Jül-4298

Jülich Supercomputing Centre (JSC)

Konzeption eines Softwaresystems zur Prognose von Benchmarking-Ergebnissen paralleler Programme im Scientific Computing

Stefanie Meier

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Konzeption eines Softwaresystems zur Prognose von Benchmarking-Ergebnissen paralleler Programme im Scientific Computing

Stefanie Meier

Berichte des Forschungszentrums Jülich; 4298
ISSN 0944-2952
Jülich Supercomputing Centre (JSC)
Jül-4298

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL) unter
http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Konzeption eines Softwaresystems zur Prognose von Benchmarking-Ergebnissen paralleler Programme im Scientific Computing

Das Scientific Computing bildet in der modernen Wissenschaft die Schnittstelle zwischen Theorie und Experiment. Es werden dabei große Simulationsrechnungen auf Supercomputern ausgeführt. Daher ist es sowohl für den Wissenschaftler wie auch für Rechenzentrumsbetreiber entscheidend, wie sich eine Anwendung auf einem bestimmten Rechner verhält. Insbesondere ist bei einer Neuanschaffung eines Supercomputers wichtig, abschätzen zu können, wie die Performance dieses Rechners aussieht. Zur Ausmessung eines Rechners werden Benchmark-Suiten genutzt. Für eine Neuanschaffung kann eine Messung aber noch nicht vorgenommen werden, daher ist man an einer Prognose interessiert.
In dieser Arbeit wird das Benchmarking im Kontext des Supercomputing betrachtet und die Anfragen, die an das Benchmarking gestellt werden, untersucht. Es werden Ansätze der Benchmarking-Prognose besprochen, wobei besonders die Möglichkeiten der Künstlichen Intelligenz vielversprechend erscheinen und näher betrachtet werden.
Im Detail werden Expertensysteme mit Ontologien sowie Künstliche Neuronale Netze beschrieben und verglichen. Nach einer Entscheidung für eine der beiden Methoden, wird dieses Verfahren in einer ausführlichen Testreihe im Hinblick auf die Benchmarking-Prognose untersucht. Diese Tests werden dann im abschließenden Teil der Arbeit zur Grundlage der Konzeption für ein Softwaresystem, das Prognosen von Anwendungsbenchmarks auf einem neuen Rechnersystem möglich machen soll.

Design of a Software System for the Prediction of Benchmarking Results of Parallel Programs in Scientific Computing

In today's research environment scientific computing acts as an interface between theory and experiment. Large simulations are calculated on supercomputers. Therefore it is quite important for scientists how an application program behaves on a certain computer. But that is also true for the management and administrators of a computer center who want to meet the needs of their users. Especially during a procurement process for a new supercomputer it is fundamental to estimate the performance of that computer. Normally, benchmark suites are used to measure the performance of a supercomputer. But on a future computer these benchmarks cannot be run, so one is interested in having a prediction.
In this master thesis, benchmarking is introduced in the context of supercomputing. Questions which benchmarking should answer are reviewed. Several possibilities of performing a prediction for benchmark results are shown. It seems that Artificial Intelligence is a promising method for this kind of prognosis and is investigated further.
In detail, expert systems together with ontologies and artificial neural networks are taken into account and compared with each other. After a decision for one of these methods, it is used in an extensive test series. In the last part of this thesis the conception of a software system for prediction of application benchmarks on new computer systems is built on the performed tests.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In den letzten Jahren hat sich in der Wissenschaft als Schnittstelle zwischen Theorie und Experiment das *Scientific Computing* etabliert. Es handelt sich dabei um die Simulation von Experimenten am Computer unter Nutzung von mathematischen Modellen als Grundlage.
Zumeist geht es dabei um sehr aufwendige Simulationen, die nicht mehr bzw. nur mit hohem zeitlichen Aufwand mit einem seriellen Computerprogramm gelöst werden können. Daher werden für diese Problemstellungen die Programme parallelisiert, d.h. unabhängige Befehlsfolgen in einem Programm können parallel auf mehreren Prozessoren ausgeführt werden. Die Daten der Programme werden über das Netzwerk, das die Prozessoren verbindet, kommuniziert.
Um solche parallelen Programme ausführen zu können, wird ein entsprechend mit mehreren Prozessoren ausgerüsteter Rechner benötigt. Handelt es sich dabei nicht nur um eine kleine Anzahl von Prozessoren, kommt man in den Bereich des Supercomputing oder *High Performance Computing*.

Das High Performance Computing ist ein Bereich der Informatik, der sich mit der Entwicklung und dem Betrieb von Supercomputern sowie ihrer Programmierung befasst. Supercomputer sind Systeme, die zum Zeitpunkt ihrer Markteinführung die oberste Leistungsklasse bilden.
Die derzeit führenden Supercomputer sind zumeist massiv-parallele Systeme, d.h. sie bestehen aus einer Vielzahl an Prozessoren, die durch ein schnelles Netzwerk miteinander verbunden sind. Ein Beispiel dafür ist die Blue Gene-Reihe von IBM. Hier sind die Prozessoren (beim derzeit führenden System sind das über 200000) sowohl durch ein 3D-Torus-Netzwerk als auch durch ein hierarchisches Netzwerk verbunden. Die einzelnen Prozessoren sind dabei, um eine hohe Packungsdichte zu erreichen, im Vergleich mit den aktuellen Möglichkeiten des Marktes gar nicht so schnell, die Leistung wird aber durch die hohe Parallelität und schnelle Vernetzung, sowie die gute Balance zwischen Prozessoren und Speicher erreicht.
Neben den beschriebenen massiv-parallelen Systemen finden sich auch Vektorrechner in der oberen Leistungsklasse. Auch diese Systeme arbeiten massiv-parallel, die einzelnen Prozessoren sind dabei aber Vektorprozessoren, die in der Lage sind, eine Berechnung gleichzeitig auf einem Satz von Daten, also einem Vektor, durchzuführen. Beispiele sind hier das neu eingeführte SX9-System von NEC sowie der *Earth Simulator*, ebenfalls von NEC, der über mehrere Jahre hinweg die Rangliste der leistungsstärksten Supercomputer anführte, bevor er von den Blue Gene-Systemen abgelöst wurde.
Die neueste Entwicklung sind hybride Systeme. Auch diese arbeiten parallel, die einzelnen Recheneinheiten bestehen jedoch aus einer Kombination von verschiedenen Arten

an Prozessoren. Der *Roadrunner* von IBM ist das erste große System dieser Art und wird derzeit als weltweit schnellster Rechner bezeichnet. Er besteht aus einer Kombination von Opteron- und Cell-Prozessoren, bei der die Opteron-Prozessoren Datenpakete entgegennehmen, um sie an die Cell-Prozessoren zur schnellen Verarbeitung weiterzugeben.

An diesen Beispielen wird klar, dass die unterschiedlichen Rechner nicht einfach zu vergleichen sind. Wie kann man unter dieser Voraussetzung die mehrfach angesprochene Leistungsfähigkeit überhaupt messen? Wie können diese unterschiedlichen Architekturen miteinander verglichen werden?
Es wird eine einheitliche Maßeinheit benötigt und ein Meßinstrument, um die darauf basierende Größe für den jeweiligen Rechner zu bestimmen.

Im Supercomputing ist die Top500-Liste entwickelt worden, in der die 500 leistungsfähigsten Rechner aufgeführt werden, die zu einem festgelegten Zeitpunkt auf dem Markt und auch im Einsatz sind [W1]. Leistungsfähigkeit bedeutet in diesem Zusammenhang Geschwindigkeit. Die Maßeinheit, mit der die Geschwindigkeit und damit die Position in der Rangliste bestimmt werden, erhält man über den *Linpack-Benchmark*. Es handelt sich dabei um ein Computerprogramm zum parallelen Lösen eines linearen Gleichungssystems. Das Programm wird auf dem auszumessenden System gestartet, die linearen Gleichungssysteme werden gelöst und gleichzeitig errechnet das Programm die Anzahl der Gleitkommaoperationen, die es dafür benötigt. Aus der Laufzeit und dieser Anzahl an Operationen kann dann die Anzahl der Gleitkommaoperationen pro Sekunde (FLOPS - Floating Point Operations per Second) bestimmt werden.
Diese Art der Leistungsbewertung gibt einen Überblick über die aktuelle Supercomputer-Landschaft, sie wird aber durchaus auch kritisiert. Ein Supercomputer ist ein komplexes System und zur tatsächlichen Leistung gehört im Betrieb z.B. auch die Speicherkapazität und -bandbreite, die im Linpack-Programm nicht ausreichend zum Tragen kommt. Zudem handelt es sich beim Linpack nur um ein einzelnes Programm, durch das nicht sicher gestellt werden kann, dass eine Vielfalt von anderen Programmen auf dem mit dem Linpack ausgemessenem Rechnersystem ebenso gut läuft wie dieser Benchmark.

Damit ist bereits ein Einsatzgebiet eines *Benchmarks* deutlich geworden. Es handelt sich hier um den Vergleich von Rechnern.
Man kann sich leicht vorstellen, dass die Meßbarkeit der Leistung von Supercomputern auch in weiteren Bereichen eine große Rolle spielt, zu nennen wäre hier vor allem die Beschaffung eines Supercomputers. Hat der Computer die gewünschte Leistungsfähigkeit? Ist der Computer gut geeignet für die geplanten Anwendungen, also die parallelen Simulationen, die mit ihm bearbeitet werden sollen? Diese zweite Frage bezieht sich also nicht auf reine Geschwindigkeit, sondern auch auf die angesprochene Überprüfung von anderen Kriterien des Rechners, wie z.B. eine gute Speicheranbindung.

Wir sehen also, das dem *Benchmarking* eine wichtige Rolle im Supercomputing zufällt. Es werden Rechner vergleichbar gemacht, indem ihre Leistung bestimmt wird, zudem können auch die Vorzüge oder Nachteile einer bestimmten Rechnerarchitektur heraus-

gestellt werden. Für die verschiedenen Zwecke sind unterschiedliche Rechenprogramme erforderlich, zudem müssen diese Programme ständig an neue Architekturen angepasst und die Messungen immer wieder aktualisiert werden.
Man kann sich vorstellen, dass bei der hohen Geschwindigkeit im Bereich der Computerentwicklung das Benchmarking eine komplexe Aufgabe ist. Mit dieser Aufgabe beschäftigen sich in den Rechenzentren und auch in vielen Projekten weltweit wissenschaftliche Mitarbeiter. Es wird z.B. untersucht, welche Programme als Benchmark dienen können. Zudem werden Rechner ausgemessen. Dabei entstehen große Mengen an Daten, die verglichen werden müssen und aus denen oft wichtige Schlußfolgerungen gezogen werden (man denke an die Anschaffung eines Supercomputers und die damit verbundenen Kosten). Das bedeutet, dass die Analyse der Daten eine wichtige Rolle spielt. Den Wissenschaftlern helfen dabei ihre hohen Erfahrungswerte auf diesem Gebiet. Durch diese Erfahrung ist es ihnen z.B. auch möglich, bei einer Rechnerbeschaffung Leistungen abzuschätzen, die das neue, noch nicht existierende Rechnersystem erbringen soll.

In dieser Arbeit soll das Benchmarking genauer betrachtet und Möglichkeiten untersucht werden, wie es automatisiert unterstützt werden kann. Es existieren bereits Werkzeuge, mit denen der reine Prozeß des Benchmarkings, d.h. die Konfiguration und wiederholte Ausführung von Messungen, erleichtert wird. Auch werden von einigen dieser Werkzeugen bereits kurze Auswertungen ermöglicht, wie die graphische Darstellung des Laufzeitverhaltens eines Programms. Die darauf folgenden Analysen und Abschätzungen werden dann aber von den Wissenschaftlern vorgenommen. Daher ist in der Arbeit zu untersuchen, welche Anfragen die Wissenschaftler an das Benchmarking und die dadurch erhaltene Datenbasis stellen und ob diese Anfragen automatisiert beantwortet werden können.
Ein wichtiger Grund für Benchmarking-Untersuchungen ist wie schon gesehen die Planung und Anschaffung von neuen Rechnern. Daher soll der Schwerpunkt der Untersuchungen darauf liegen, ob aus einer Datenbasis von Benchmark-Messwerten eine automatische Prognose der Performance auf noch nicht ausgemessenen Rechnern gemacht werden kann.

1.1 Aufbau der Arbeit

Die Arbeit ist wie folgt aufgebaut:
Im folgenden zweiten Kapitel wird zunächst das Benchmarking detaillierter erläutert und auch im Kontext von Projekten betrachtet, in denen diesem Bereich eine wichtige Rolle zukommt.
Im dritten Kapitel werden Anfragen an das Benchmarking analysiert, um Ansatzpunkte für die zu schaffenden Automatismen zu finden.
Im vierten Kapitel werden Teilbereiche der Künstlichen Intelligenz beschrieben, die das Potential haben, bei der gestellten Aufgabe hilfreich zu sein. Dieses Vorgehen ergibt sich daraus, dass ein wesentliches Einsatzgebiet von künstlicher Intelligenz Aufgaben sind, die sich mit der Automatisierung intelligenten Vorgehens befassen, wie es hier beim Vor-

gehen der Benchmarking-Experten vorliegt.
Eines dieser Verfahren bildet die Grundlage des Konzepts, die Verfahren müssen daher zuvor verglichen und ihre Vor- und Nachteile herausgearbeitet werden, was in Kapitel 5 geschieht.
In Kapitel 6 werden Tests beschrieben, die zur Herausarbeitung des Konzepts durchgeführt wurden, worauf in Kapitel 7 schließlich die Konzeption des Systems folgt.
Abschließend wird im 8. Kapitel eine Zusammenfassung und ein Ausblick gegeben.

2 Benchmarking

In diesem Kapitel soll das Gebiet des Benchmarkings genauer untersucht und eine Definition des Begriffs *Benchmarking* gefunden werden.

