies erspart dem Master eine Verwaltung der Berechnungsjobs. Die Identifikationsnummer stellt die Zeilennummer der Matrix dar, in der die Daten des Individuums gespeichert sind. Der berechnete Fitneßwert wird mit Hilfe der Identifikationsnummer an die richtige Stelle des Fitneßvektors geschrieben. Die Übermittlung eines festgelegten Codes beendet den Slave.

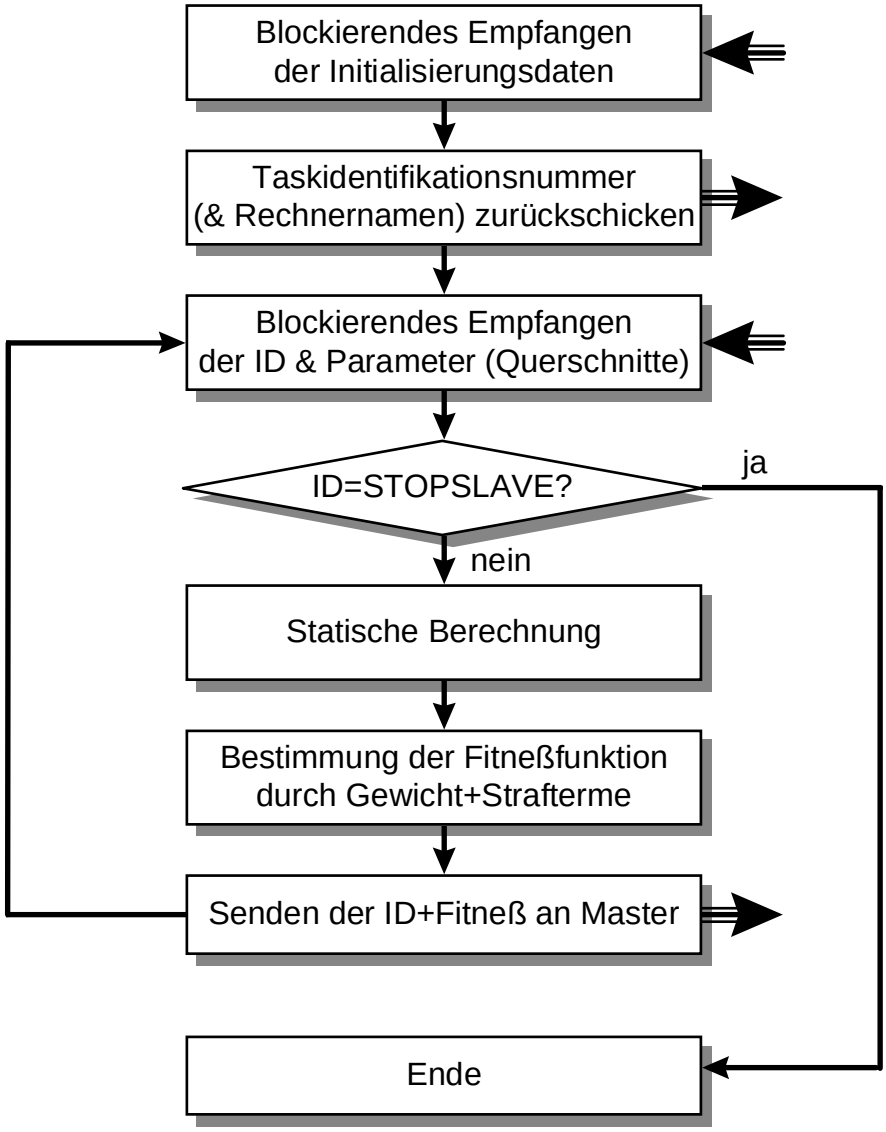


Abbildung 31: Flußdiagramm - Slave

7.1.3 Scheduler

Der Kern der parallelen Berechnung ist der Scheduler, der im Masterprogramm die Vergabe der Berechnungsjobs an die Slaves verwaltet. Abbildung 32 zeigt seine Funktionsweise:

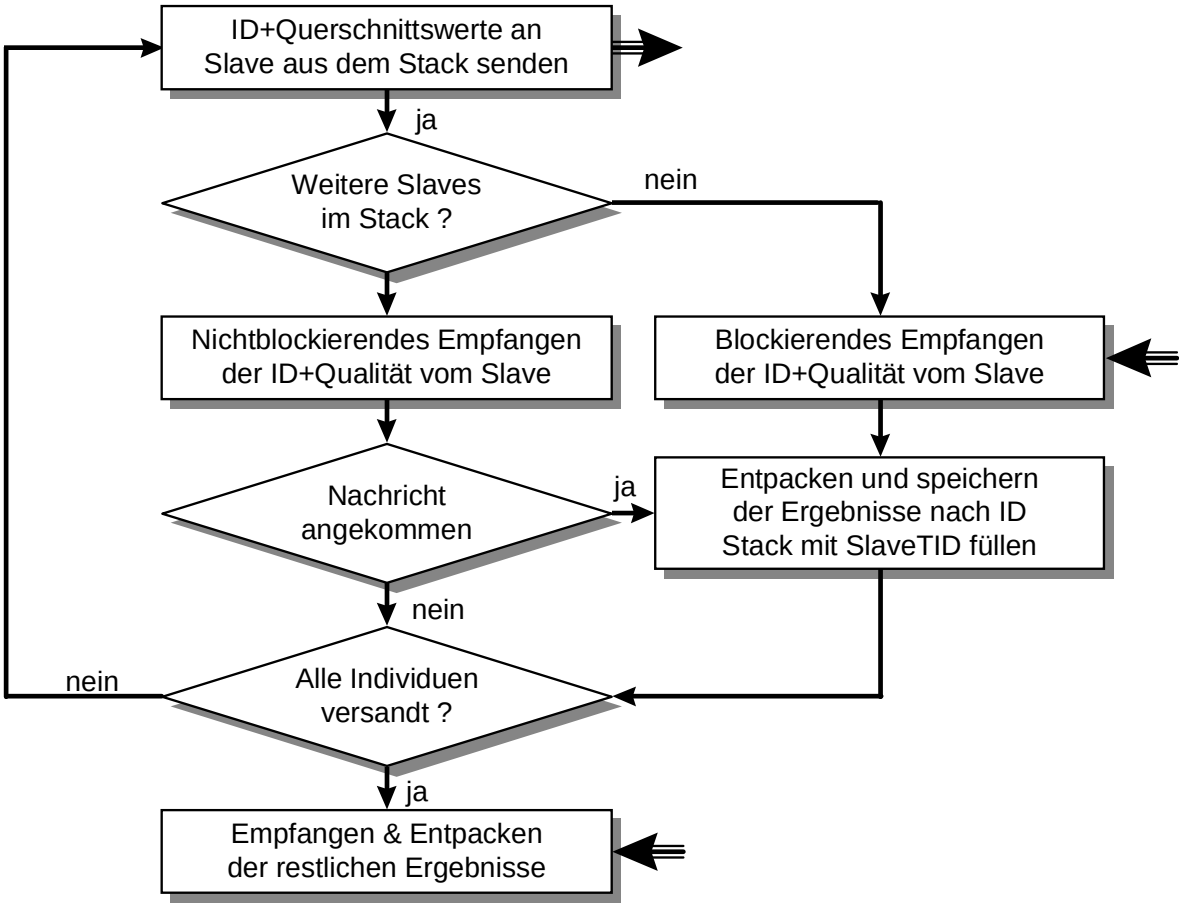


Abbildung 32: Flußdiagramm - Scheduler

Das Verfahren kann nach Rothermel [9] als statisches Verfahren angesehen werden. Bereits vergebene Berechnungsjobs können, im Gegensatz zu den aufwendigen dynamischen Verfahren, nicht mehr migriert werden. Da der Master die Verteilung der Jobs übernimmt, ist das verwendete Verfahren den sender-initiierten Lastausgleichsverfahren zuzuordnen.
Ein Slave der einen Berechnungsjob beendet hat, wird im FIFO-Stapel abgelegt. Durch dieses Stapelprinzip wird erreicht, daß möglichst wenige Rechner möglichst optimal ausgelastet werden. Eine Warteschlange (queue) lastet alle Rechner tendenziell gleich aus. Es ist aber vorteilhaft, zu möglichst wenigen Rechnern Verbindungen aufbauen zu müssen falls eine verbindungsorientierte Kommunikation gewählt wird.
Eine Erweiterung dieses Konzepts besteht darin, den Slaves zusätzlich zur ID den Zeitpunkt des Versendens der Nachricht mitzugeben der von den Slaves mit den Berechnungsdaten zurückgeschickt wird. Beim Empfang der Slavedaten kann der Master aus dieser Startzeit die Gesamtzeit der jeweiligen Berechnung mit Versendezeit ermitteln. Ein Sortieren des Vergabestack „bedient“ die schnellsten Rechner bevorzugt. Zusätzlich zu dieser Zeitmarke sendet der Slave seine Gesamtberechnungszeit. Aus diesen beiden Informationen kann die

Kommunikationszeit und das Verhältnis dieser Zeit zur Gesamtvergabezeit bestimmt werden.