In der Einleitung wurde deutlich, dass mit Testprogrammen die Leistungsfähigkeit eines Computers bestimmt werden kann. Auch in anderen Zusammenhängen wird von Benchmarking gesprochen, nämlich immer dann, wenn ein Vergleich vorgenommen werden soll. In diesem Sinn findet der Begriff auch Verwendung in der Wirtschaft bei Unternehmensprozessen oder Anlagestrategien. Man möchte sich mit seinen Wettbewerbern vergleichen, um seine Marktposition zu verbessern. Hier werden also in den meisten Fällen nicht-greifbare Dinge wie Vorgänge oder Arbeitsabläufe verglichen. Daraus ergibt sich die allgemeine Definition:

Definition 1 *Benchmarking ist eine vergleichende Analyse eines Objekts oder Vorgangs [W2].*

Oft ist es hilfreich, wenn man einen Referenzwert hat, um das Ergebnis des Benchmarks relativ dazu angeben zu können. Wenn der Referenzwert zum Beispiel den Wert 1 aufweist, wird durch einen höheren Wert von z.B. 3 ein entsprechend besseres, in diesem Fall 3-fach besseres, Ergebnis angezeigt. Die absolute Zahl ohne Referenz hätte weniger Aussagekraft.

In dem Bereich der Informatik, der in dieser Arbeit betrachtet werden soll, geht es um einen realen Gegenstand, einen Supercomputer. Wie aber schon in der Einleitung beschrieben, ist dieser Gegenstand eine komplexe Sache, es existieren verschiedenste Architekturen. Die Rechner haben unterschiedliche Netzwerktypen, unterschiedliche Prozessoren sowie Konfigurationen des Speichers. Daraus folgt auch, dass nicht auf allen Systemen die gleichen Programme lauffähig sind.

Daher ist es zur Leistungsmessung erst einmal erforderlich Computerprogramme zu haben, mit denen ein Vergleich möglich wird. In dieses Gebiet ist bereit viel Entwicklungsarbeit geflossen. Es existieren Computerprogramme, mit denen einzelne Kriterien eines Supercomputers ausgemessen werden können, die sogenannten *Low-Level-Benchmarks.* Ein Beispiel dafür ist der HPC Challenge Benchmark [W3], mit dem einzelne Komponenten eines Rechners ausgemessen werden können, wie z.B. die Speicherbandbreite oder die Kapazität des Netzwerks.
Die Programme von Low-Level-Benchmarks sind speziell entwickelt worden, um Messungen machen zu können. Es ist jedoch mit ihnen nur möglich, einzelne Teile eines

Supercomputers zu messen wie z.B. die genannte Bandbreite zum Speicher. Für die Einsetzbarkeit eines Supercomputers ist aber vielmehr die Geschwindigkeit realer Anwendungsprogramm wichtig, der Rechner soll ja eingesetzt und nicht nur vermessen werden! Die Geschwindigkeit eines Anwendungsprogramms wird aber in der Regel nicht nur von einer Komponente des Rechners abhängen, sondern es wird sich ein Zusammenspiel von Prozessorleistung, Netzwerk- und Speicherkennwerten ergeben. Daher ist es auch wichtig, Anwendungsprogramme für das Benchmarking heranzuziehen.

Es wurde also bisher vom Benchmarking als Leistungsvergleich von Rechnern gesprochen und vom Benchmark als dem Messinstrument. Die Programme, die für Benchmarks benutzt werden, wurden kurz in Low-Level-Benchmarks bzw. Anwendungsprogramme kategorisiert. Eine Zusammenstellung von mehreren Benchmarkprogrammen wird als *Benchmark-Suite* bezeichnet. Eine Benchmark-Suite kann unter verschiedenen Gesichtspunkten erstellt worden sein, z.B. können alle Programme zum Austesten eines bestimmten Programmtyps, wie z.B. rechenintensiver Programme [W4], oder einer bestimmten Hardware-Komponente, wie dem Netzwerk, dienen oder die Programme sollen das Anwendungsspektrum der Nutzer eines Rechenzentrums oder Projekts widerspiegeln [W10]. Der Anwender einer Benchmark-Suite kann aus der Obermenge, welche die Suite ihm bereitstellt, die für ihn relevanten Programme auswählen. Dabei ist jedoch zu beachten, dass die Benchmarkprogramme selbst vom Anwender nicht verändert werden dürfen. Zum Vergleich der Laufzeiten oder des Durchsatzes ist es wichtig, dass die Arbeitslast der Programme klar definiert ist.

Benchmarking wurde allgemein als vergleichende Analyse definiert, dabei kann es sich um den Vergleich mehrerer Rechner untereinander handeln, ein Beispiel ist die beschriebene Top500-Liste. Es können auch verschiedenen Programme auf einem Rechner verglichen werden. Dadurch werden Aussagen möglich, welche Programme auf einem Rechnertyp gut laufen und welche eher nicht, z.B. werden parallele Programme, die viel Kommunikation erfordern von einem Rechner mit schnellem Netzwerk profitieren. Es ist auch möglich, ein Programm mit unterschiedlichsten Konfigurationen auf einem Rechner zu testen. Die Konfiguration kann sich dabei auf die Anzahl der genutzten Prozessoren beziehen. Es kann sich z.B. die Frage stellen, ob die Erweiterung eines Supercomputers durch eine größere Anzahl an Prozessoren sinnvoll ist. Dazu wird die Laufzeit des Programms mit einer unterschiedlichen Anzahl an Prozessoren gemessen und der *Speedup* bestimmt [L8]. Der Speedup ist der Vergleich der parallelen Programmläufe zum sequentiellen Lauf auf einem Prozessor hinsichtlich der Laufzeit, man spricht von einem gut *skalierenden* Programm, wenn der Speedup auch für hohe Prozessorzahlen nahe der Anzahl der Prozessoren liegt, d.h. werden die Prozessoren verdoppelt, sollte sich auch die Geschwindigkeit des Programms verdoppeln. Die Skalierung eines Programms kann auch über die *Effizienz* beschrieben werden, die den Speedup ins Verhältnis setzt zur eingesetzten Prozessorzahl, d.h. $E(p) = S(p)/p$. Bei einem skalierenden Programm sollte die Effizienz nahe bei 1 sein und bei wachsender Prozessorzahl gleich bleiben.

Abschließend kann also Benchmarking für diese Arbeit wie folgt definiert werden:

Definition 2 *Benchmarking sind Messungen zum Vergleich von parallelen Supercomputern unter Nutzung von geeigneten Messprogrammen, den Benchmarks.*

Es wurde deutlich, dass Benchmarking ein wichtiges Kriterium bei der Leistungsbewertung und damit auch bei der Planung und Beschaffung von Supercomputern ist.
Aus diesem Grund beschäftigen sich das Jülich Supercomputing Centre und auch große Projekte wie die europäischen Projekte DEISA und PRACE sowie internationale Vereinigungen wie SPEC oder NAS mit Benchmarking [W6, W9, W12, W14].

2.1 Benchmarking im Jülich Supercomputing Centre

Das Jülich Supercomputing Centre (JSC), ein Institut des Forschungszentrums Jülich (FZJ), stellt Wissenschaftlern am Forschungszentrum Jülich, an Universitäten und Forschungseinrichtungen in Deutschland (und z.T. in Europa) sowie der Industrie Rechenkapazität der höchsten Leistungsklasse zur Verfügung. Dies umfasst die Bereitstellung, den Betrieb und insbesondere die Weiterentwicklung der Supercomputer und der technischen Infrastruktur, u.a. der Datenspeicher, Visualisierungssysteme und Netzwerke sowie der Software. Eine weitere wesentliche Aufgabe ist die Benutzerunterstützung, z.B. die Parallelisierung von Anwendungsprogrammen, die auf den Supercomputern eingesetzt werden sollen.

Benchmarking ist im JSC insbesondere bei der Anschaffung und Planung von neuen Rechnern von Bedeutung.
Zum einen müssen die bei der Anschaffung eines Rechners im Vertrag festgelegten Leistungsmerkmale überprüft werden. Das geschieht nach Lieferung des Rechners, die Benchmarkingergebnisse sind eine Grundlage der Abnahme des Rechners.
Für den Fall, dass nicht ein neuer Rechner angeschafft wurde, sondern ein bestehender erweitert wird oder durch einen Rechner der gleichen Architektur aber z.B. mit schnelleren Prozessoren ersetzt wird, ist das Benchmarking wichtig, um die gewünschte Leistungssteigerung zu überprüfen. Die gleichen Benchmarkläufe müssen dann auf dem alten und dem neuen System durchgeführt werden. Nicht jedes Programm wird aus den schon beschriebenen Gründen die gleiche Leistungssteigerung erfahren, aber es kann bestimmt werden, ob in der Gesamtheit das Ziel erfüllt wurde.
Zum anderen sind Benchmarks im JSC auch eine Hilfe bei der Planung und Entwicklung von Rechensystemen und neuen Architekturen. Die geplante Architektur kann als Prototyp aufgebaut und die Leistung überprüft werden.

In diesem Kapitel wurde bereits deutlich, dass für das Vermessen eines Rechners in der Regel viele verschiedene Benchmarks nötig sind. Es sind aber auch viele Benchmarkläufe dieser Benchmarks erforderlich. Um z.B. eine vollständige Speedupkurve zu erhalten, müssen Benchmarkläufe mit den unterschiedlichen Anzahlen an Prozessoren hintereinander durchgeführt werden. Die Konfiguration sowie das Kompilieren und automatische Absetzen von Benchmarkläufen auf unterschiedlichen Rechnerplattformen

verlangt aber einen hohen Verwaltungsaufwand. Zudem werden durch einen Benchmarklauf große Mengen an Daten erzeugt, die gesammelt und ausgewertet werden müssen. Dabei werden meistens auch verschiedenartige Benchmarkläufe miteinander kombiniert. Schließlich sind die Daten zu analysieren, die Plausibilität zu prüfen und Statistiken zu erzeugen. Daher gehört zum professionellen Benchmarking auch eine Benchmark-Umgebung, ohne die alle diese Schritte von Hand durchgeführt werden müssten.

2.1.1 Benchmarking-Umgebung JuBE

Aus den oben genannten Gründen ist am Jülich Supercomputing Centre mit der Entwicklung des Softwarewerkzeugs JuBE (Jülich Benchmarking Environment [W5]) begonnen worden. Mit Hilfe von JuBE kann eine Benchmark-Suite automatisch kompiliert werden, zudem können die geplanten Läufe automatisiert gesteuert und auf dem Rechner gestartet werden.
Alle Parameter, die für die Konfiguration und Kompilation benötigt werden, werden in Konfigurationsdateien gespeichert. JuBE generiert daraus einen oder mehrere Benchmarkläufe, für jeden Lauf wird dabei ein separates Verzeichnis erzeugt. Anpassungen an der Konfiguration müssen vorgenommen werden, wenn ein neuer Benchmark aufgenommen oder eine neue Rechnerplattform definiert werden soll. Für jeden Benchmark können einzeln verschiedene Benchmarkläufe definiert werden. Dadurch wird es ermöglicht, z.B. eine Spezifikation zu erzeugen, durch die alle Läufe für ein Speedupdiagramm in einem Benchmarklauf durchgeführt werden können.
JuBE erzeugt für jeden Benchmarklauf einen eindeutigen numerischen Schlüssel. Dieser Schlüssel ist auch Teil der Namen für temporäre Arbeitsverzeichnisse und Ausgabedateien. Dadurch kann JuBE zum einen die Jobs eines Laufs über den Batch-Scheduler des Systems abschicken ohne auf eine definierte Abarbeitungsreihenfolge angewiesen zu sein. Zum anderen ermöglicht der Schlüssel die Speicherung der Ergebnisdateien in einem zentralen Verzeichnis oder einer Datenbank. Ein zweiter Aufruf von JuBE sammelt diese Ergebnisdateien, die alle Meßergebnisse enthalten, und durchsucht sie, um an relevante Performancedaten zu gelangen.
In der folgenden Grafik sind alle Komponenten der aktuellen Benchmarking Umgebung zu sehen und an den grünen Markierungen zu erkennen.

Das System befindet sich zwar schon im Einsatz, ist aber auch noch in Weiterentwicklung. In der Graphik ist zum einen der gelbe Bereich mit dem Start-GUI zu sehen, durch das die Konfiguration der Umgebung vereinfacht werden soll. Zum anderen sieht man den roten Bereich, in dem die Planung einer Analyse-Schnittstelle der Benchmarking-Ergebnisse zu sehen ist. Es soll also in Zukunft möglich sein, Benchmarks automatisiert auszuwerten, d.h. man kann sich z.B. Speedup-Kurven anzeigen lassen. Diese Schnittstelle soll helfen, die große Anzahl an Ergebnisdaten noch besser verarbeiten zu können. Auch die Untersuchungen dieser Arbeit sollen zum besseren Verständnis der Analyse von Benchmarking-Läufen beitragen.

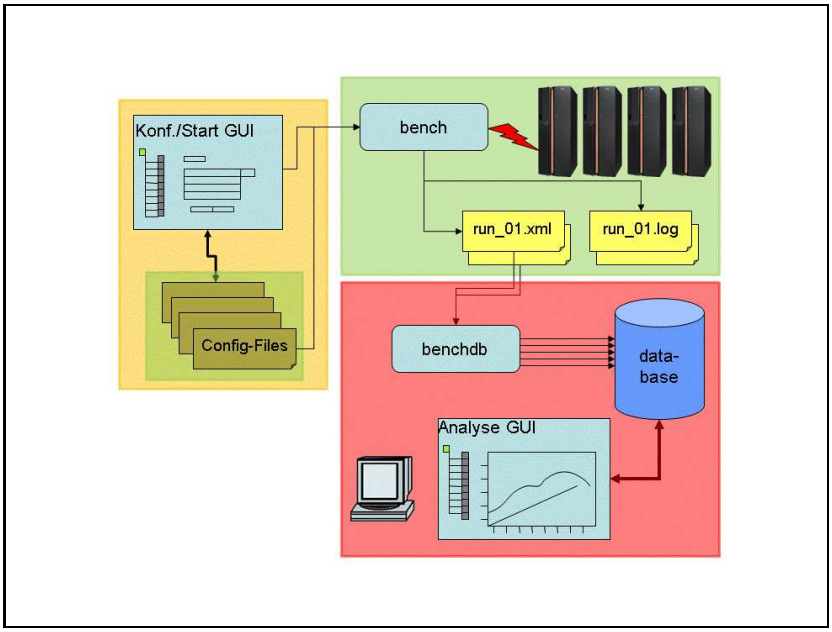


Abbildung 2.1: Die Jülicher Benchmarking-Umgebung JuBE ermöglicht das automatische Erzeugen und Starten von Benchmarkläufen. In Planung befindet sich eine Datenbank-Schnittstelle mit Möglichkeiten der Datenanalyse.

2.2 Benchmarking in den EU-Projekten DEISA und PRACE

Das JSC ist Partner in den europäischen Projekten DEISA und PRACE und arbeitet dort unter anderem in den Arbeitspaketen zum Benchmarking mit. In den folgenden Abschnitten sollen die beiden Projekte kurz beschrieben werden und insbesondere der Schwerpunkt auf ihre Arbeit im Benchmarking gelegt werden.

2.2.1 Distributed European Infrastructure for Supercomputing Applications (DEISA)

DEISA ist ein im 6. Rahmenprogramm der EU gefördertes Projekt, an dem elf Höchstleistungsrechenzentren in Europa beteiligt sind [W6]. Das Ziel von DEISA ist es, eine europaweite Supercomputing-Infrastruktur aufzubauen und zu betreiben, die auf den lokalen Rechenzentren fußt. Diese Infrastruktur umfasst ein gemeinsames Netzwerk, das die Supercomputer der beteiligten Partner verbindet, und ebenso den Aufbau eines gemeinsamen Dateisystems sowie eines grundlegenden Software-Stacks, der den Benutzern auf allen beteiligten Systemen zur Verfügung stehen muß. Zur Nutzung der Supercomputer-Ressourcen wird im Projekt auch die benötigte sogenannte Middleware bereitgestellt und weiterentwickelt, also die Software, über die der Anwender seine Programme an die Rechner der verschiedenen Rechenzentren senden kann. Es wurden Prozesse geschaffen, um den Endnutzern den Zugang zu den Ressourcen zu ermöglichen, so dass die verteilte

Rechnerlandschaft für sie keinen Zuwachs an Komplexität darstellen soll.
Nutzer haben somit Zugang zu im Netz verteilten Ressourcen und können Ihre Anwendungen auf den Rechnern laufen lassen, die am besten auf sie zugeschnitten sind.
DEISA ist damit eines der Projekte im *Grid-Computing*. Von Grid-Computing spricht man dann, wenn es um eine Infrastruktur geht, die in der Regel geographisch getrennte, unabhängige Ressourcen zur gemeinsamen Nutzung bereitstellt [W7].

DEISA Benchmarking

Benchmarking spielt bei DEISA eine Rolle, um die verschiedenen Systeme, die die DEISA-Partner in das Projekt eingebracht haben, miteinander vergleichen zu können [W8].

Dazu wurde in einem Arbeitspaket des Projekts eine eigene Benchmark-Suite entwickelt. Der Aufbau dieser Suite wird hier beschrieben, da für ein Benchmarking Prognosesystem die Daten von Benchmark-Suiten die Grundlage bilden.
Die DEISA Benchmark-Suite besteht aus wissenschaftlichen Programmen verschiedenster Disziplinen, die die Nutzungsverteilung der Forschungsgebiete auf den beteiligten europäischen Großrechnern widerspiegeln sollen. Es handelt sich um 12 wissenschaftliche Programme aus den Gebieten

- Astrophysik,
- Stömungssimulation,
- Erde und Klima,
- Lebenswissenschaften und Informatik,
- Materialwissenschaft,
- Plasmaphysik,
- Quantenchromodynamik.

Durch diese Programme soll die Leistung der unterschiedlichen Rechner messbar gemacht werden. Die Suite ist konzipiert für Rechner mit einer Peak-Performance bis in einen Bereich von mehreren hundert Teraflops. Sie steht seit dem Projektende frei zur Verfügung.
In dem Arbeitspaket zum Benchmarking wurde zudem Dokumentation über die Programme der Benchmark-Suite geschrieben, sowie Leitfäden für die Installation der Programme und die Konfiguration von JuBE, das das JSC in DEISA mit eingebracht hat, entwickelt. Um die Benchmark-Suite auf der Webseite [W8] des Projekts zur Verfügung stellen zu können, mussten Lizenzvereinbarungen abgeklärt werden. Daraufhin wurden auf der Webseite auch Beispielergebnisse abgelegt. Um die Vergleichbarkeit eigener Messungen mit diesen Ergebnissen zu gewährleisten, wurden Regeln zur Benutzung der Suite festgelegt.

Inzwischen existiert mit DEISA2 ein im 7. Rahmenprogramm der EU gefördertes Nachfolgeprojekt zu DEISA. In diesem Projekt wird die Benchmark-Suite weiterhin gepflegt und an neue Anforderungen angepasst.
Neue Anforderungen bestehen hier unter anderem darin, dass die Suite auf weitere Rechnerarchitekturen übertragen wird. Im ersten DEISA-Projekt wurden IBM-Power4 sowie CRAY XT4 Systeme vermessen, inzwischen hat sich die Rechnerlandschaft der beteiligten Partner weiter verändert. Die Benchmark-Suite wird zum einen auf die Nachfolger IBM-Power6 sowie CRAY XT5 übertragen werden, zum anderen aber auch auf zusätzliche Plattformen wie IBM Blue Gene/P oder das Vektorsystem CRAY X2. Das hat zur Folge, dass sich die zur Verfügung stehende Datenbasis an Benchmark-Ergebnissen vergrößern wird. Eine Aufgabe wird es in DEISA2 sein, aus diesen Messungen Faktoren zu ermitteln, die die verschiedenen Rechner der beteiligten Partner vergleichbar machen und auch als Basis zur Preisgestaltung einer CPU-Stunde auf einem speziellen System bzw. einer Standardeinheit innerhalb des Projekts dienen können.

2.2.2 Partnership for Advanced Computing in Europe (PRACE)

PRACE ist ein im 7. Rahmenprogramm der EU gefördertes Projekt [W9]. An PRACE nehmen 14 Partner teil, wie in DEISA sind das Höchstleistungsrechenzentren in Europa. Die europaweite Supercomputing-Infrastruktur, die durch PRACE geschaffen werden soll unterscheidet sich allerdings von der DEISA-Struktur dahingehend, dass hier ein gemeinsames Rechenzentrum für Europa entsteht und nicht eine verteilte, gleichwohl gemeinsam genutzte, Rechnerlandschaft. Dieses Rechenzentrum zeichnet sich dadurch aus, das auch Hardwarebeschaffung und Standortwahl eines Supercomputers von den Projektpartnern abgestimmt werden und die Rechenzeit durch ein eigenes, gemeinsames Peer-Review-Verfahren an Forscher in Europa verteilt wird. Zudem soll durch den Zusammenschluß in PRACE die Beschaffung eines Petaflop-Systems für Europa ermöglicht werden, es entsteht damit ein sogenanntes Tier-0-Rechenzentrum, wohingegen bei DEISA Tier-1-Rechenzentren miteinander verbunden sind, s.Abb. 2.2.

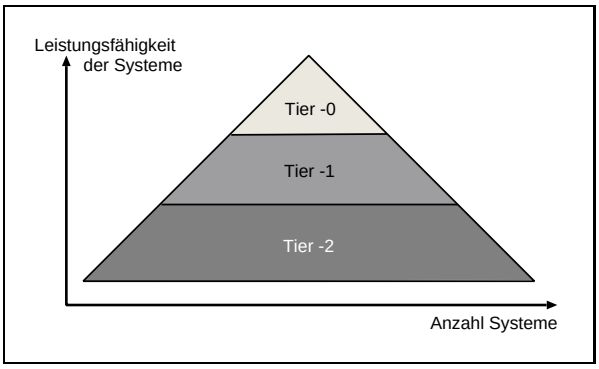


Abbildung 2.2: Die Pyramide zur Charakterisierung von Rechenzentren basierend auf der Art ihrer Rechnersysteme.

PRACE Benchmarking

Die Zielrichtung des Benchmarking in PRACE geht daher noch einen Schritt weiter als in DEISA, da nicht nur Rechner verglichen werden sollen, sondern die Benchmarking-Ergebnisse entscheidend sind im Rahmen von Beschaffungsprozessen der Supercomputer. Zudem sollen sich diese Rechner mit ihrer Peak-Performance im Petaflop-Bereich befinden. Die Benchmarking-Suite, die in PRACE entwickelt werden soll, muß also Programme enthalten, die bis in diesen Bereich hinein skalieren.
Die Schwierigkeit besteht hier darin, dass die Benchmark-Suite geplant wird für die Vermessung von Rechnern, die zur Zeit noch nicht existieren. Dies ist der Punkt, an dem diese Arbeit ansetzt.
Auch die PRACE Benchmark-Suite soll hier kurz beschrieben werden.
Um einen geeigneten Satz an Benchmark-Programmen zu finden, wurde in PRACE eine Erhebung unter den Projektpartner gemacht. Das Ziel war, zunächst die Programme zu identifizieren, die aktuell die meiste Rechenzeit in Europa verbrauchen, zudem sollten die Nutzer ihre zukünftige Planung hinsichtlich benötigten Rechenressourcen abschätzen. Zur Klassifizierung der Programme, die so in die Auswahl für eine PRACE-Benchmark-Suite gekommen sind, wurde eine Matrix erarbeitet, die die Programme zum einen in ein wissenschaftliches Gebiet einordnet, zum anderen Aufschluß über die Algorithmen liefert [W10]. Die Programme wurden in die folgenden Forschungsgebiete eingeteilt:

- Astronomie,
- Rechnergestützte Chemie,
- Rechnergestütztes Ingenieurwesen,
- Stömungsforschung,
- Kondensierte Materialien,
- Erde und Klima,
- Lebenswissenschaften,
- Teilchenphysik,
- Plasmaphysik,
- Sonstige.

Bei den Algorithmen handelt es sich um typische Kerne aktueller Simulationsprogramme, in der Suite sollte jeder Kern vorhanden sein. Die Liste von Algorithmen entstammt einer Untersuchung der University of California in Berkeley [W11] über die aktuellen Herausforderungen im parallelen Supercomputing. In dem Bericht werden die derzeitigen Supercomputer-Architekturen den Anwendungen gegenübergestellt sowie Programmiermodelle betrachtet. Die Anwendungen werden nach ihren numerischen Methoden klassifiziert, *7 dwarfs* genannt. Die 7 dwarfs sind:

- Strukturierte Gitter,
- Unstrukturierte Gitter,
- Dichtbesetzte Matrizen,
- Dünnbesetzte Matrizen,
- Partikelmethoden,
- Spektralmethoden,
- Simulationsmethoden (z.B. Monte Carlo).

Diese Algorithmen sollen helfen zu erkennen, welche Programme gut geeignet sind, die verschiedenen Rechnerarchitekturen zu testen, die für die PRACE-Protoypen in Frage kommen.

2.3 Benchmarking im internationalen Umfeld

Für das Benchmarking im internationalen Vergleich werden im Folgenden zwei Organisationen als Beipiel herangezogen - SPEC und NAS.
Diese beiden Organisationen stellen jeweils Benchmark-Suiten für das Supercomputing zur Verfügung. Obwohl darauf hingewiesen wird [W13], dass insbesondere für eine Rechnerbeschaffung die Anwendungen, die auf dem System laufen sollen, der beste Benchmark sind, sollen die Benchmarks einer unabhängigen Vereinigung einen objektiven Vergleich unterstützen.

2.3.1 Standard Performance Evaluation Corporation - SPEC

Die Standard Performance Evaluation Corporation (SPEC) ist eine Vereinigung von Rechnerherstellern, Software-Firmen, Universitäten, Forschungszentren, Systemintegratoren und Beratern, die es sich zum Ziel gesetzt haben, standardisierte Benchmarks für Computersysteme bereitzustellen [W12].

SPEC hat eine Reihe von Benchmark-Suiten entwickelt, mit denen unterschiedliche Komponenten eines Rechners untersucht werden können. So existieren Benchmarks für die Ausmessung der Performance von Graphikkarten oder Dateisystemen sowie die Untersuchung von der Performance rechenintensiver Programme und der MPI- bzw. OpenMP-Programmierung [1].
An der SPEC CPU2006 Suite kann SPECs Strategie beim Aufbau einer Suite gesehen werden. Die Suite CPU2006 wurde zum Testen von rechenintensiven Programmen entwickelt, bei denen besonders die Prozessoren und der Speicher beansprucht werden. Bei den Benchmarks handelt es sich nicht um Low-Level-Benchmarks, sondern sie sind

[1] MPI (Message Passing Interface) und OpenMP (Open Multi-Prozessing) sind Programmiermodelle für parallele Rechnerarchitekturen.

abgeleitet von realen Anwendungen vieler Forschungsgebiete. Diese Programme wurden jedoch bearbeitet, um eine möglichst hohe Portabilität zu gewährleisten und weitgehenste Hardwareunabhängigkeit zu erreichen.

2.3.2 NASA Advanced Supercomputing - NAS

Die Advanced Supercomputing Abteilung der NASA stellt ebenfalls seit vielen Jahren Benchmarks für das Supercomputing zur Verfügung, die NAS Parallel Benchmarks (NPB) [W14]. Der Schwerpunkt von NAS liegt auf Anwendungen aus der Computational Fluid Dynamic (CFD).

Die Benchmarks sollen möglichst generisch sein, um wie bei SPEC keine Architektur zu bevorzugen. Die Strategie bei der Entwicklung der NPB Suiten unterscheidet sich von der Vorgehensweise von SPEC aber dahingehend, dass eine NPB Suite zu einem grossen Teil aus Kernen von Programmen besteht und nur ergänzt wird durch Nachbildungen von typischen Anwendungen der CFD. Diese Reduzierung auf Kerne von Algorithmen kann die Übertragung von Benchmarking-Ergebnissen auf verschiedene Rechner erleichtern, da die Messungen nicht von Programmiermodellen abhängen, sondern auf abstrakte Kerne bezogen werden können.