Bei der Evolutionsstrategie müssen pro Generation die Fitneßfunktionswerte für λ Kinder bestimmt werden. Diese Berechnungen verteilt der Master auf n Slaves. Die kleinste Berechnungseinheit stellt hierbei die Fitneßberechnung von einem Individuum dar. Die geringste Kommunikation und damit die grobgranularste Aufteilung ist eine Berechnungseinheit von $\frac{\lambda}{n}$ Individuen. Damit führt jeder Slave pro Generation nur eine Berechnung durch, falls der erste Slave seine Berechnung nicht beendet hat bevor der letzte seinen Auftrag erhält. Der Algorithmus kann bei dieser Aufteilung keinen flexiblen Lastausgleich mehr durchführen. Wird bei jeder Zuteilung nur ein Individuum berechnet, erhalten langsamere Rechner im Schnitt weniger Berechnungen zugeteilt. Die Kommunikation wächst aber, da bei identischen Leistungen der Slaves $\frac{\lambda}{n}$ Zuteilungen erfolgen müssen. Der günstigste Wert ist auch von der Größe einer Berechnung abhängig.

7.1.4 Lastaufteilung auf Slaves

Im folgenden werden Versuche auf dem HP-Workstationcluster des Instituts für Baustatik verglichen. Auf dem Rechner „bcip2“ wurde der Master des beschriebenen Programms gestartet. Die 14 Slaves startet dieser Master auf den Maschinen „bcip2“-„bcip15“[*]. Der Scheduler vergibt pro Zuteilung die Berechnung eines Individuums. Verglichen wird ein Fachwerk mit 10 Stäben mit einem Fachwerk mit 333 Stäben, bei dem der Berechnungsaufwand pro Individuum um ein Vielfaches höher ist. Der Slave auf dem Master-Rechner muß nicht über das Netzwerk angesprochen werden und die Kommunikation ist hier bedeutend schneller. Das wirkt sich bei kleinen Berechnungseinheiten stärker aus als bei großen. Für das Fachwerk mit 10 Elementen müßten also immer mehrere Individuen zugleich zur Berechnung versendet werden. Bei einem Vergleich der Lastverteilung in Abbildung 33 und Abbildung 34 auf Seite 77 ist diese Tendenz eindeutig zu erkennen.

[*] Hewlett Packard Risc Workstations HP700/33

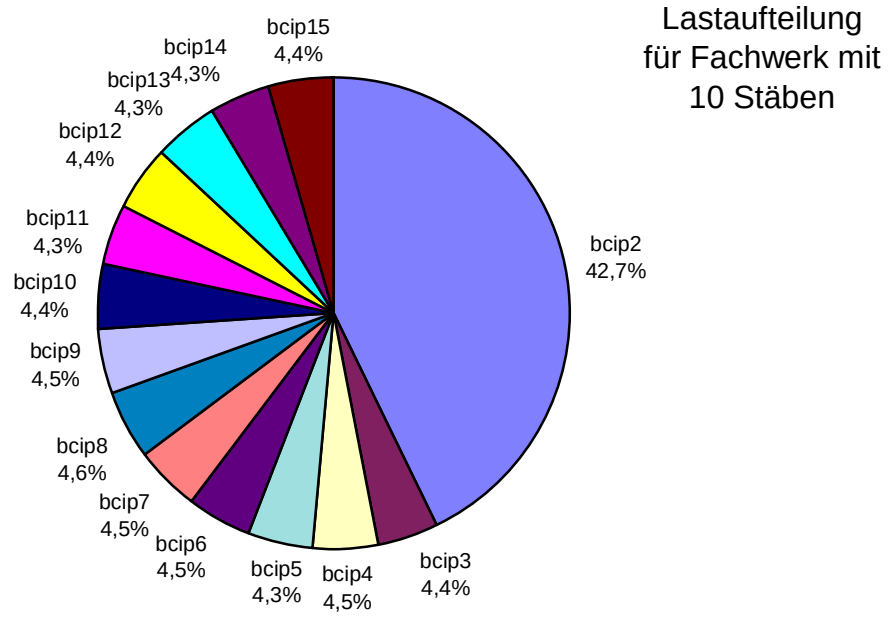


Abbildung 33: Lastaufteilung für 10-Stäbe Fachwerk

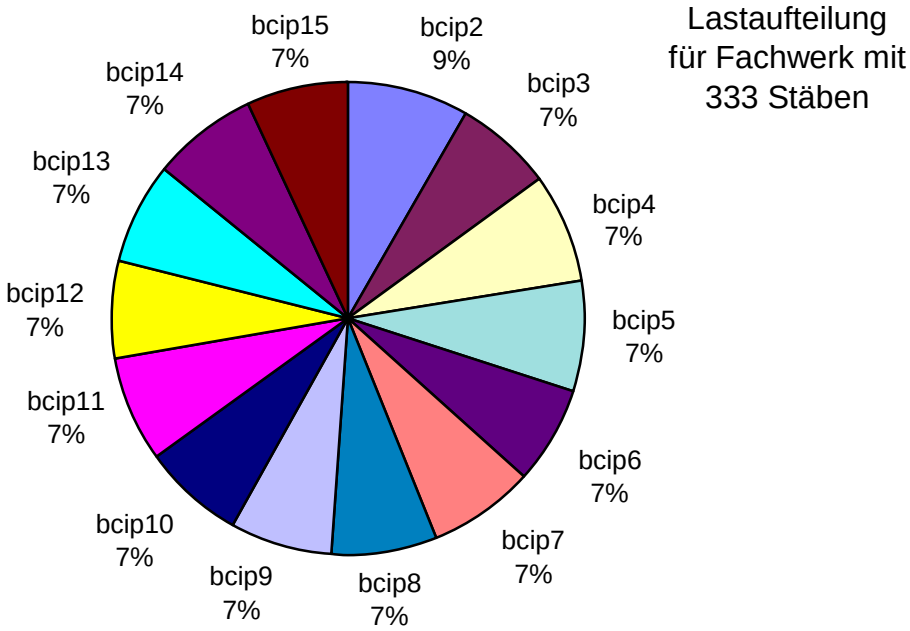


Abbildung 34: Lastaufteilung für 333-Stäbe Fachwerk

7.1.5 Speedup

Desweiteren werden die Dauer der Berechnung für Konfigurationen mit 1 - 14 Rechnern verglichen. Neben der Gesamtzeit wird zusätzlich die Zeit für die parallele Berechnung der Fitneßwerte ermittelt, um eine Abschätzung des sequentiellen Anteils des Programms nach Gleichung (10) zu erhalten. Bei der Bestimmung des Speedups nach Gleichung (6) wird die Berechnungszeit des sequentiellen Programms durch die Berechnungszeit des Programms mit einem Slave auf dem Master-Rechner angenähert. Da für diesen speziellen Slave die Kommunikationszeiten extrem niedrig sind (vgl. 7.1.4), ist eine relativ geringe Abweichung zu erwarten, auch wenn ein rein sequentielles Programm im Vergleich hierzu effizienter ist. Nach dem Amdahlschen Gesetz nach Gleichung (13) wird der maximal erreichbare Wert des Speedups ohne Kommunikationszeiten bestimmt. Die Differenz zwischen diesem und dem tatsächlich erreichten Wert ist damit durch die Kommunikationsdauer bedingt. Um abzuschätzen, wie viel Rechenzeit der Berechnungsalgorithmus im Vergleich zum Gesamtprogramm verbraucht, wird ein Programmlauf mit 50 und 250 zu berechnenden Generationen gemessen.

Zwischen dem 10 Stäbe Fachwerk und dem 333 Stäbe Fachwerk sind aufgrund der unterschiedlichen Berechnungsgrößen der Einzelprobleme Unterschiede in der Granularität gegeben. Auswirkungen auf die Effizienz der Parallelisierung sind unübersehbar (siehe folgende Abbildungen).

Beim 10 Stäbe Fachwerk wird durch die parallele Berechnung sowohl für den kurzen, als auch langen Programmlauf nicht einmal eine Rechenzeitverkürzung erreicht. Der Kommunikationsaufwand macht die Gewinne zunichte. Eine Lastverteilung kann nicht erreicht werden. Für dieses kleine Einzelproblem ist eine solche Parallelisierung nicht nur unnötig, sondern sogar unsinnig.

Für das 333 Stäbe Fachwerk sind diese Aussagen nicht zutreffend und eine Tendez für große Probleme ist absehbar. Aus den Kurven der Berechnungsdauer aus Abbildung 37 ist der annähernd konstant bleibende sequentielle Anteil an der Gesamtberechnungsdauer, die deutlich absinkt, erkennbar. Beim Speedup ist dieser sequentielle Anteil in den Grenzwerten nach Amdahl erfaßt. Ein linearer Speedup kann nur bei rein parallelen Programmen erreicht werden (siehe Abbildung 4). Verglichen mit diesen Grenzwerten ist der tatsächlich erreichte Speedup zufriedenstellend. Insbesondere wenn man das langsame Netzwerk, auf dem die Kommunikation aufbaut, in Betracht zieht.

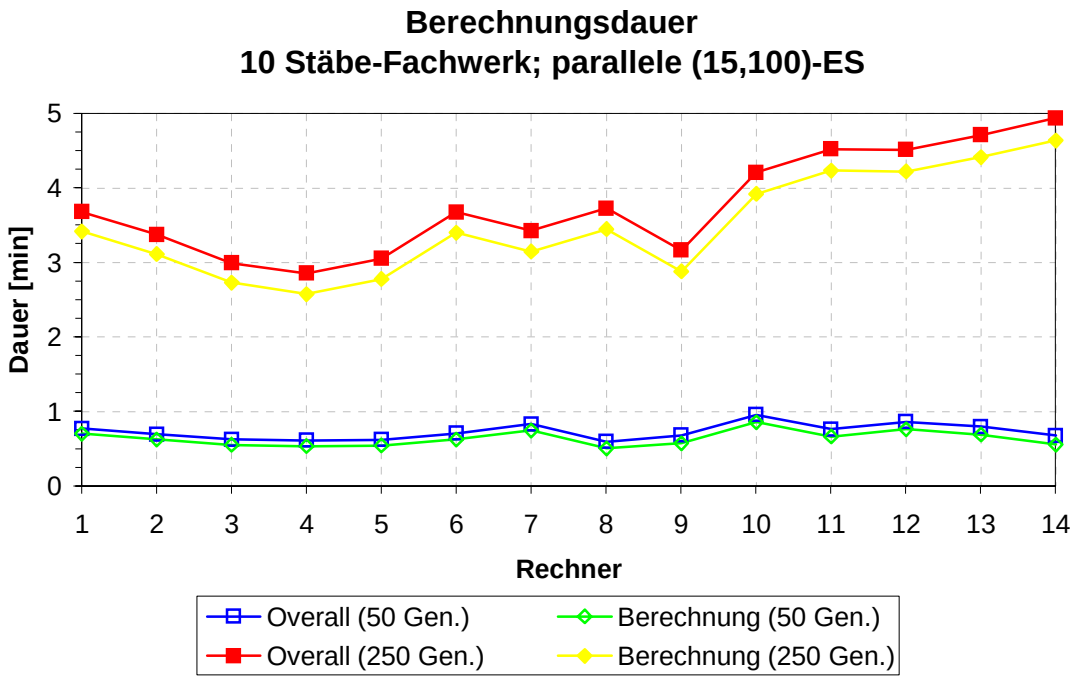


Abbildung 35: Berechnungsdauer - 10 Stäbe Fachwerk

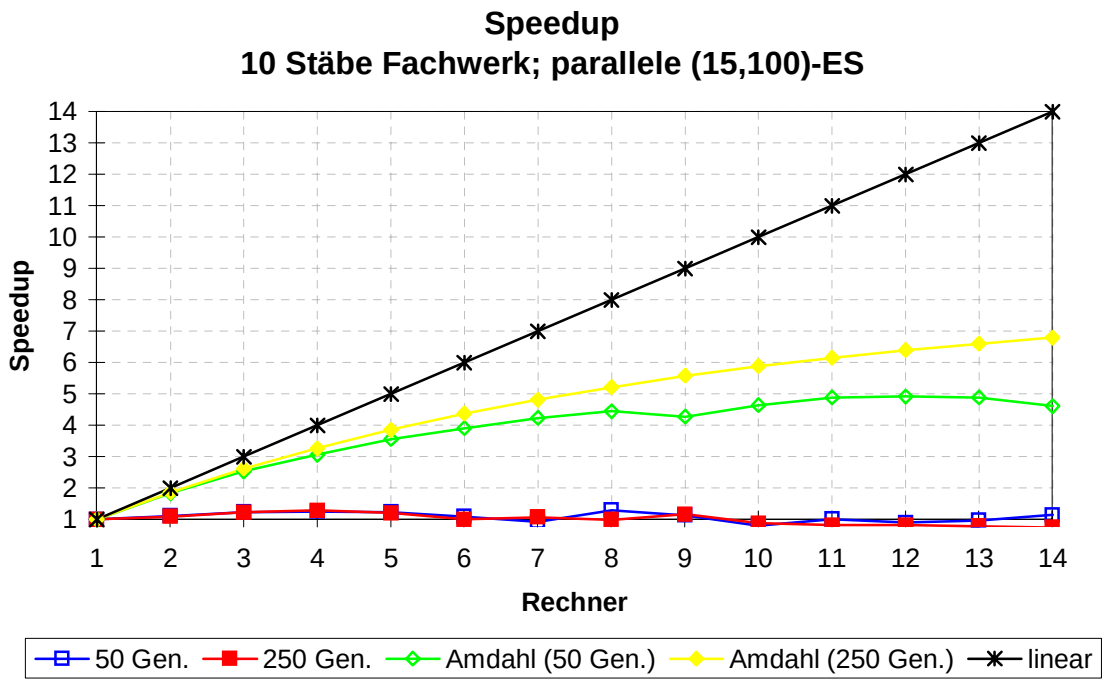


Abbildung 36: Speedup - 10 Stäbe Fachwerk

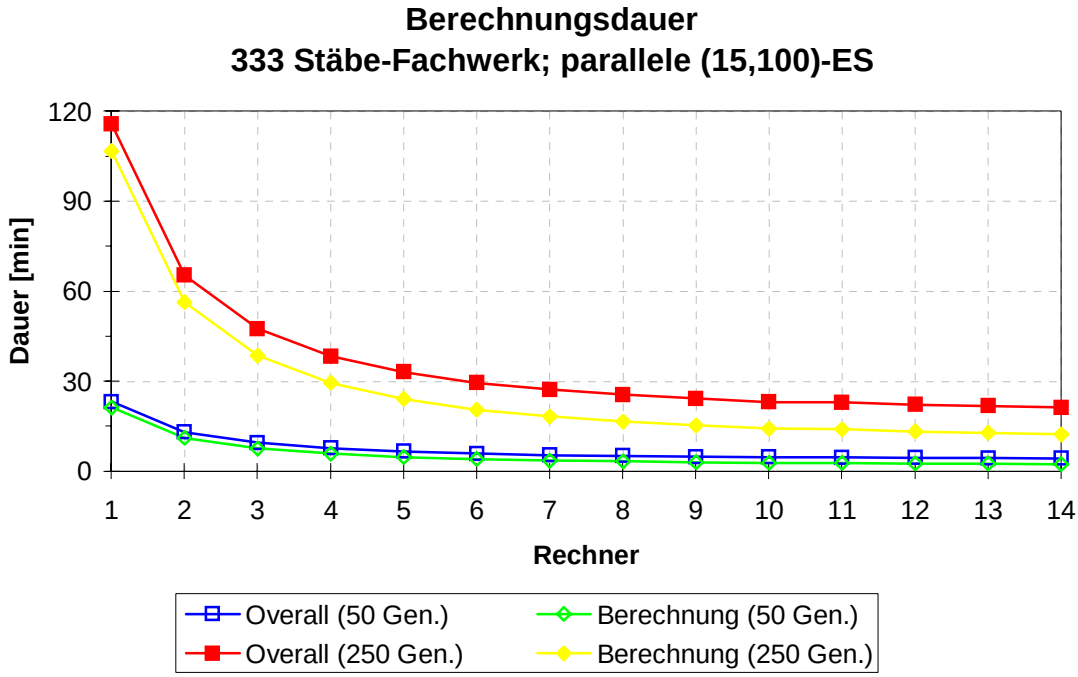


Abbildung 37: Berechnungsdauer - 333 Stäbe Fachwerk

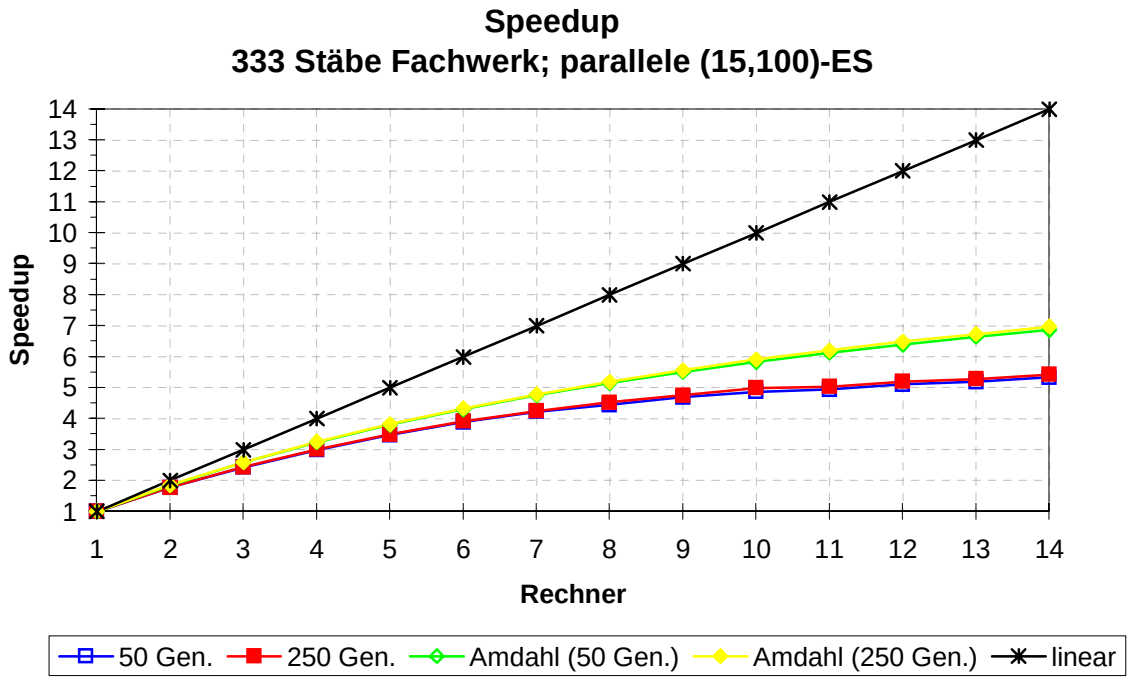


Abbildung 38: Speedup - 333 Stäbe Fachwerk

7.2 Konkretes Beispiel

7.2.1 Problemstellung

Als Beispiel einer diskreten Strukturoptimierung dient ein Fachwerk mit zehn Stäben, bei dem die Querschnitte der einzelnen Stäbe optimiert werden. Das Beispiel wurde bereits von Galante [19] mit einem genetischen Algorithmus berechnet und mit mehreren anderen Lösungsvorschlägen verglichen. Um auf diese Ergebnisse referenzieren zu können wurde auf eine Umrechnung der Daten in das metrische System mit SI-Einheiten verzichtet. Das Fachwerk ist in Abbildung 39 dargestellt:

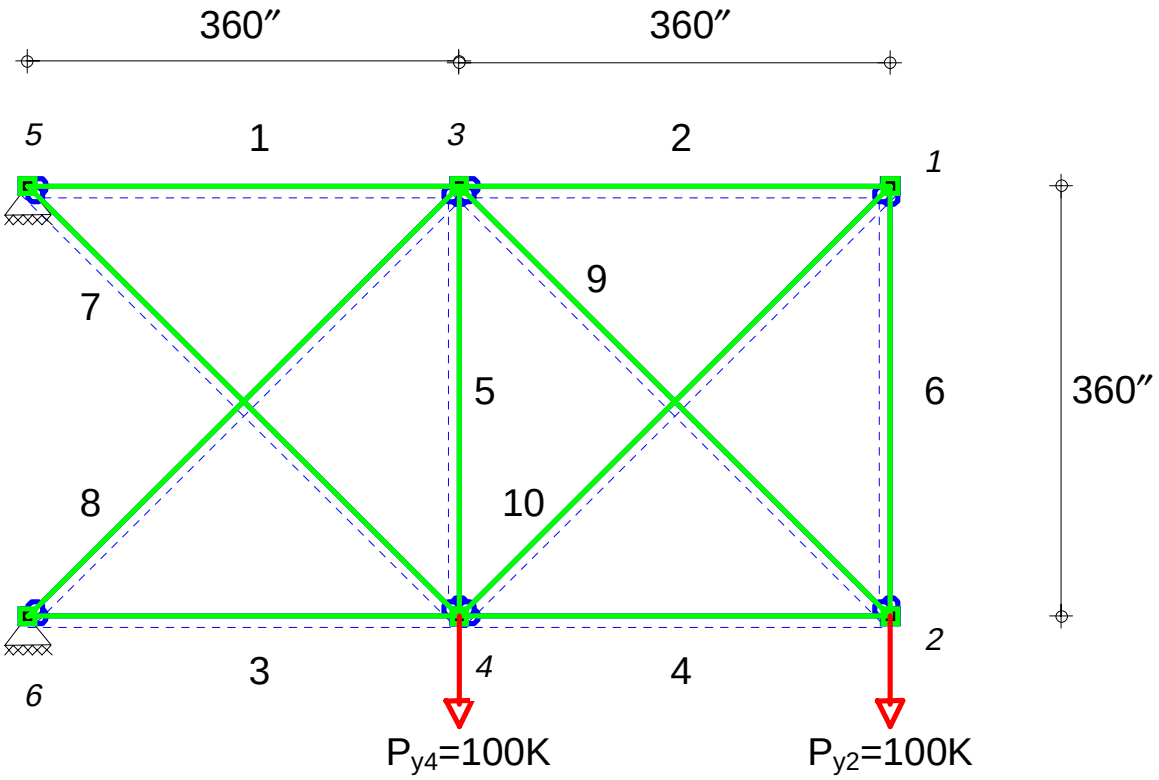


Abbildung 39: Zu optimierendes Fachwerk