2.4 Benchmarking – Kurzzusammenfassung

In diesem Kapitel wurde zunächst eine Definition für das Benchmarking im Rahmen des Supercomputing gegeben sowie verschiedene Arten von Benchmarks erläutert: Anwendungs- und Low-Level-Benchmarks.
Weiterhin wurden die Arbeiten zum Benchmarking im Jülich Supercomputing Centre, insbesondere die Entwicklung der Benchmarking-Umgebung JuBE, beschrieben. Anhand der europäischen Projekte DEISA und PRACE sowie den internationalen Organisationen SPEC und NAS wurde das Umfeld des Benchmarking beleuchtet und die verschiedenen Strategien zum Aufbau einer Benchmark-Suite untersucht.

3 Problemanalyse

In den vorangegangenen Kapiteln wurde eine Definition zum Benchmarking gegeben, die Einsatzgebiete aufgezeigt und auch kurz Probleme angerissen, die sich u.a. durch die schwere Vergleichbarkeit unterschiedlichster Programme sowie Rechner und auch durch die Komplexität der Auswertung von Benchmarking-Läufen ergeben.
In diesem Kapitel soll dargelegt werden, welche Anfragen konkret dem Benchmarking gestellt werden. Eine Analyse dieser Fragen soll dazu führen, Ansatzpunkte zur Automatisierung zu finden.

3.1 Anfragen an das Benchmarking

In der Diskussion mit Wissenschaftlern im Bereich Benchmarking und insbesondere auch im aktuellen Prozess der Entwicklung einer Benchmark-Suite im Projekt PRACE zeigt sich eine Reihe Fragen, für die man sich aus dem Benchmarking Antworten erhofft.

Ausgehend von einem vorhandenen Satz an Benchmark-Programmen und zugrunde liegenden Rechnersystemen sind folgende Fragen von Interesse:

1. Das bekannte Programm P_1, von dem bereits Messungen vorliegen, ist schon auf dem Rechner R_1 gelaufen. Welche Aussagen können über den Rechner R_1 gemacht werden, ohne ihn genauer zu kennen? Dieses Vorgehen kann als BlackBox-Prinzip für Rechner bezeichnet werden und kommt zur Anwendung bei der Untersuchung eines neuen Rechners.

2. Wie wird sich das Programm P_2 auf einem unbekannten, evtl. noch nicht verfügbarem Rechner R_2 verhalten? Das ist die Voraussage von Messwerten, diese Anfrage spielt eine Rolle bei der Rechnerbeschaffung, sie wird bei Angeboten von den Benchmarkern der Rechnerhersteller gemacht:
   Das Programm P_2 ist bekannt und ebenso die Kenngrößen des neuen Rechners, aber dieser ist noch nicht gebaut, daher können noch keine Messungen gemacht werden, sondern nur Voraussagen basierend auf der vorhandenen Datenbasis und Kennwerten des zukünftigen Systems.
   Man kann diese Fragen auch auf die Gesamtheit der Programme einer Benchmarksuite anwenden: Wie würde sich die Benchmark-Suite auf einem neuen, derzeit noch nicht vorhandenen Rechner verhalten? Das entspricht dem, was bei einer Ausschreibung gemacht wird: eine gewünschte Leistung für die Programme einer Benchmark-Suite wird vorgegeben und soll dann vom Hersteller eingehalten werden.

3. Das unbekannte Programm P_3 ist schon auf dem bekannten Rechner R_3 gelaufen. Welche Aussagen können über P_3 gemacht werden, ohne es genauer zu kennen? Dieses Vorgehen entspricht einem BlackBox-Prinzip für Programme.

Eine Benchmark-Suite liefert als Antwort auf diese Fragen Messergebnisse wie Programmlaufzeiten oder Durchsatz. Diese Ergebnisse werden hinsichtlich ihrer Güte interpretiert, um auf eine Antwort wie „gut“ oder „schlecht“ zu kommen. Dabei ist dieses Güteresultat in der Regel kein klares Ja oder Nein, sondern es handelt sich um Wahrscheinlichkeiten [0.0,1.0].

Es gibt aber auch die Situation, wie zur Zeit im Projekt PRACE, dass eine Benchmark-Suite erst in der Entwicklung ist. Dann sehen die Fragen anders aus:

1. Welche Programme sollen in die Benchmark-Suite aufgenommen werden? Dazu muss bekannt sein, welche Kennwerte oder Elemente des Rechners untersucht werden sollen, aber auch die Eigenschaften der einzelnen Programme müssen gut bekannt sein. Es kann zudem notwendig sein, die Programme nach Verbreitung und Akzeptanz, dem Themengebiet oder anderen Kriterien zu gewichten.

2. Wie würde sich ein neues Programm in einer bestehenden Benchmark-Suite verhalten? Würde die Datenbasis dadurch erweitert oder ist die Information schon darin enthalten? Zum Beispiel besteht in PRACE derzeit die Frage, ob es ausreicht im Bereich Chemie einen einzigen Simulationscode in die Suite aufzunehmen, oder ob zur Ergänzung ein zweiter notwendig ist. Also geht es im Grunde um die Frage, ob mit Hilfe der Messwerte des ersten Codes die Messwerte des zweiten vorausgesagt werden können oder nicht. Hilfreich kann bei dieser Frage die Betrachtung der Kerne der Anwendungen sein, also die 7 dwarfs, die evtl. einen Hinweis darauf geben können, wie verwandt die Programme tatsächlich sind.

3. Bei der Entwicklung einer Benchmark-Suite stellt man sich auch die Frage, ob nur Anwendungsprogramme in die Suite aufgenommen werden sollen oder auch künstliche Benchmark-Programme aus den typischen Kernen, die Frage ist dann, wie gut eine Sammlung von Programmkernen die Charakteristik der realen Programme repräsentiert.

Nicht auf alle diese Fragen kann heute schon eine Antwort aus dem Benchmarking gegeben werden, insbesondere nicht durch automatisierte Prozesse. Zur Zeit basiert ein gutes Benchmarking zu einem großen Teil auf den Erfahrungen der Benchmarker und der Zusammenarbeit mit den Programmentwicklern und Herstellern.
Die Frage ist, lassen sich diese Anfragen automatisisert beantworten? Gibt es Techniken oder Verfahren, die zur Lösung dieses Problems herangezogen werden können?

Durch die beschriebenen Anfragen ergibt sich eine Dreier-Beziehung zwischen den Bereichen Rechner - Programme - Messungen, s. Abb. 3.1 auf der nächsten Seite. Aufgabe eines automatisierten Benchmarking-Systems wäre es dann diese Beziehungen nachprüfbar oder vorhersagbar zu machen.

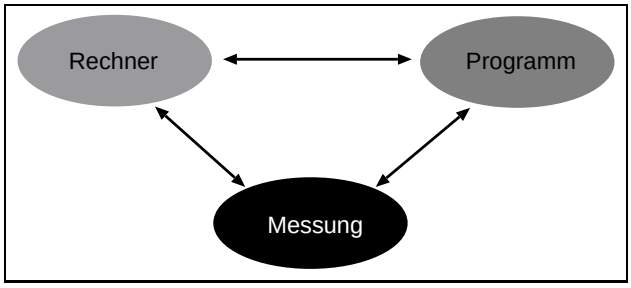


Abbildung 3.1: Im Benchmarking existiert eine Dreier-Beziehung zwischen verschiedenen Bereichen: den auszumessenen Rechnern, den darauf laufenden Programmen und den Benchmarking-Messungen, durch welche die Leistung sichtbar wird.

3.2 Benchmarking-Klassen

Um das Schaubild 3.1 zu konkretisieren, muss zunächst beantwortet werden, wie die einzelnen Bereiche, die im Folgenden als Klassen bezeichnet werden sollen, im Inneren aussehen. Die entscheidene Frage ist danach, wie die Verbindungspfeile gestaltet werden können, damit es möglich wird, Anfragen an das System zu stellen.

3.2.1 Programme

Die Klasse der *Programme* besteht aus Kenngrößen von parallelen Programmen. Interessante Kenngrössen zur Untersuchung eines Benchmarkprogramms sind z.B.:

- Die *Datenrepräsentation* des Programms, handelt es sich um 2D-, 3D- oder Partikel Daten, sind die Daten strukturiert oder unstrukturiert (entscheidend z.B. bei CFD-Codes)?
- Die *Datenzerlegung* des Algorithmus, z.B.werden bei Particle In Cell (PIC) die Simulationsteilchen in Zellen eingeteilt und die Wechselwirkung zwischen den Zellen berechnet.
- Das *Parallelisierungsschema*, wie werden die Daten auf die Prozessoren verteilt? Dabei können bei einer Simulation die Teilchen z.B. gleichmäßig auf die CPUs verteilt werden oder es wird eine geometrische oder funktionale Gebietsaufteilung vorgenommen. Die Aufteilung hat direkten Einfluss darauf, wie häufig ein Austausch von Daten an den Rändern der Gebiete vorgenommen werden muss und wie gross die Datenmengen beim Austausch sind.
- Das *Kommunikationsmuster*, dort wird die Korrelation zur Hardware besonders deutlich. Wie werden die Daten untereinander ausgetauscht, z.B. direkt zwischen zwei Prozessoren oder als globale Kommunikation? Das kann z.B. bedeuten, dass ein spezielles Netz für globale Kommunikation hilfreich für die Anwendung sein kann.

- Die *Speicherzugriffsmuster* eines Programms müssen ebenfalls zur Rechnerarchitektur passen, wie ist der Zugriff auf die Daten im Hauptspeicher, sind die Daten z.B. sequentiell oder wahlfrei abgelegt? Das kann auch im Zusammenhang mit der Cache-Konfiguration des Rechners interessant sein.

- Das *I/O-Verhalten*, also die Ein- und Ausgabe, beeinflusst das Laufzeitverhalten eines Programms. Evtl. muss das Programm auf Ausgaben einzelner Prozessoren warten, wenn die Ausgabe nicht parallel gestaltet wird. Zudem kann die Geschwindigkeit, die die Hardware für das Schreiben oder Lesen von Dateien zur Verfügung stellt, ein limitierender Faktor sein.

- Die *Laufzeitkonfiguration* - darunter versteht man z.B. die Anzahl der Prozessoren, auf denen das Programm gestartet wird, oder die Größe der Eingabedaten - ist keine Programmeigenschaft, hat aber einen entscheiden Einfluss auf das Laufzeitverhalten und die darauf aufsetzenden Benchmark-Messungen eines Programms. Klar ersichtlich ist das bei der gewählten Anzahl an Prozessoren, die für den Lauf gewählt werden, bei einer hohen Anzahl sollte das Programm zwar schneller bearbeitet werden, aber die Kommunikation wird je nach Kommunikationsanteil in der Anwendung und Kommunikationsmuster ansteigen.

Um die Klasse der Progamme detailliert aufbauen zu können, wird eine Recherche bei Entwicklern bzw. Benutzern der Programme erforderlich sein oder das Wissen kann durch Source-Code-Analyse oder Literatur-Studie erlangt werden.

3.2.2 Rechner

Die Klasse der *Rechner* besteht aus Kenngrößen der Rechner-Hardware, die ausgemessen werden soll, typische Größen sind dabei z.B. die folgenden:

- Die *Architektur* des Rechners ist entscheidend für die Programmiermodelle der Programme. Schon im Einleitungskapitel wurden Beispiele der aktuellen Architekturen von Supercomputern aufgeführt, es handelte sich dort um Massiv-Parallel Rechner (MPP) und hybride Systeme. Ebenfalls verbreitet sind Symmetrische Multiprozessor-Systeme (SMP), bei denen die Prozessoren auf einen gemeinsamen Speicher zugreifen. Dabei sei angemerkt, dass die Untersuchung einer sequentielle Rechnerarchitektur nicht Bestandteil der Arbeit ist.

- Die *Anzahl der Knoten* bzw. *CPUs* ist entscheidend für die mögliche Parallelität der Programme.

- Die *Anzahl CPUs pro Knoten* muss bei Benchmarking-Auswertungen beachtet werden, da ein Programm sein Verhalten verändern kann, wenn es über Knotengrenzen hinweg kommunizieren muss.

- Ähnliche Aussagen gelten auch für die *Größe der Knoten mit gemeinsamem Speicher* und die *Anzahl Cores pro Chip*.

- Die Grösse und Art des *Speichers* beeinflusst die Zugriffszeiten der Programme auf die benötigten Daten.
- Das gilt ebenso für die Grösse und Anzahl der *Caches*. Auch hier können ähnlich wie beim Überschreiten von Knotengrenzen Effekte bei Messungen auftreten, wenn z.B. die Datenmenge nicht mehr in den Cache passt und häufig aus- und eingelagert werden muss.
- Das *Netzwerk* ist entscheidend für die Kommunikationsgeschwindigkeit von Programmen, es kann durch seine Architektur (Torus, Tree, FatTree, Bus,...) und durch zugehörige Kennwerte wie Latenz, Bandbreite und Bisektionsbandbreite beschrieben werden.

Die Liste ist erweiterbar und sicherlich gibt es schon eine Reihe von Klassifizierungen von Rechnern (z.B. von Flynn die Klassifizierung nach Instruktionen pro Daten [L3], dies ist aber sehr allgemein gehalten und geht nicht so sehr ins Detail wie in der obigen Liste angestrebt). Daher kann es schwierig werden hier eine allgemeingültige Klassifizierung zu finden, in dem Fall muss ein eigener Satz an Kenngrößen definiert werden.
Das Wissen zum detaillierten Aufbau der Klasse der Rechner kann aus Datenblättern oder der Literatur erlangt werden kann.

3.2.3 Messungen

Die Programme und die Rechner sind damit zunächst einmal isoliert voneinander beschrieben. Das Element, dass Aussagen zwischen diesen beiden Klassen möglich macht, sind die Messungen, so dass das Schaubild 3.1 eigentlich wie in Abbildung 3.2 verstanden werden muss.

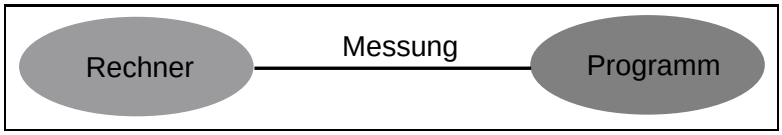


Abbildung 3.2: Die Messungen ermöglichen erst Aussagen über die Verbindung der Klassen Rechner und Programm.

Die Klassifizierung der *Messungen* baut sich aus dem Wissen auf, dass aus realen Läufen der Programme auf verschieden Rechner kommt. Hier gibt es auch Bereiche, in die Performancedaten von parallelen Programmen einsortiert werden können, z.B.:

- die *Gesamtlaufzeit* eines Programms,
- die mit *Kommunikation* verbrachte Zeit,
- die verbrachte Zeit in einzelnen Programmteilen oder Schleifeniterationen,

- die Größe und Anzahl der übertragenen Nachrichten,

- die Anzahl der missglückten Cachezugriffe (Cache-miss).

Diese Angaben können durch Profiling-Infomationen, z.B. mit Tools wie *gprof*, *xprofiler* oder auch Scalasca erhalten werden. Scalasca erstellt zudem Traces, also genaue Aufzeichnungen aller Daten auf jedem Prozessor, z.B. die Ein- und Austrittszeitpunkte von Routinen [W15]. Aus den Messwerten können auch abgeleitete Größen entstehen, wie der Speedup.
Hinzu kommen Kennwerte, die auf einzelnen CPUs bzw. SMP-Nodes gemessen werden, z.B. mit dem Hardware Performance Monitor (HPM, [W16]). Dazu zählen die MFlop-Rate sowie die Rate an erfolgreichen bzw. missglückten Cachezugriffen.
Das Benchmarking-System wird sich auf die Analyse paralleler Programme fokussieren, so dass die serielle Performance von Programmen nicht so eine große Rolle spielt. Dabei ist zu beachten, dass grundsätzlich schon ein Zusammenhang zwischen serieller und paralleler Performance besteht, ein gut optimierter serieller Code ist wichtig für ein wirklich gut skalierendes Programm. Dadurch kann gegebenenfalls ein zu großer Kommunikations-Overhead in der Parallelisierung vermieden werden.

Vorhandene Datenbasis

Im Projekt DEISA ist durch die DEISA Benchmark-Suite bereits eine Datenbasis an Messwerten entstanden. Wie im Abschnitt 2.2.1 auf Seite 10 dargelegt, besteht die Suite aus 12 parallelen Anwendungsprogrammen. Mit diesen Programmen wurden die Supercomputer der Projektpartner vermessen. Im Laufe des Projekts wurden dazu etwa 4500 Programmläufe gemacht, um für jedes Programm auf jedem Rechner Skalierungskurven zu erhalten.
Von den Programmläufen stehen die Konfigurationen des Laufes zur Verfügung, das Ergebnis des Benchmarks ist jeweils eine Laufzeit. Detailliertere Werte wurden in DEISA nicht vermessen, entscheidend war die Laufzeit und der sich daraus ergebene Speedup der Programme.

Im Projekt PRACE entsteht derzeit die PRACE Benchmark-Suite. Die ersten Messungen wurden schon durchgeführt, so dass im weiteren Verlauf des Projekts auch dort eine entsprechende Datenbasis entstehen wird. Bei PRACE ist geplant, zusätzlich zum Speedup auch Werte wie den Anteil an Kommunikation oder die Anzahl an Cache-misses auszumessen.

Zudem liefern SPEC und NAS auf ihren Webseiten Ergebnisse ihrer Benchmarks. Diese Ergebnisse entstehen aus der Laufzeit der Programme zu einem gegebenen Referenzwert.

3.3 Zusammenhänge der Klassen

Die Klassen sind somit umrissen und es wurde dargelegt, dass die Verbindung zwischen Programmen und Rechnern über die Messungen hergestellt werden kann. Die eigentlichen Zusammenhänge bestehen nun zwischen den einzelnen Elementen der Klassen, wie in Abb. 3.3 skizziert.

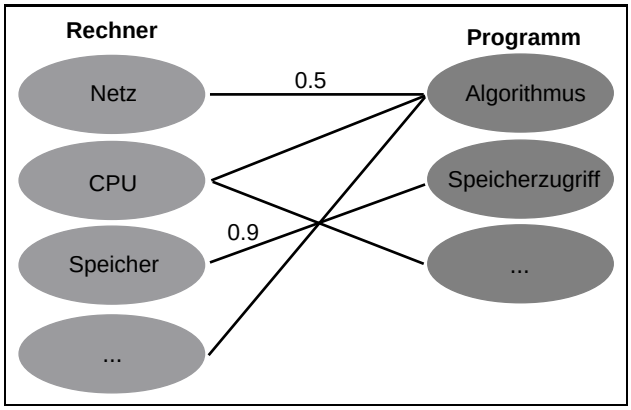


Abbildung 3.3: Die Verbindungen zwischen den Klassen Rechner und Programm bestehen zwischen den einzelnen Klassenelementen.

In dem Schaubild ist zu erkennen, dass die Verbindungen gewichtet wurden. Durch die Gewichte soll die Stärke der Abhängigkeit eines Programms von den Komponenten des Rechners dargestellt werden. Diese Verbindungen und Gewichte müssen durch Läufe von Benchmark-Programmen hergestellt werden. Durch diese Läufe ist z.B. herauszufinden, wie stark die Performance eines bestimmten Programms von der Kommunikation und damit vom Netzwerk abhängt. Entscheidend dabei ist, dass diese Verbindungen nicht starr sind, d.h. sie sind nicht entweder vorhanden oder gekappt, sondern können stärker oder schwächer ausgeprägt sein. Daher ist für das Modell erforderlich, diesen Grad an Abhängigkeit auszudrücken.

Ein weiteres Problem ist, dass das Schaubild 3.3 voraussichtlich zu einfach ist. Es sind zwar die Beziehungen zwischen Programm und Rechner zu sehen und diese Beziehungen sind über Messungen aus Benchmarkläufen gewichtet, aber es ist dabei z.B. noch ausser Acht gelassen, wie die Programmläufe ausgesehen haben, also die angesprochenen Laufzeitkonfigurationen. Ein Programm kann sich völlig anders verhalten, wenn es als serieller Job gestartet wird (in diesem Fall wird das ganz deutlich, da die Kommunikation damit ja gar keine Rolle mehr spielt) als bei einem Lauf auf 512 Prozessoren. Das Beispiel muss auch gar nicht so extrem gewählt werden, z.B. bei dem IBM MPP-System *Jump* des JSC kann der Wechsel von 32 auf 64 Prozessoren erhebliche Auswirkungen haben, da damit zwei Nodes angesprochen werden, und somit nicht mehr auf ein einen gemeinsamen Speicher zugegriffen werden kann und externe Kommunikation benötigt wird.

Es wird also vermutlich eine mehrstufige Beziehung aufgebaut werden müssen, wie in Abb. 3.4 angedeutet.

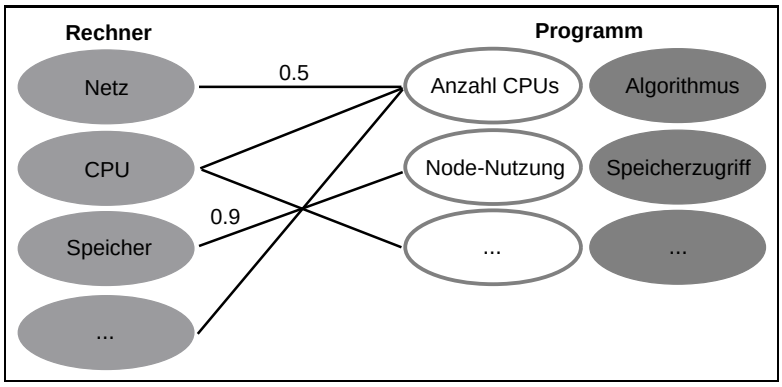


Abbildung 3.4: Die Laufzeitkonfigurationen bestimmen wesentlich die Messungen und müssen in das System integriert werden. Eventuell geschieht diese Einbindung über eine mehrstufige Beziehung.

Eine weitere Frage ist, wie dann in ein solches bestehendes System neue Rechnerkonfigurationen oder auch Programme eingefügt werden können.
Ein Programm sollte auf die einzelnen Klassenelemente abgebildet werden und man sollte dann davon ausgehen können, dass es sich entsprechend seiner Zuordnung verhält. Sollte es Abweichungen im Verhalten geben, wären diese zu untersuchen.
Bei einem Rechner bräuchte man z.B. bei einer neuen CPU Aussagen vom Hersteller (z.B. die CPU ist doppelt so schnell wie der Vorgänger), um die Gewichtung entsprechend anpassen zu können.

3.4 Resultate der Analyse

Ein kompletter Aufbau eines solchen Systems könnte im Benchmarking die Beantwortung der beschriebenen Anfragen, also Schritte, die bisher „von Hand“ aus der Erfahrung der Experten heraus gemacht wurden, automatisieren.
Man könnte dieses Vorgehen auch als „virtuelles“ Benchmarking bezeichnen, da mit einer Benchmark-Suite und der vorhandenen Datenbasis des Systems, also den Klassifizierungen und der Kenntnis über die Verbindungen und Abhängigkeiten sozusagen ein „virtueller Lauf“ auf dem zukünftigen noch nicht vorhandenen Rechner gemacht werden kann.
Die Fragen, die damit verbunden sind lauten: Lassen sich lange Simulationsläufe der Anwendungsprogramme vermeiden? Kann man Benchmarking-Ergebnisse voraussagen, lassen sich also Benchmarking-Läufe simulieren? In dieser Arbeit sollen auf diese Fragen Antworten gefunden werden.

3.4.1 Mögliche Methoden

Für ein solches Benchmarking-Simulations-System werden Methoden benötigt, die es ermöglichen, auf der Grundlage einer Datenbasis, Entscheidungen zu treffen bzw. Schlussfolgerungen zu ziehen und Vorhersagen zu machen.

Performance Prediction Tools

In diesem Umfeld gibt es das Gebiet der Performance Prediction Tools [L4]. Diese Tools dienen dazu, Laufzeiteigenschaften von parallelen Programmen vorherzusagen. Sie werden z.B. bei der Entwicklung von optimierenden Compilern eingesetzt sowie zur Unterstützung der Suche einer geeigneten Rechnerarchitektur für ein paralleles Programm [L1]. Dadurch ist die Verwandschaft zu der Problemstellung dieser Arbeit erkennbar, es ist jedoch so, dass der Ansatz dieser Tools meist darin besteht, dass ein konkretes Programm im Detail instrumentiert wird, um die Vorhersage dann mit Hilfe eines mathematischen Modells vorzunehmen. So fließen nicht nur generelle Programmeigenschaften in die Untersuchung ein.
Für diese Arbeit wird ein allgemeinerer Ansatz gesucht, der das Potential hat, neben dem Schwerpunkt der Prognose eine komplette Simulation der Anfragen an das Benchmarking durchzuführen.

Künstliche Intelligenz

Es wurde bereits festgestellt, dass die Benchmarking-Analyse zur Zeit von Wissenschaftlern, also Experten auf diesem Gebiet, vorgenommen werden. Das bedeutet, dass bei dieser Aufgabe zum einen Expertenwissen benötigt wird - also Kenntnisse, die sich ein Experte auf dem Gebiet des Benchmarking angeeignet hat wie z.B. das Wissen über die Netzabhängigkeit eines Programms - zum anderen auch ein Vorgehen, das diese menschliche Intelligenz voraussetzt. Es gibt keine festen Regeln, denen ein automatisierter Algorithmus zur Prognose folgen könnte, sondern es werden intelligente Konzepte wie z.B. das Schlußfolgern herangezogen. Daher kann die Nutzung von Methoden der Künstlichen Intelligenz (KI) hilfreich sein. Dieser Bereich der Informatik soll im folgenden Kapitel näher betrachtet werden, um einen Lösungsansatz zu erhalten.

3.4.2 Weiteres Vorgehen

Es soll ein System geschaffen werden, aus dem unter Ausnutzung der beschriebenen Zusammenhänge Antworten zu den Anfragen an das Benchmarking gewonnen werden können, also im Grunde Benchmarkläufe simuliert werden können. Der Schwerpunkt der Simulation von Benchmarking-Prozessen soll wie herausgestellt auf der Prognose von Benchmarking-Ergebnissen liegen.
Die Aufgabe der Arbeit zur Entwicklung eines Konzepts für ein solches Benchmarking-Prognose-System gliedert sich in folgende Teile:

1. Es muss zunächst eine Methode gefunden werden, die geeignet ist, eine solche Prognose durchzuführen. Da diese Arbeit in der Regel von Experten durchgeführt wird, wäre es günstig, sich deren Vorgehen zum Vorbild zu machen. Daher erfolgt eine Betrachtung von Methoden aus der KI. Bei der Methode ist darauf zu

achten, ob die beschriebene Gewichtung der Verbindungen berücksichtigt werden kann. Zudem muss untersucht werden, wie das benötigte Wissen in das System eingebracht werden kann, wie beschrieben spielen dabei auch Parameter wie die Laufkonfigurationen eine Rolle.