Elastizitäts-Modul [Ksi]:	10^4
Spezifisches Gewicht [lb/in^3]:	0,1
Maximale Normalspannung [Ksi]:	25
Maximale Durchbiegung [in]:	2

Tabelle 12: Materialdaten und Nebenbedingungen

Die Querschnitte für die Fachwerkselemente sind aus 42 vorgegebenen Profilquerschnitten nach Abbildung 40 zu wählen. Die Zuordnung der Querschnittsnummer zum Querschnitt bedeutet im Vergleich zur Funktionsoptimierung (vgl. 6.9) eine zusätzliche Abbildung. Die Querschnittswerte sind der Größe nach geordnet. Dies ist Voraussetzung für eine geordnete Suche. Wie man sieht ist eine lineare Zuordnung nicht nötig.

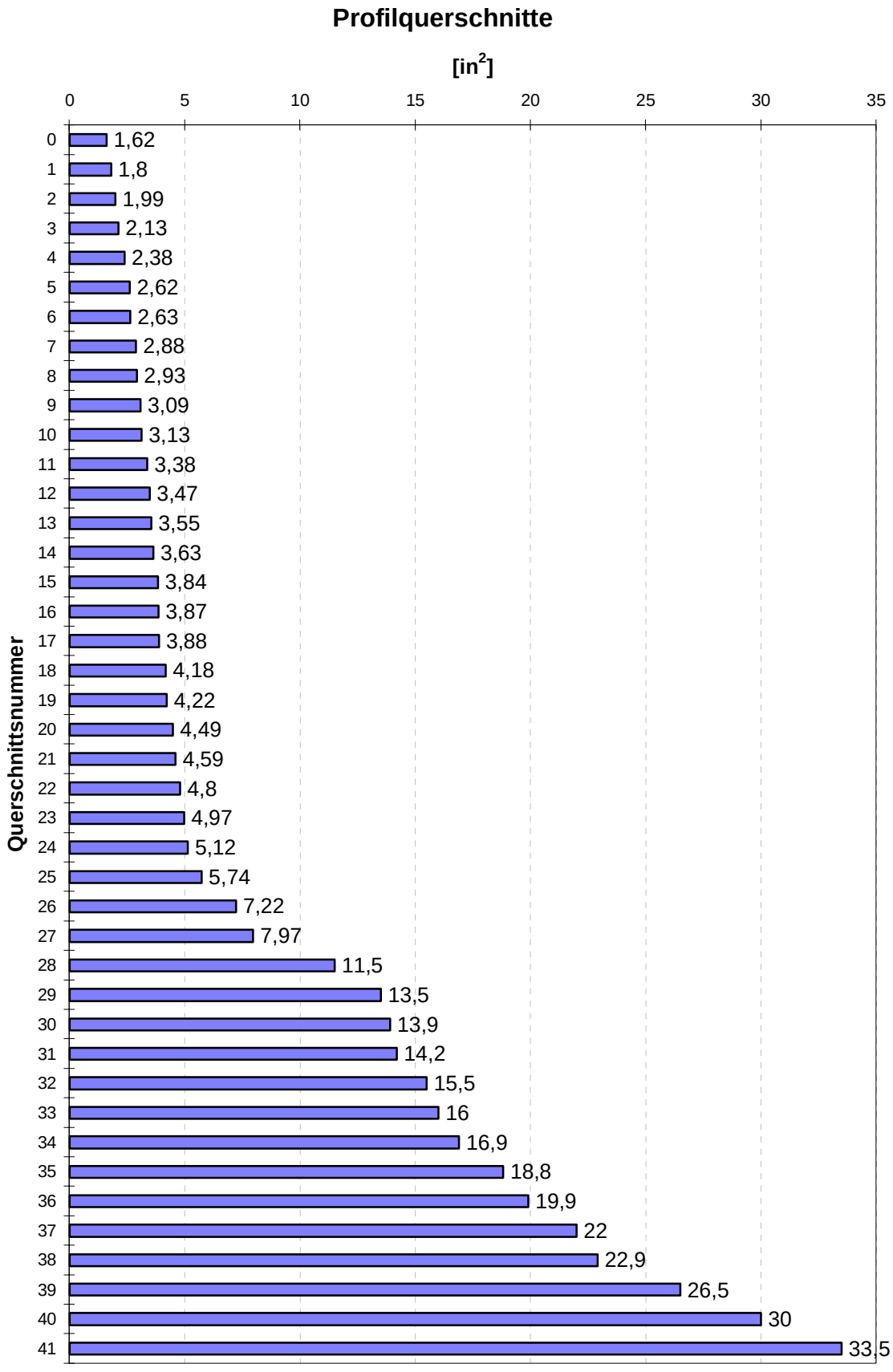


Abbildung 40: Profilquerschnitte

7.2.2 Resultate

Das Problem wird zunächst mit einer (15+100)-ES mit globaler Standardabweichung nach Kapitel 6.6.3 angegangen. Die Rekombination der Individuen erfolgt mit einem einfachen Crossover. Betrachtet man den Verlauf der Fitneß des besten Elter nach Abbildung 41, er-