2. Der Aufbau des Konzepts hängt von der gewählten Methode ab. Es wird sich zeigen, ob für die Realisierung bzw. Tests eine Software-Entwicklung notwendig wird oder ob existierende Software genutzt werden kann. Falls Software für die KI-Methode vorliegt, wird sie auf das Problem angepasst oder konfiguiert werden müssen.
   Die Grundlage für das System wird eine Reihe von Benchmark-Messungen sein. Die Konfigurationen und Ergebnisse der Messungen werden in ein für die Software lesbares Format konvertiert werden müssen. Abhängig von der Methode kann es notwendig werden, zusätzliche Benchmarkläufe zu generieren oder Läufe nach bestimmten Gesichtspunkten auszuwählen.
3. Nach der Auswahl einer Methode muss ein Konzept für das Benchmarking-System entworfen werden. Dabei wird entwickelt, wie die beschriebenen Klassen für das System abgebildet werden und Anfragen an das System gestellt werden können. Dazu wird voraussichtlich eine genauere Klassifizierung der Komponenten (Rechner, Programm) nötig sein, wie es in diesem Kapitel auch schon angesprochen wurde. Dadurch können die Daten, die in das System eingehen, entsprechend eingeteilt und abgefragt bzw. gemessen werden. Die genaue Kenntnis des Programms bzw. des Rechners ist offensichtlich wichtig für eine korrekte Prognose.
   Dabei ist zu beachten, dass je genauer die Klassifizierung vorgenommen wird, also je mehr Werte berücksichtigt werden, desto genauer wird auch die Simulation bzw. Prognose sein. Auf der anderen Seite wächst damit auch die Komplexität an.
4. Das Konzept kann anhand eines Prototyps oder einer Testreihe geprüft werden. Die Ergebnisse sind kontrollierbar anhand vorhandener Ergebnisse, z.B. aus den Benchmarking-Daten der Projekte DEISA und PRACE. Auch Vorhersagen können mit den vorhandenen Messungen simuliert und dadurch überprüft werden.

3.5 Problemanalyse – Kurzzusammenfassung

In diesem Kapitel wurde zunächst analysiert, auf welche Anfragen hin aus dem Benchmarking heraus Antworten gesucht werden. Daraus ergab sich eine Dreier-Beziehung zwischen Programm- und Rechnereigenschaften sowie den Messungen, die Aussagen über diese Verbindung von Programm und Rechner erlauben. Es wurde gezeigt, dass diese verschiedenen Bereiche über ihre Attribute, wie z.B. die Netzgeschwindigkeit zur Kommunikation im Programm, in Bezug gesetzt werden können, und dabei auch auf eventuelle Schwierigkeiten hingewiesen.
Schließlich wurden Ansätze aufgezeigt, die als mögliche Methoden zur Automatisierung der Benchmarking-Prognose dienen können. Es wurde die Entscheidung getroffen, Methoden der Künstlichen Intelligenz genauer zu untersuchen, um auf diesem Weg eine Methode zu finden, die für das Konzept der automatisierten Benchmarking-Prognose genutzt werden kann.

4 Künstliche Intelligenz

In diesem Kapitel sollen Verfahren betrachtet werden, die für das geplante System in Frage kommen. Die Verfahren entstammen, wie im Abschnitt 3.4 auf Seite 22 angesprochen, dem Gebiet der künstlichen Intelligenz, daher soll in dieses Gebiet eine kurze Einführung gegeben werden.

Die Definition der künstlichen Intelligenz (KI) ist nicht einfach, es sollen Prozesse menschlicher Intelligenz auf den Computer übertragen werden. Aber die menschliche „Intelligenz“ ist schon nicht klar definiert, handelt es sich um die Fähigkeit, Problemlösungen zu finden? Umfasst Intelligenz rein kognitive Fähigkeiten oder geht der Begriff darüber hinaus? Besonders schwierig wird die Abgrenzung, wenn auch moderne Begriffe wie emotionale Intelligenz betrachtet werden.
Ein Versuch, Intelligenz zu beschreiben, wurde 1950 von Alan Turing unternommen [L18]. Er entwickelte den *Turing-Test*: In zwei voneinander abgetrennten Räumen befinden sich ein Mensch und eine Maschine. An beide wird eine Reihe von beliebigen Anfragen gestellt, ohne dabei zu wissen, in welchem Raum sich der Mensch bzw. die Maschine befindet. Wenn nun aus den Antworten heraus nicht die Entscheidung getroffen werden kann, in welchem Raum sich der Mensch und in welchem sich die Maschine befindet, kann davon gesprochen werden, dass die Maschine genauso *intelligent* ist wie der Mensch.

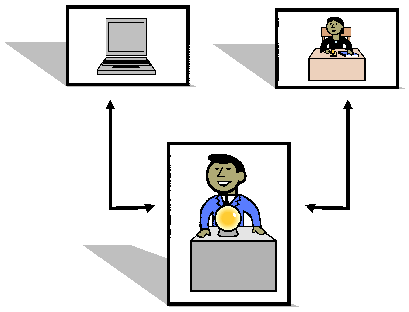


Abbildung 4.1: Der Turing-Test von Alan Turing zur Definition von Intelligenz.

Dieser Test hat natürlich auch Schwächen: Er setzt voraus, dass die Maschine auf die gleiche Art wie der Mensch kommuniziert, obwohl das sicher keine Voraussetzung für Intelligenz ist, zudem ist auch eine menschliche Unterhaltung als solche höchstens ein Teil von Intelligenz, aber umfasst nicht den ganzen Begriff.

Alternativ ist es möglich, eine Liste von Kriterien für KI aufzustellen. Nach [L6] sollten folgende Eigenschaften dazu gehören:

- Die Fähigkeit mit Symbolen zu arbeiten,
- die Fähigkeit zur Wissensrepräsentation,
- die Fähigkeit, Wissen zweckentsprechend einzusetzen und Schlussfolgerungen zu ziehen,
- die Fähigkeit zu Abstraktion und Spezialisierung, sowie zur Anpassung,
- Lernfähigkeit, sowie die Fähigkeit, auch mit ungenauen Informationen Entscheidungen zu treffen,
- Mustererkennung, d.h. Wahrnehmung der Umwelt und Kommunikation.

Über diese Eigenschaften hat man ebenfalls eine mögliche Definition der KI erhalten.
Abschließend soll noch eine sehr knappe Definition von Rich und Knight zitiert werden:

Definition 3 *Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.[L12]*

Ein Zweig der Forschung an KI beschäftigt sich mit dem Versuch, die menschliche Intelligenz zu simulieren bzw nachzubilden, also eine kreative Intelligenz zu schaffen, der man auch einen Begriff wie 'Bewusstsein' zusprechen könnte. Dieses Teilgebiet wird als *starke KI* bezeichnet.
Zum anderen gibt es die Themenbereiche der *schwachen KI*. Hier sollen Anwendungsprobleme gelöst werden, für die man hofft, sich die Abläufe, wie Intelligenz funktioniert, zu Nutze machen zu können. Dem entspricht auch die obige Definition von KI von Rich und Knight.
Diese schwache KI ist inzwischen weiter verbreitet und kommt u.a. in folgenden Gebieten zum Einsatz:

- Spracherkennung, Bildverarbeitung, Mustererkennung,
- Diagnose, Prognose, Simulation,
- Robotik,
- automatisches Programmieren.

Die Methoden, die für diese Arbeit interessant sind, entstammen der praxisorientierten, schwachen KI und sollen im Folgenden näher betrachtet werden. Untersucht werden dabei zunächst Wissensbasierte Systeme, da die Möglichkeit besteht, dass mit einem solchen System das Wissen der Experten im Bereich Benchmarking maschinell nutzbar gemacht werden kann. Dem gegenübergestellt werden Neuronale Netze, bei denen die Datenbasis der verschiedenen Benchmark-Suiten ausgenutzt werden könnte.

4.1 Wissensbasierte Systeme

Ein Teilgebiet der KI sind Wissensbasierte Systeme [L11]. Als Wissensbasierte Systeme bezeichnet man Systeme, bei denen eine Wissensbasis die grundlegende Rolle spielt, diese ist aber abgekoppelt vom eigentlichen Algorithmus. Somit kann bei einem solchen System die Wissensbasis ausgetauscht werden ohne in das umgebende Programm eingreifen zu müssen.

Ein Wissensbasiertes System besteht also aus einer Wissensdatenbasis mit einem umgebenden Framework. Diese beiden Komponenten sollen im folgenden genauer betrachtet werden.

4.1.1 Wissensdatenbasis und Ontologien

Die Wissensdatenbasis ist das grundlegende Element eines Wissensbasierten Systems. Das Wissen entstammt je nach Anwendung aus unterschiedlichen Quellen. Nimmt man ein System zur Diagnose von Krankheiten als Beispiel, könnte das Wissen aus erfolgreich gestellten Diagnosen gewonnen werden sowie aus medizinischer Fachliteratur oder von Medizinern direkt. Es handelt sich dabei nicht um eine reine Datenmenge von Objekten auf denen gearbeitet werden soll, sondern auch um Wissen in Form von Regeln zwischen diesen Objekten.
Bleibt man beim Beispiel der medizinischen Diagnosestellung, sind in der Datenbasis also die Namen und Symptome von Krankheiten abgelegt, aber auch die Regeln, mit denen eine Krankheit erkannt werden kann. Z.B. soll aus den Symptomen Fieber und Ausschlag die Diagnose Masern gefolgert werden. Um Aussagen dieser Art machen zu können, bedient man sich der *Prädikatenlogik*. Mit Hilfe von Symbolen, Junktoren und Quantoren können in dieser Logik die entsprechenden Aussagen dargestellt werden, die obige Aussage könnte darin wie folgt aussehen:

$$\forall x\, Fieber(x) \wedge Ausschlag(x) \Rightarrow Masern(x)$$

Wissensbasen werden häufig in relationalen Datenbanken abgespeichert. Wichtiger als diese Technik ist aber, dass das Wissen möglichst schnell und sicher abgerufen werden kann, dass es also gut strukturiert ist und die Beziehungen zwischen den Objekten klar definiert sind. Dazu können über die technischen Methoden einer Datenbank hinaus Konzepte wie die im Folgenden beschriebenen Ontologien hilfreich sein.

Ontologie in der Philosophie

Der Begriff Ontologie stammt ursprünglich aus der Philosophie. Er bezeichnet dort die Lehre vom Seienden.

Meyers Lexikon definiert den Begriff wie folgt [W17]:

Definition 4 ***Ontologie [griechisch] die****, philosophische Disziplin, die nach der grundlegenden Struktur des Seienden und der Wirklichkeit fragt; sie untersucht dabei auch verschiedene Klassifikationen der existierenden Dinge; des Weiteren formuliert und vergleicht die Ontologie verschiedene Modelle, die die ontologischen Annahmen unserer Sprache theoretisch nachbilden und erklären oder diese durch geeignetere Annahmen ersetzen. [...]*

Ontologien in der Informatik

Ontologien in der Informatik greifen diesen philosophischen Ansatz der Struktur und Klassifikation von Begriffen bzw. Dingen auf. Ontologien dienen in der Informatik dazu, Wissen zu repräsentieren und sie unterstützen bei der Strukturierung von Daten. Zudem helfen sie beim Datenaustausch und ermöglichen es, Wissensbestände zusammenzufügen [W18]. Wie das durch das Konzept der Ontologien möglich wird, wird nun deutlich werden.

Der Mensch nutzt z.B. bei der Suche nach Informationen auf Webseiten oder in Bibliotheken sein Kontextwissen. Doppeldeutige Begriffe werden so für den jeweiligen Themenbereich eindeutig. Soll hingegen eine solche Informationssuche automatisch von einem Rechner, also einer Maschine, durchgeführt werden, so muss diesem das Kontextwissen über das betroffene Themengebiet mitgegeben werden. Es werden also Metadaten benötigt, die die Bedeutungen der Begriffe in den Dokumenten und auch die Zusammenhänge zwischen den Begriffen beschreiben. Diese Metadaten werden als *Ontologie* bezeichnet.
Die bekannteste Definition des Begriffes Ontologie in der Informatik hat T.Gruber gegeben [W19, W20]:

Definition 5 *An Ontology is a formal specification of a shared conceptualization of a domain of interest.*

Eine Ontologie zeichnet sich also dadurch aus, dass sie maschinell interpretierbar ist und einen gemeinsamen Konsens von Begrifflichkeiten festlegt. In der Regel bezieht sie sich nur auf ein bestimmtes Themengebiet. Dadurch wird deutlich, dass man in der Informatik von dem Plural Ontologien sprechen kann, was in der klassischen philosophischen Ontologie nicht möglich ist.
Daher werden Ontologien auch in Typen unterteilt:

- upper model - auch Top-Level Ontologie, domänenunabhängig
- middle model - mittlere Spezialisierung
- domain model - stark spezialisierte Ontologie

Bei der Anwendung in einem konkreten Wissensbasierten System wird es sich in den meisten Fällen um eine mittlere oder stark spezialisierte Ontologie handeln.

Um eine Ontologie aufzubauen, braucht man eine zugrundeliegende Menge von Worten, ein Lexikon, mit denen reale Dinge bezeichnet werden können. Zwischen den Begriffen müssen dann Beziehungen etabliert werden, die Semantik. Zusätzlich können noch weitere Regeln für das System definiert werden.
Einige wichtige Begriffe zum Aufbau einer Ontologie werden im Folgenden kurz erläutert:

- Die Beschreibung gemeinsamer Eigenschaften von *Begriffen* wird als *Klasse* bezeichnet. Es ist möglich, eine Struktur von über- und untergeordneten Klassen zu bilden.

- *Instanzen* repräsentieren Objekte in der Ontologie und stellen das zur Verfügung stehende Wissen dar. Diese werden anhand vorher definierter Klassen erzeugt.

- *Relationen* beschreiben Beziehungen zwischen den Instanzen.

- Es ist möglich, Eigenschaften der Klassen und Relationen zu *vererben*. Dabei werden alle Eigenschaften an das zu vererbende Element weitergegeben. Auch Mehrfachvererbung bei Klassen ist möglich.

- *Axiome* sind Aussagen innerhalb der Ontologie, die immer wahr sind. Diese werden dazu verwendet, Wissen zu repräsentieren, das nicht abgeleitet werden kann.