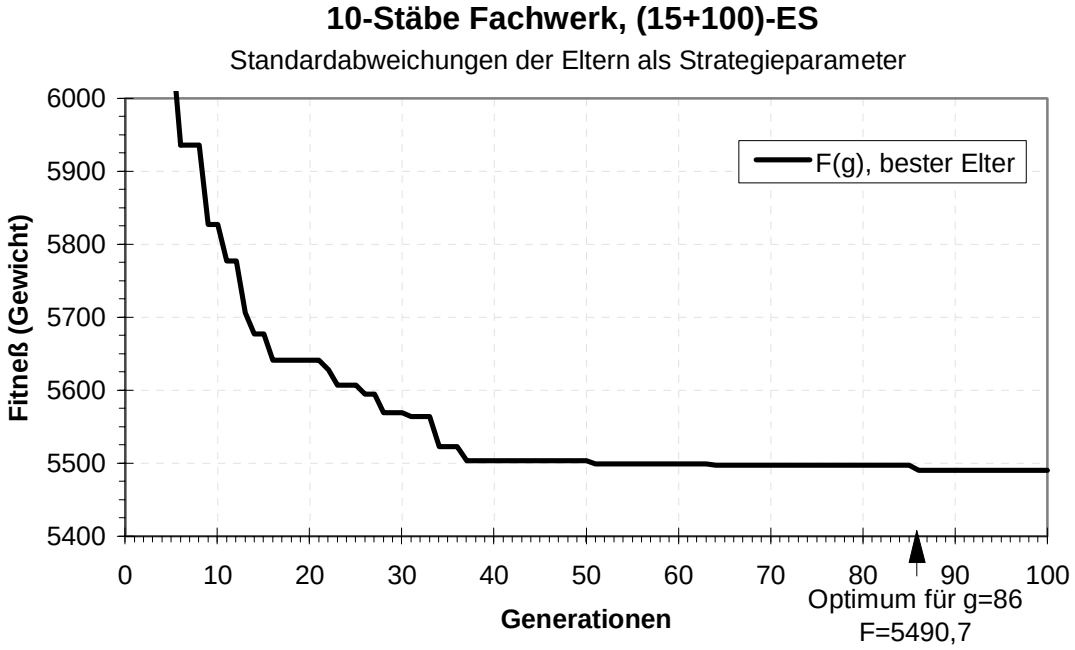


Abbildung 41: Fitneß des besten Elter bei (15+100)-ES

kennt man die anfänglich schnelle Konvergenz. Für weitere Verbesserungen benötigt der Algorithmus allerdings verhältnismäßig viele Schritte. Dies deutet darauf hin, daß die Strategieparameter in diesem Stadium nicht richtig angepaßt sind. Schon nach der 37. Generation ist der Fitneßfunktionswert des besten Elter mit 55037 schon nahe am Optimum. Aber erst nach 86 Generationen ist das Optimum mit einem Gesamtgewicht von 5490,7 erreicht.

Die entsprechenden dem besten Elter zugeordneten Querschnitte sind in Abbildung 42 ersichtlich:

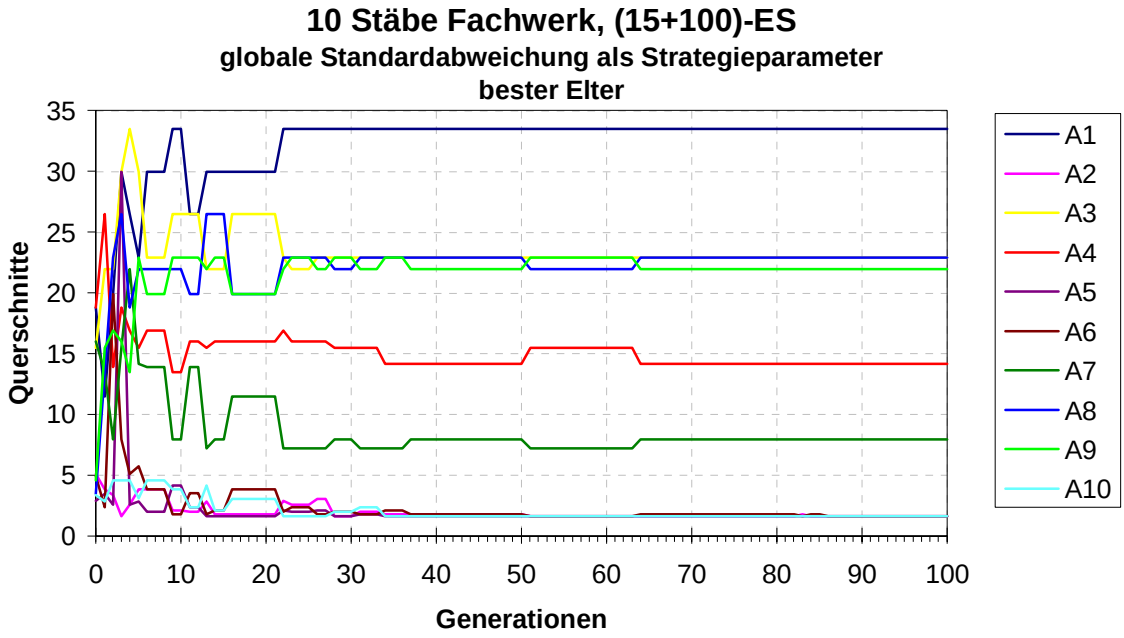


Abbildung 42: Querschnittsverteilung des besten Elter bei (15+100)-ES

Nach einer kurzen „Orientierungsphase“ von ca. 5 Generationen ist die Größenordnung der einzelnen Stabquerschnitte schon relativ gut getroffen. Ab ca. 25 Generationen wird nur noch um eine Querschnittsnummer variiert.
Dem wird eine (15,100)-ES mit n Standardabweichungen als Strategieparameter nach 6.6.2 gegenübergestellt. Während die Rekombination der Parameter mit einem einfachen Crossover durchgeführt wird, werden die Standardabweichungen intermediär rekombiniert.

Die Annäherung des besten Elter an das Optimum ist inAbbildung 43 dargestellt:

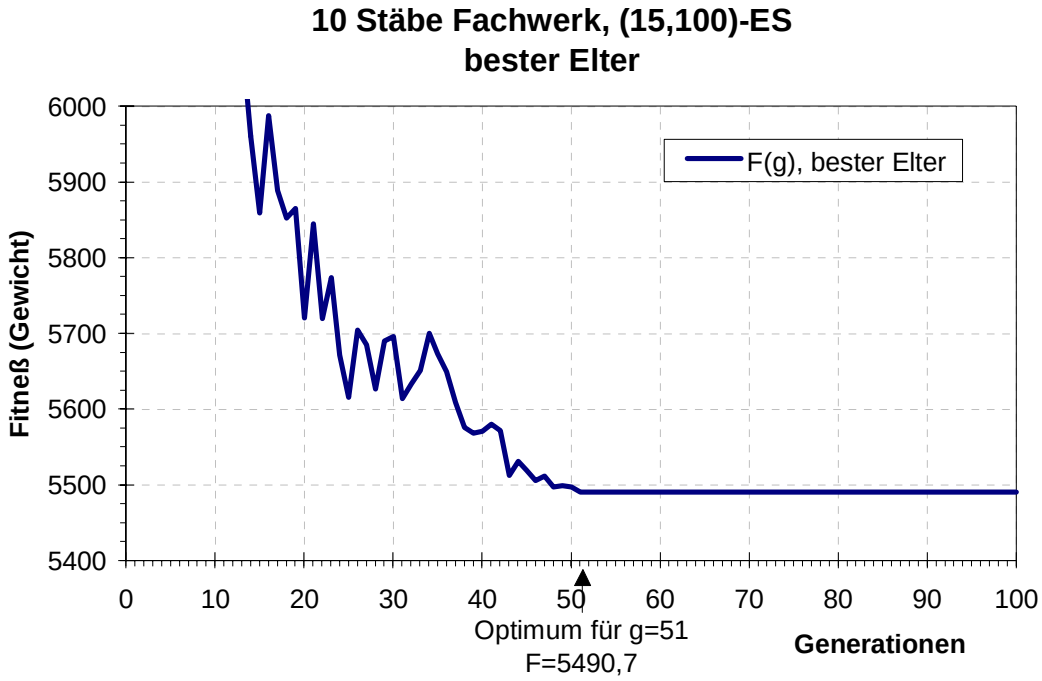


Abbildung 43: Fitneß des besten Elter bei (15,100)-ES

Mit 51 Generationen, die zum Erreichen des Optimums benötigt wurden, ist die Konvergenz schneller als bei der zuvor vorgestellten (15+100)-ES. Dabei werden für den besten Elter auch Rückschritte in Kauf genommen. Diese Rückschritte kommen dann vor, wenn keines der Kinder die Qualität des besten Elter erreicht. Sie sagen nichts über die Gesamtgüte aller Eltern der neuen Generation aus. Es kann vorteilhaft sein, daß ein Elter mit guten Eigenschaften, aber schlechten Fortpflanzungseigenschaften zugunsten einer besseren Weiterentwicklung stirbt.

Die dem besten Elter zugeordneten Querschnitte in Abbildung 44 variieren stärker, da dieser nur für eine Generation existiert. Eine gleichbleibende Verteilung ist erreicht, wenn die Eltern Kinder erzeugen, von denen keines besser ist aber mindestens eines die gleiche Qualtität besitzt wie der beste Elter.

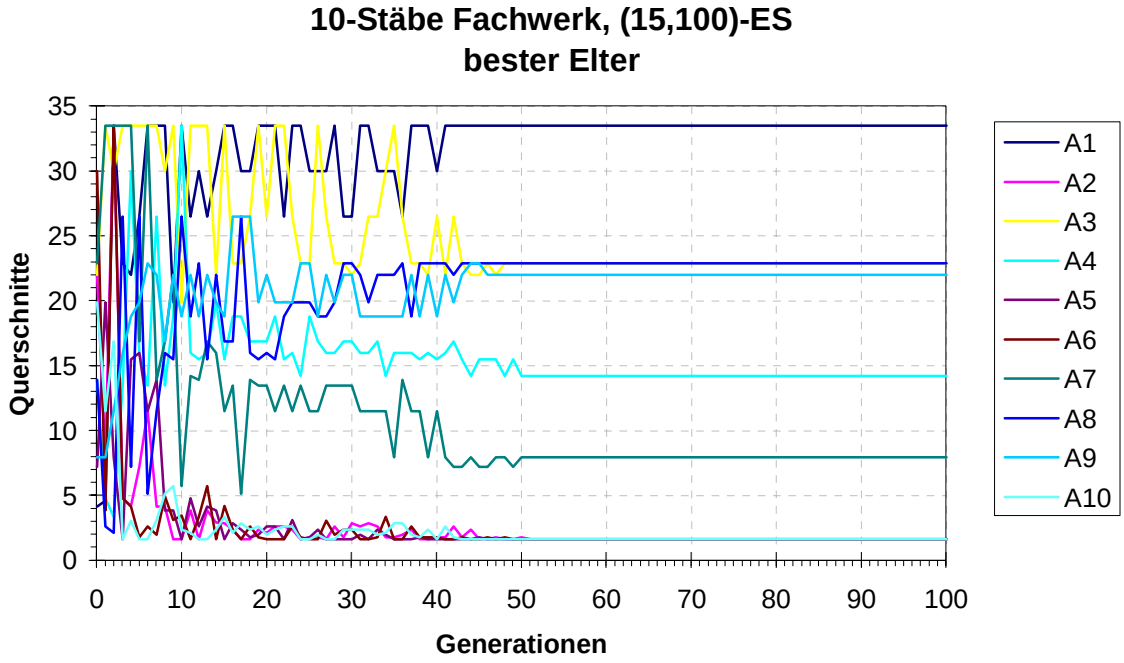


Abbildung 44: Querschnittsverteilung des besten Elter bei (15,100)-ES

Die Selbstanpassung und das Lernen der Individuen findet man auch in ihren Fortpflanzungseigenschaften, den Standardabweichungen, die als Strategieparameter dienen:

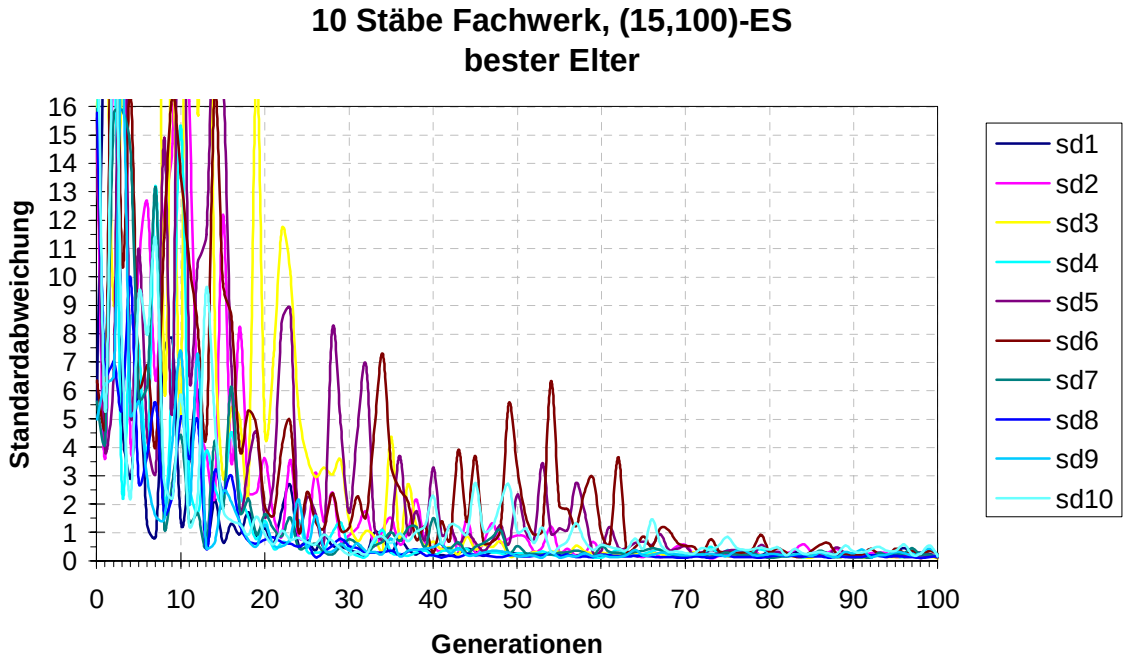


Abbildung 45: Strategieparameter für Eltern

Große Änderungen bringen späteren Generationen keine Vorteile, also sterben sie aus und der Suchradius nahe des Optimums schnürt sich ein.

Auch bei bei der Komma-Strategie kann die Betrachtung der globalen Standardabweichungen der Eltern Aufschluß über das Konvergenzverhalten geben. In Abbildung 46 ist ersichtlich, daß sie gegen Null streben:

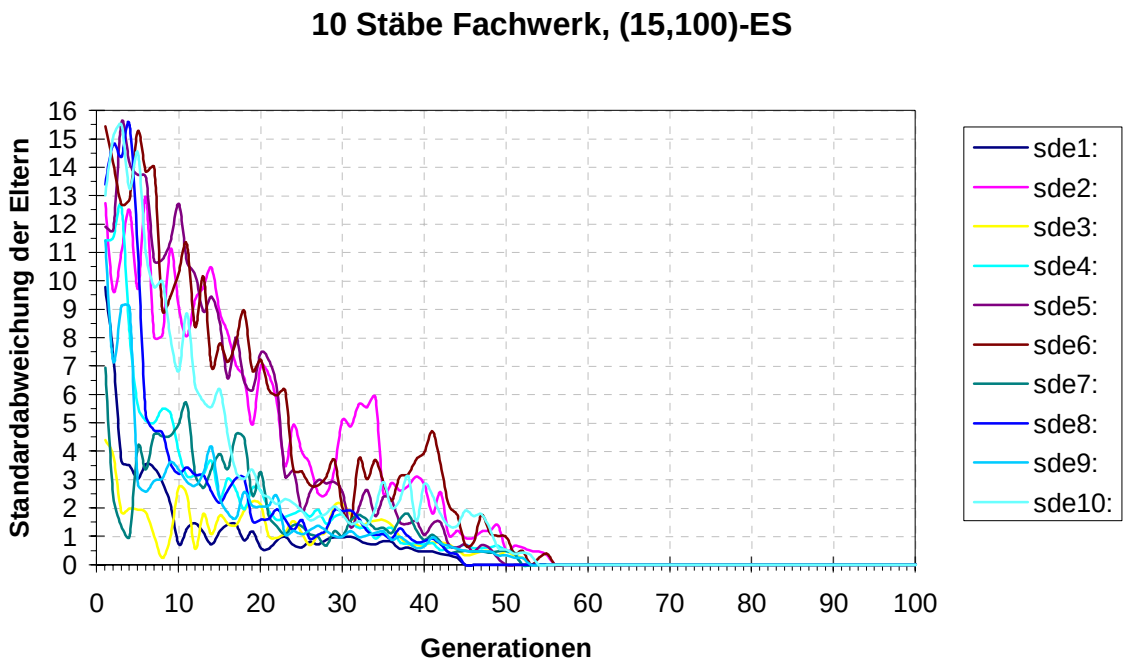


Abbildung 46: Standardabweichung der Eltern bei der Kommastrategie

Das bedeutet, daß sich die Eltern immer mehr angleichen und der Genpool verarmt. Eine Standardabweichung von Null zeigt an, daß alle Eltern das gleiche Merkmal besitzen. Das Optimum wird zu einem Zeitpunkt erreicht, an dem auch die globale Standardabweichungen der Eltern nahe Null liegen. Kurz darauf ist für identische Eltern eine weitere Suche sinnlos.

Als Ergebnis erhält man eine Querschnittsverteilung nach Tabelle 13:

Stab-Nr.:	1	2	3	4	5	6	7	8	9	10
Querschnitt:	33,5	1,62	22,9	14,2	1,62	1,62	7,97	22,9	22,0	1,62

Tabelle 13: Optimierte Querschnitte

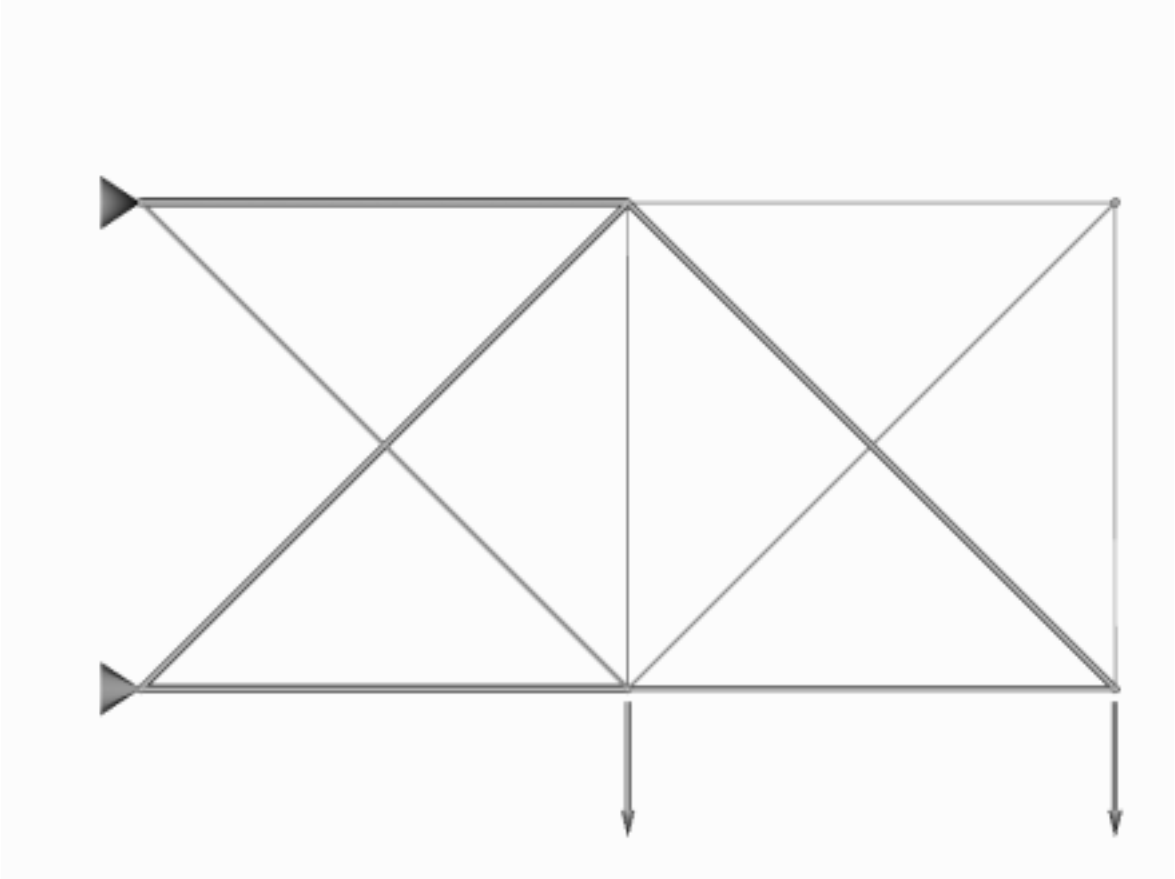


Abbildung 47: Optimiertes Fachwerk

8 Schlußbemerkung

In der Arbeit wurden parallele Programmiertechniken auf diskrete Probleme der Strukturoptimierung angewendet. Parallele Programmierprinzipien erfordern angepaßte Algorithmen. Für die Programmentwicklung und die Beurteilung eines parallelen Programms ist die Kenntnis der speziellen Programmiertechniken und -konstrukte ebenso notwendig wie das Verständnis des Aufbaus der verwendeten Bibliotheken und der Funktionsweise des Netzwerks. Netzwerke sind bei Workstationcluster ein Flaschenhals. Deshalb muß darauf geachtet werden, möglichst wenig zu kommunizieren und eine grobgranulare Prozeßverteilung sicherzustellen. Nicht nur die vergleichsweise niedrigen Übertragungsraten, sondern insbesondere die hohen Latenzzeiten wirken sich negativ auf die Leistung aus. Ein besonderes Problem stellt bei Workstationcluster im nicht-dedizierten Parallelbetrieb die schwankende und unbestimmbare Systemauslastung dar. Unter diesen Voraussetzungen ist es unmöglich die Einzelprozesse statisch optimal zu verteilen. Ein dynamischer Lastausgleich hingegen ist sehr aufwendig, so daß auf suboptimale statische Lastausgleichsverfahren zurückgegriffen wird. Aufgrund des geringen Zusatzaufwands bezüglich Hard- und Software ist ein Einstieg in diese Technologie bei Workstationcluster lohnenswert. Bei Problemen, die sich für eine Parallelisierung in diesem Umfeld eignen, sind in dieser Arbeit bereits Erfolge erzielt worden.

Die diskrete Strukturoptimierung verwendet für die Entwurfsvariablen nur bestimmte, diskrete Werte. Die Zielfunktionen, die sich daraus ergeben, sind nicht differenzierbar und unstetig. Besonders die nicht-deterministischen Optimierungsalgorithmen erweisen sich hier als brauchbar. Für eine zielgerichtete Suche ist es nötig, eine Art Gedächtnis aufzubauen, um Ergebnisse vorangegangener Versuche in die weitere Suche indirekt einfließen zu lassen.

Die Evolutionsstrategie erweist sich als ein Optimierungsverfahren, das auch für diskrete Entwurfsvariablen und die angesprochenen Probleme verwendbar ist. Die Selbstanpassung bewirkt gute Konvergenzeigenschaften. Durch die Wahl der einstellbaren Kontrollparameter kann die Konvergenz noch weiter verbessert werden.
Die ES eignet sich auch im besonderen Maße für die parallele Programmierung. Dabei sind je nach Problemstellung verschiedene Stufen möglich.

- Bei sehr großen Systemen macht eine statische Analyse Sinn, die auf parallelen Gleichungslösern aufbaut. Hierfür kann ScaLAPACK als Grundlage dienen.
- Bei sequentieller statischer Analyse können bei der ES für ausreichend umfangreiche, grobkörnige Einzelberechnungen diese mit einem Master-Slave Prinzip auf einzelne Rechner verteilt werden.
- Kleine feinkörnige Berechnungen sind für MIMD-Systeme ungeeignet. Die ES kann aber auch für diesen Fall als Multipopulations-ES effizient eingesetzt werden (vgl. [20]). Für

Evolutionsstrategien mit einer Population können bei großen Populationen mehrere Berechnungen als Block zusammengefaßt werden. Um eine grobe Granularität zu erreichen fassen beide Ansätze viele kleine Berechnungen zu einer großen zusammen.