Beispiel

Anhand der Beispielontologie 4.2 auf der nächsten Seite kann der Aufbau einer Ontologie nachvollzogen werden. Die dicken Pfeile kennzeichnen dabei abgeleitete Klassen. Man erkennt z.B. die Klasse *Person*, in der unter anderem die Namen einer Person abgelegt werden können, abgleitet von dieser Klasse ist dann die Klasse der *Mitarbeiter*, also Personen, die zu einer Firma bzw. Forschungseinrichtung gehören. Über die schmaleren Pfeile werden diese Beziehungen erzeugt, der Mitarbeiter *ist Teil von* dem Forschungszentrum.
Die Abbildung kann nur einen Ausschnitt einer solchen Ontologie zeigen. Die Attribute, die jeder Klasse zugeordnet werden können, sind nur exemplarisch für die Klasse Person aufgeführt. Zudem ist es noch vorstellbar, dass die Beziehungen auch wechselseitig aufgebaut werden, d.h. also zur Beziehung *benutzt* würde *wird benutzt* ergänzt, um Anfragen in beide Richtungen stellen zu können.

Die Abbildung 4.3 zeigt Instanzen dieser Ontologie. Man erkennt, dass eine Instanz auch wiederverwendet werden kann, in diesem Fall rechnen mehrere Personen auf dem Rechner *Jugene*.

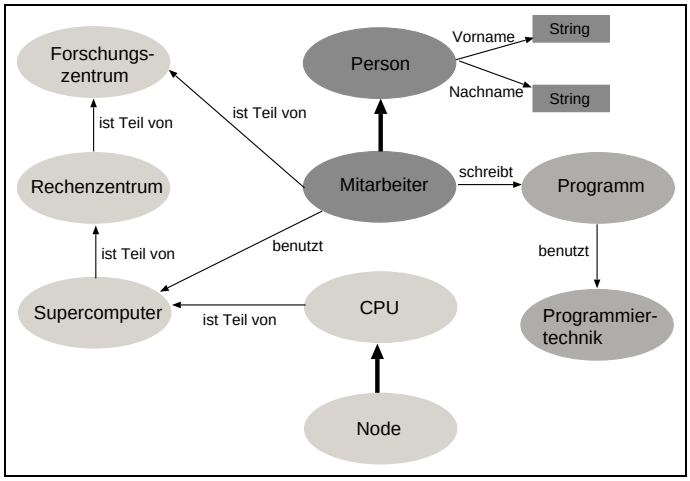


Abbildung 4.2: Beispiel einer Ontologie im Bereich eines Rechenzentrums.

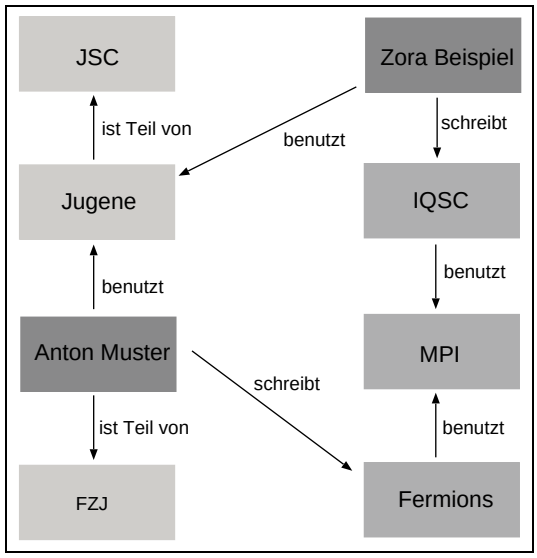


Abbildung 4.3: Mögliche Instanzen der Beispiel-Ontologie.

An diesem Beispiel ist auch zu sehen, dass der Aufbau einer Ontologie nicht immer sehr einfach ist, insbesondere da oft das Themengebiet schwer abzugrenzen ist, in dem gegebenen Beispiel wird das deutlich durch den Übergang von der Hardware, also dem Rechner, zu der Beschreibung von Software und Programmen bzw. Programmiertechniken. Sind diese beiden Elemente in einer Ontologie zu vereinen oder sollten zwei generiert werden? Diese zwei wären dann wiederum sehr spezialisiert, aber wo kann eine Abgrenzung gemacht werden ohne das komplette Gebiet der Informatik in der Ontologie abbilden zu müssen? Noch deutlicher wird diese Problematik, wenn man beachtet, dass

in der Beispielontologie auch Nutzer der Rechner sowie Institutionen aufgeführt sind. An dieser Stelle wird die Informatik sogar verlassen, der Entwickler muss entscheiden, wie detailliert er diese Bereiche beschreiben kann oder ob es schon existierende Ontologien gibt, die integriert werden sollten. Insbesondere muß er sich über das Gebiet, das er beschreiben möchte im Klaren sein, zur Beschreibung des Gebiets *Benchmarking* wären voraussichtlich die Klassen *Rechner*, *Messung* und *Programm* zu schaffen während die Klasse *Mitarbeiter* keine Rolle spielen sollte.

Eigenschaften und Anwendungsgebiete

Zwei wichtige Eigenschaften und die sich daraus ergebenen Anwendungsgebiete finden sich im Folgenden:

Automatisches Schließen Durch die Vererbung und Definition von Axiomen ist es möglich, dass eine Maschine aus den zur Verfügung gestellten Informationen automatische Schlüsse ziehen kann. Anwendung findet das zum Beispiel in der Biologie. Bei Sequenzähnlichkeit zweier Proteine schließt man von der Funktion des bekannten Proteins auf die Funktion des unbekannten Proteins [W21]. Im Bereich des Benchmarking ließe sich die Frage stellen, ob aus einer Programmähnlichkeit ein gleiches Programm-, also z.B. Laufzeitverhalten, folgt. Diese Fähigkeit kann natürlich auch in einem Wissensbasierten System zum Einsatz kommen, wenn das umgebende Framework diese Information entsprechend nutzt, so dass Wissensbasierte Systeme eines der wichtigsten Einsatzgebiete von Ontologien bilden.

Kommunikation Ein wichtiger Anwendungsbereich ist aber auch die Kommunikation vom Maschinen untereinander wie sie z.B. bei der Nutzung von Webdiensten zum Tragen kommt. Beide Seiten müssen eine gemeinsame Interpretation der Daten haben, was durch Ontologien ermöglicht wird.
Eine Rolle spielt hier das *Semiotische Dreieck* (s. Abb. 4.4[L17]), durch das zum Ausdruck kommt, dass ein Symbol, das benutzt wird, die dahinter stehende Bedeutung evtl. nicht vollständig oder eindeutig wiedergeben kann. Der Bezug zwischen Symbol und Gegenstand wird erst über die dahinterstehende Vorstellung hergestellt. Ein beliebtes Beispiel ist hier das Wort Jaguar, das für ein Auto oder auch für ein Tier stehen kann.

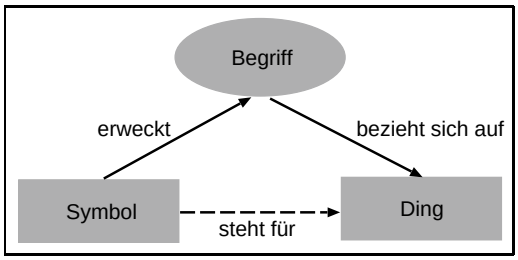


Abbildung 4.4: Das *Semiotische Dreieck* macht deutlich, dass ein Symbol erst über die Vorstellung, also den Begriff dahinter, einen Gegenstand eindeutig kennzeichnet.

Es ist zur Verständigung von Maschinen (und auch Menschen) also wichtig, sich auf eine gemeinsame Ontologie geeinigt zu haben.

Ein Beispiel, wo Ontologien zur Kommunikation zur Anwendung gebracht werden, ist das *Semantische Netz* (engl. Semantic Web). Es ist eine Erweiterung des aktuelle WWW, die Idee des semantischen Webs ist es, die Webinhalte so aufzubereiten, dass Maschinen sie automatisch verarbeiten und „verstehen“, das heißt die Bedeutung erfassen können. Eine Suchmaschine soll zum Beispiel nicht nur nach dem Wort „Bauer“ suchen, unabhängig von dem Zusammenhang, sondern wissen, dass es sich um einen Namen handelt und entsprechend nur solche Einträge zurückliefern, die sich auf Personen mit dem Namen Bauer beziehen und nicht auf den Beruf. Ebenso ist vorstellbar, dass Daten miteinander in Beziehung gesetzt werden, z.B. Wetterdaten mit Informationen über Gesundheit.
Dabei sind Ontologien das entscheidene Hilfsmittel, um Daten zu interpretieren und miteinander in Beziehung zu setzen.

Aktuelle Forschung an Ontologien

Ontologien sind in der Informatik noch ein sehr aktuelles Forschungsgebiet, es existieren noch eine Reihe von Problemen, die es zu lösen gilt:

- Standardisierung - Zur Zeit existieren verschiedene „Standards“.
- Wiederverwendbarkeit - Viele Ontologien sind sehr spezialisiert und können somit nicht einfach auf verschiedene Anwendungsgebiete übertragen werden.
- Überspezialisierung - Die Neigung zur starken Spezialisierung führt auch beim Ontologieaufbau zu Problemen, es muß ein Mittelweg gefunden werden zwischen Genauigkeit und Einsetzbarkeit.
- Aktualisierung - Eine Ontologie kann wiederverwendetet werden, was passiert jedoch, wenn die Original-Ontologie verändert wird?
- Wahrheitskonsens - Hier kommen kulturelle und sprachliche Unterschiede zwischen verschiedenen Nationen oder Kulturkreisen zum Tragen.

Modellierung von Ontologien

Bei Ontologien spielt auch der Begriff Software-Lebenszyklus bzw. Ontologie-Lebenszyklus eine Rolle. Um eine Ontologie aufzubauen wird man in etwa folgende Schritte einhalten:

- Anforderungsspezifikation (Aufgaben, Ziele und Benutzerkreise),
- Informelle Repräsentation (in natürlicher Sprache),
- Entwickung einer ersten Klassifikation,
- Konzeptualisierung und Formalisierung,

- Integration evtl. vorhandener Ontologien,
- Bewertung und Dokumentation,
- Wartung.

Zur Erstellung von Ontologien existieren Werkzeuge, z.B. Editoren wie den Ontologie-Editor Protege [W22].

Zusammenfassend ist zu sagen, dass eine Ontologie somit Zusammenhänge zwischen Begriffen herstellt. Es ist also möglich, sie als grundlegende Technik für die Wissensbasis in einem Wissensbasierten System oder *Expertensystem* zu nehmen.

4.1.2 Expertensysteme

Ein Expertensystem ist ein wissenbasiertes System, die Begriffe sind in der Literatur nicht immer klar voneinander abgegrenzt. Es wird zur Lösung von Aufgaben eingesetzt, die im allgemeinen Expertenwissen, d.h. Spezialwissen, verlangen. Es zeichnet sich wie bei einem Wissensbasierten System üblich durch eine große Datenbasis als Wissensgrundlage aus, die hier in der Regel von Experten eingegeben wird. In diesem Abschitt soll über die Wissensbasis hinaus auf die Strategien zur Lösungsgenerierung des Systems und das umgebende Framework eingegangen werden.

Es existieren hinsichtlich der Problemlösung unterschiedliche Typen von Expertensystemen.

1. Beim ersten Typ wird die Lösung eines Problems durch Klassifikation vorgenommen, d.h. eine Anfrage an das System führt zu einem Vergleich der Anfrage mit den vorhandenen Datensätzen in der Wissensbasis. Die Problemlösung wird aus dem Fall generiert, der dem aktuellen Problem am nächsten kommt.
2. Ebenfalls zum Einsatz kommen Expertensysteme, bei denen die Einordnung des vorliegenden Problems durch einen Entscheidungsbaum vorgenommen wird. Beim Entscheidungsbaum werden Attribute des Problems abgefragt und je nach Antwort in einen der folgenden Äste verzweigt, bis schließlich ein Blatt des Baumes erreicht wird, das die Problemlösung enthält.
3. Im dritten Typ ist die Existenz des Regelwerks entscheidend, das bestimmt, wie mit den Daten aus der Datenbank operiert werden darf, d.h. ob bzw. auf welche Art Schlussfolgerungen möglich sind. Dazu kann die bereits vorgestellte Prädikatenlogik genutzt werden.

Gemeinsam haben diese drei Typen die folgenden Komponenten:

- Die zu Grunde liegende Wissensbasis, wie sie zuvor beschrieben wurde.

- Die Inferenzkomponente, die die Problemlösung auf Basis der Wissensgrundlagen, der Regeln und der Lösungsmethode generiert.
- Eine Wissenserwerbkomponente, über die neues Wissen hinzugefügt oder vorhandenes Wissen verändert werden kann.
- Eine Erklärungskomponente, durch die ein Expertensystem auch die Möglichkeit hat, die Lösung zu erläutern. Auf diese Weise kann der Weg, der zur Lösung geführt hat, nachvollzogen werden.
- Eine Benutzerschnittstelle, die sowohl vom Experten genutzt wird (um das Wissen über vorhandene Fälle bzw. Regeln oder Attribute einzugeben) als auch vom Endbenutzer, um Anfragen an das System zu stellen.

Diese Komponenten und ihre Verbindungen sind in Abbildung 4.5 zu sehen.

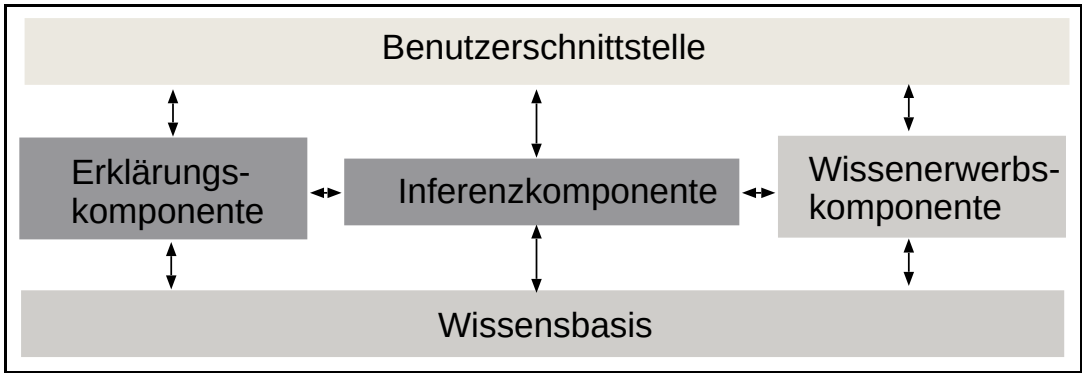


Abbildung 4.5: Der schematische Aufbau eines Expertensystems aus Komponenten zur Wissensakquise -und verwaltung sowie zur Lösungsgenerierung einer Anfrage. Über eine Benutzerschnittstelle sind die einzelnen Komponenten ansprechbar.