Jede Problemstellung verlangt nach einer angepaßten parallelen Umsetzung. Ansonsten wird mit Mehraufwand auf mehreren Rechnern das gerechnet, was im sequentiellen Fall auf einem Rechner in gleicher Zeit möglich ist. Die Parallelisierung sollte kein Selbstzweck sein! Die Entwicklung paralleler Programme bedingt einen Mehraufwand. Die Effizienz eines parallelen Programms wird durch die Prozeßverwaltung und Kommunikation herabgesetzt. Die Parallelisierung ist bei sehr rechenintensiven Anwendungen gerechtfertigt, die bei sequentieller Ausführung entweder nicht mehr berechenbar sind oder nur in einem unverhältnismäßig hohen Zeitaufwand gelöst werden können. Hier arbeiten die Programme meist auch effizienter. Bei bedachter Anwendung eröffnet sich ein Gebiet mit neuen Möglichkeiten, insbesondere auch auf dem Gebiet der Strukturoptimierung.

9 Literaturverzeichnis

[1] Bletzinger, Kai-Uwe:
Formoptimierung von Flächentragwerken. Dissertation, Bericht Nr. 11/1990. Institut für Baustatik, Universität Stuttgart, 1990

[2] Bräunl, Thomas:
Parallele Programmierung. Braunschweig, Wiesbaden: Vieweg, 1993

[3] Geiger, Alfred:
Parallelrechner - Architektur und Anwendung. Rev. 1.2, Rechenzentrum, Universität Stuttgart, 1995

[4] Flynn, M.:
Very High Speed Computing Systems, Proceedings of the IEEE, vol. 54, pp. 1901-1909, 1966

[5] Amdahl G.:
Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities, AFIPS Conference Proceedings, vol. 30, pp. 483-485, Atlantic City NJ, Apr. 1967

[6] Gustafson, J.:
Reevaluating Amdahl's Law, Communications of the ACM, Technical Note, vol. 31, no. 5, 532-533, Mai 1988

[7] Eberhard, Peter:
Zur Mehrkriterienoptimierung von Mehrkörpersystemen. Dissertation, Institut B für Mechanik, Universität Stuttgart, 1995

[8] Rothermel, Kurt:
Rechnernetze. Vorlesungsmanuskript. Institut für parallele und verteilte Höchstleistungsrechner, Universität Stuttgart, 1993

[9] Rothermel, Kurt:
Verteilte Systeme. Vorlesungsmanuskript. Institut für parallele und verteilte Höchstleistungsrechner, Universität Stuttgart, 1996

[10] Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; Sunderam, V.:
PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing. Cambridge: MIT Press, 1994

[11] Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; Sunderam, V.:
PVM 3 User's Guide and Reference Manual. Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1994

[12] N. N.:
ScaLAPACK Users' Guide, updated very preliminary draft, May 22, 1996

[13] Beguelin, A.; Dongarra, J.; Geist, A.; Manchek, R.; Sunderam, V.:
Recent Enhancements to PVM. International Journal for Supercomputer Applications, Vol. 9, No. 2, Summer 1995.

[14] Überhuber, Christoph:
Computer-Numerik. Berlin, Heidelberg: Springer, 1995

[15] Rechenberg, Ingo:
Evolutionsstrategie. Stuttgart: Frommann-Holzboog, 1973

[16] Schwefel, Hans-Paul:
Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Basel, Stuttgart: Birkhäuser, 1977

[17] Bäck, Thomas; Schwefel, Hans-Paul:
Evolution Strategies I: Variants and their computational implementation. In: Genetic Algorithms in Engineering and Computer Science, S. 111-126. Chichester: John Wiley & Sons, 1995

[18] Schöneburg, E.; Heinzmann, F.; Feddersen, S.:
Genetische Algorithmen und Evolutionsstrategien. Bonn: Addison-Wesley, 1994

[19] Galante, M.:
Genetic Algorithms as an Approach to Optimize Real-world Trusses, Int. j. numer. methods eng., vol. 39, 361-382 (1996)

[20] Cai, J; Thierauf, G.:
A parallel evolution strategy for solving discrete structural optimization, In: Advances in parallel and vector processing for structural mechanics, S. 239-244. Edinburgh: Civil-Comp Press, 1994

10 Indexverzeichnis

N

O

P

R

S

T

U

V

Z

11 Abbildungs- und Tabellenverzeichnis

Anhang

Parallele Integration

Das PVM-Programm zur parallelen Integration mit der Trapezregel dient der Demonstration der Programmierung mit PVM:

Definitions-Headerdatei:

```
/* trapez1_msgdef.h */
/* Martin Bernreuther, July 1996 */

#define SLAVEPRGNAME "trapez1_slave"

#define M2S_Startup 0
#define S2M_Result 1
```

Masterprogramm:

```
/* trapez1.c */
/* Martin Bernreuther, July 1996 */
/* Calculate the integral of a function */

#include <stdio.h>          /* using printf    */
#include "pvm3.h"           /* using PVM-calls */
#include "trapez1_msgdef.h" /* MessageID definition */

/* MAIN
     ============================================================================*/
 int main(void)
 {
   double x1,x2; /* integration interval */
   int    nproc; /* number of processes,slaves   */
   int    np;    /* subdivisions for slave */
   double wp;        /* integration interval width for slave */
   double x1p,x2p;   /* integration interval for slave */
   double pResult;   /* Slave Result */
   double SumIntegr; /* Final Result */
   int    iproc;     /* loop variable */
   int    mytid,ntask,slave_tid,info,bufid; /* PVM variables */

   mytid=pvm_mytid(); /* determine own task identifier, start PVM */

/* Initialize variables */
/*-------------------------------------------------------*/
   /* integration interval [x1,x2] */
   x1   =  0.0;
   x2   =  2.57079632679489661922;
   nproc=   10;      /* number of slave processes */
   np   =   50;      /* subdivisions for slave process */
/*-------------------------------------------------------*/
   wp=(x2-x1)/nproc;  /* integration interval width for slave process */

/* Create slave processes and send startup data ----------------------*/
   for (iproc=1;iproc<=nproc;iproc++)
   {
     /* calculating integration  */
     x1p = x1+(iproc-1)*wp;   /* slave integration interval */
     x2p = x1p+wp;            /*          [x1p,x2p]         */

     /* start slave task */
     ntask = pvm_spawn(SLAVEPRGNAME,(char**)0,PvmTaskDefault,"",1,&slave_tid);
```

```
     (void) printf ("%d.process: x1p=%f; x2p=%f; %d
     subdiv.\n",iproc,x1p,x2p,np);
     /* initialize&pack sendbuffer */
     info=pvm_initsend(PvmDataDefault);
      info=pvm_pkdouble(&x1p,1,1);
      info=pvm_pkdouble(&x2p,1,1);
      info=pvm_pkint(&np,1,1);
     /* send message M2S_Startup to slave_tid */
     info=pvm_send(slave_tid,M2S_Startup);
     (void) printf ("process sent to t%d\n",slave_tid);
   }

/* Receive Results & calculate final result */
   SumIntegr = 0.0; /* Sum up final result */
   for (iproc=1;iproc<=nproc;iproc++)
   {
     bufid=pvm_recv(-1,S2M_Result);         /* blocking receive */
      info=pvm_upkdouble(&pResult,1,1);     /* unpack slave result */
     (void) printf ("Result of process: %f (%d more)\n",pResult,nproc-iproc);
     SumIntegr += pResult;
   }
   (void) printf ("\nIntegral: %f\n",SumIntegr);

/* Exit PVM & stop program */
   pvm_exit();
   return 0;
}
```

Slave-Programm:

```
/* trapez1_slave.c */
/* Martin Bernreuther, July 1996 */

#include <math.h>               /* using sin()          */
#include "pvm3.h"               /* using PVM functions  */
#include "trapez1_msgdef.h"     /* MessageID definition */

double function(double x);


/* Definition of the function */
#define PI 3.14159265358979323844
double function(double x)
{
  double fx; /* function return value */
  if (x<=PI/2.) fx=sin(x);
  else          fx=1.-(x-PI/2.)*(x-PI/2.);
  return fx;
}
/* MAIN ============================================================*/
int main(void)
{
  double x1,x2;     /* integration interval */
  int subdiv;       /* subdivisions */
  double dx;           /* delta x */
  double SumIntg;      /* sum up function evaluations */
  int i;               /* loop variable */
  double x;            /* actual x */
  double Result;       /* final result */
  int ptid,bufid,info; /* PVM data */

  ptid=pvm_parent(); /* determine parent task identifier (master) */

  /* receive startup data from master */
  bufid=pvm_recv(ptid,M2S_Startup);
   info=pvm_upkdouble(&x1,1,1);
   info=pvm_upkdouble(&x2,1,1);
   info=pvm_upkint(&subdiv,1,1);

  dx=(x2-x1)/subdiv;
```

```
  /* Integration */
  SumIntg=(function(x1)+function(x2))/2.;
  x=x1;
  for (i=0;i<subdiv-1;i++)
  {
    x+=dx;
    SumIntg+=function(x);
  }
  Result=SumIntg*dx;
  /* Pack&send result to master */
  pvm_initsend(PvmDataDefault);
   pvm_pkdouble(&Result,1,1);
  pvm_send(ptid,S2M_Result);

  /* exit PVM */
  pvm_exit();
  return 0;
}
```