Über ein Expertensystem können somit Lösungen zu Anfragen an eine Wissensbasis generiert und diese Datenbasis auch verwaltet werden.

4.2 Neuronale Netze

Ein weiteres wichtiges Gebiet in der KI-Forschung sind die *Neuronalen Netze* [L16, L10]. Im Gegensatz zu Expertensystemen und Ontologien wird Ihnen nicht das Expertenwissen mitgegeben, um darauf basierend eine Problemlösung zu generieren, sondern das System soll selbst Zusammenhänge zwischen Daten erkennen und diese lernen, um dann darauf basierend Lösungen zu präsentieren.

Neuronale Netze werden in den bereits erwähnten Gebieten der Mustererkennung und Bildverarbeitung oder auch Robotik/Steuer- und Regelungstechnik eingesetzt. Zudem finden sie auch Verwendung in der Optimierung und Prognose von Ereignissen.

Die Funktionsweise ist im Prinzip immer gleich, es existiert ein Satz von Eingabevektoren und das neuronale Netz kann als Abbildungsvorschrift auf einen entsprechenden Satz von Ausgabevektoren verstanden werden.

4.2.1 Ursprung in der Biologie

Das Vorbild für neuronale Netze entstammt der Biologie. Eine vereinfachte Darstellung soll im Folgenden den Aufbau von neuronalen Netzen erklären [L15].

Das menschliche Nervensystem ist aus Neuronen, also Nervenzellen und Verbindungen zwischen diesen Neuronen, den Synapsen, aufgebaut. Der Aufbau besteht in der Form eines Netzes, über das durch elektrische Signale Information weitergegeben werden kann. Jedes Neuron kann in diesem Netz mehrere Eingänge haben, über die es Signale empfängt. Sobald im Neuron ein Schwellenwert erreicht wird, „feuert“ das Neuron seinerseits selbst über seinen Ausgang eine elektrische Information.
Zu beachten ist, dass jedes einzelne Neuron für sich genommen über kein großes Wissen verfügt, erst in der Gesamtheit des Neztes wird es möglich komplexe Aufgaben zu erledigen.
Die Verarbeitungsgeschwindigkeit eines Neurons ist dabei gar nicht sehr hoch, die hohe Leistungsfähigkeit des menschlichen Gehirns muß in der enorm hohen Parallelität begründet liegen, mit der die Neuronen arbeiten. Um diese Parallelität zu ermöglichen, muss der Zugriff auf gespeicherte Information global möglich sein.

Eine interessante Beobachtung ist, dass die Anzahl der Neuronen schon zur Geburt eines Menschen festgelegt ist, Neuronen können sich nicht durch Zellteilung reproduzieren. Soll der Mensch also neues Wissen hinzulernen, kann dazu kein neues Neuron dienen. Die Wissensvermehrung geschieht durch die Schaltung neuer Verbindungen zwischen den Neuronen, das bedeuted die Erzeugung von Synapsen, bzw. die Modifikation der Verbindungen, z.B. eine Stärkung oder Schwächung der Verbindung. Die Anzahl von Verbindungen ist also von entscheidener Bedeutung bzgl. der Verarbeitung und Speicherung von Wissen.
Bezüglich des Lernens sind auch psychologische Erkenntnisse hilfreich: Der Mensch lernt z.B. durch *operante Konditionierung*. Das bedeutet, dass, wenn auf richtiges Verhalten auf einen Reiz hin eine Belohnung folgt bzw. auf falsches Verhalten eine Bestrafung, die Verbindung zwischen dem Reiz und der Verhaltensweise gestärkt bzw. abgeschwächt wird.

Künstliche Neuronale Netze (KNN) nehmen sich die beschriebene Arbeitsweise der natürlichen Neuronalen Netze zum Vorbild.

4.2.2 Ein künstliches Neuron

Es soll zunächst die Funktionsweise eines künstlichen Neurons beschrieben werden, aus denen ein KNN zusammengesetzt wird. Die Aufgabe der künstlichen Neuronen entspricht

der in einem natürlichen Neuronalen Netz, das Neuron empfängt Signale und sobald ein Schwellwert übertroffen wird, feuert es seinerseits ein Signal auf das Netz. Der Aufbau eines Neurons ist in Abbildung 4.6 dargestellt. Von links nach rechts sieht man zunächst

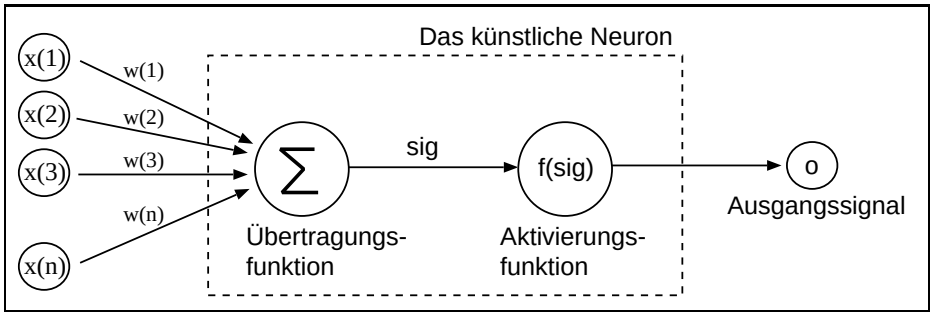


Abbildung 4.6: Die schematische Darstellung eines künstlichen Neurons mit Eingangs- und Ausgangssignalen.

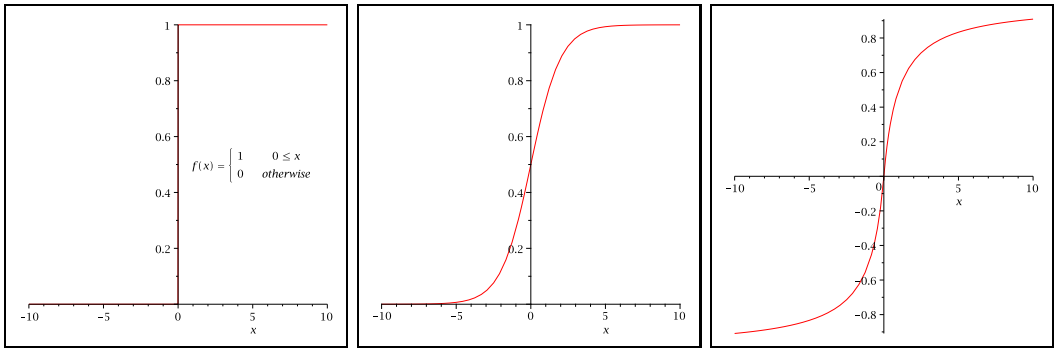
die Eingangssignale *x(i)*, die aus der Umgebung oder von anderen Neuronen kommmen und in das Künstliche Neuron eingehen. Diese Signale werden mit *Gewichten w(i)* versehen, so dass sie unterschiedlichen Einfluss auf das Neuron ausüben. Die tatsächliche Netzeingabe *sig* des Neurons wird mit Hilfe der Gewichtungen über die *Übertragungsfunktion* berechnet, die in der Regel eine gewichtete Summe ist. Das Ausgabesignal *o* des Neurons wird dann durch eine *Aktivierungsfunktion f(sig)* gesetzt.

Die Aktivierungsfunktion

Die Aktivierungsfunktion kann abhängig vom Problem gewählt werden, das das Netzwerk bearbeiten soll. Sie bestimmt, wie stark und auf welche Weise die Eingabewerte des Neurons eine Auswirkung auf das Netz bzw. die Ausgabe des Netzes hat. Häufig werden z.B. eine Schwellenwertfunktion (s. Abb. 4.7(a) auf der nächsten Seite) oder eine Sigmoidfunktion (s. Abb. 4.7(b)) verwendet. Die Aktivierungsfunktion bestimmt das Ausgabesignal, das das Neuron im Netz weitergibt.
Bei der Schwellenwertfunktion feuert das Neuron nur, wenn es eine Aktivität über einem Schwellenwert als Input erhalten hat. In dem abgebildeten Beispiel muss die Aktivität positiv gewesen sein. Ist das der Fall, gibt sie aber die volle Aktivität von 1 an das Netz, es handelt sich also um ein 'Alles oder Nichts'-Prinzip.
Bei der Sigmoidfunktion $f(x) = \frac{1}{1+e^{-x}}$ hat das Ausgabesignal hingegen einen kontinuierlichen Verlauf, sehr große und sehr kleine Eingaben werden geglättet.
Die Elliotfunktion (s. Abb. 4.7(c) auf der nächsten Seite) $f(x) = \frac{x}{1+|x|}$ als drittes Beispiel ist der Sigmoidfunktion ähnlich, hat aber den Vorteil, leichter berechenbar zu sein.

4.2.3 Das Netzwerk

Die einzelnen künstlichen Neuronen werden zu einem gemeinsamen Netzwerk verbunden. Für dieses Netzwerk gibt es sehr unterschiedliche Möglichkeiten der Strukturierung.



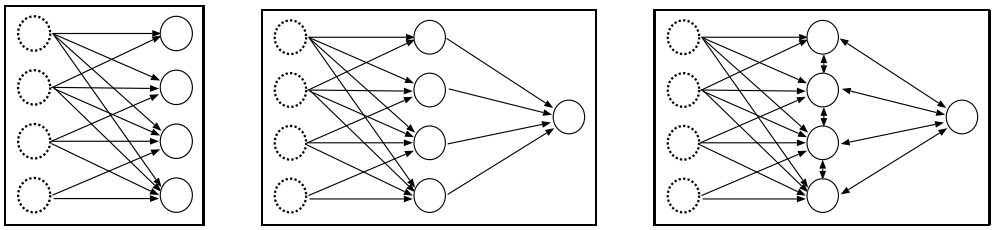
(a) Schwellenwertfunktion (b) Sigmoidfunktion (c) Elliotfunktion

Abbildung 4.7: Beispiele für Aktivierungsfunktionen eines künstlichen Neurons.

Im einfachsten Fall besteht es aus zwei Schichten, der Eingabe- und der Ausgabeschicht, s. Abb. 4.8(a). Die Eingabeschicht hat dabei keine Fähigkeiten, Information zu verarbeiten, sondern gibt diese nur auf das Netz (aufgrund diesen Verhaltens ist sie in der Abbildung jeweils durch gestrichelte Kreise dargestellt). Die Informationen fließen hier ausschließlich von links nach rechts durch das Netz, diese Eigenschaft bezeichnet man als *feed-forward*.

Bei Netzwerken, die aus mehreren Schichten bestehen, werden die innen liegenden Schichten als *verdeckte Schichten* bezeichnet. Das dreischichtige Netz in Abb. 4.8(b) ist ein Beispiel dafür, die Ausgabeschicht besteht hier nur aus einem Element.

Das Netzwerk unter Abb.4.8(c) hat augenscheinlich fast den gleichen Aufbau, es existiert jedoch ein wichtiger Unterschied zum Netzwerk unter (b): die feed-forward-Eigenschaft ist hier nicht vorhanden, es existieren Rückkopplungen innerhalb des Netzes. Ein Netz dieser Art wird als *rekurrentes Netz* bezeichnet.



(a) zweischichtiges Netzwerk (b) dreischichtiges Netzwerk (c) Netzwerk mit Rückkopplungen

Abbildung 4.8: Künstliche Neuronen werden zu Netzwerken verschaltet, drei Beispiele von Netzwerktypen sind hier dargestellt.

In den Beispielen in Abschnitt 4.2.5 auf der nächsten Seite werden verschiedene Netzwerkarchitekturen im Einsatz zu sehen sein.

4.2.4 Das Lernen

Bisher wurden die einzelnen Bestandteile eines KNN vorgestellt, also die Neuronen mit ihren Eigenschaften und beispielhaft mögliche Netzwerkstrukturen. Doch wie erhält das KNN nun Wissen, auf dessen Grundlage es Probleme bearbeiten kann? Die Antwort ist, dass einem KNN das Wissen nicht explizit eingegeben wird, sondern das Wissen ist in Trainingsdaten vorhanden und muss vom KNN über diese Datensätze erlernt werden.

Dabei gibt es verschiedene Strategien, ein KNN zu trainieren. Beim *überwachten Lernen* werden dem Netz eine Reihe an Trainingsdaten präsentiert, zu denen das gewünschte Ergebnis bekannt ist. Wird dieses Ergebnis vom Netz ausgegeben, ist es für dieses Beispiel korrekt trainiert, falls noch eine Abweichung existiert, müssen die Gewichte des Netzes weiter justiert werden.

Beim *nicht-überwachtem Lernen* muss das Netzwerk selbst aus der Eingabe der Trainingsdaten Zusammenhänge extrahieren und die Gewichte entsprechend anpassen. Es orientiert sich also an der „Ähnlichkeit“ der Lernbeispiele und nimmt eine Klassifizierung vor.

Die Anpassung der Gewichte wird jeweils durch eine *Lernregel* vorgenommen. Beispiele auch zu Lernregeln werden in den folgenden Beispielen unter 4.2.5 beschrieben.

4.2.5 Beispiele von Netzwerktypen

Wie schon erwähnt existieren verschiedenartigste Architekturen für KNN. Diese Architekturen entstanden durch unterschiedliche Zielsetzungen, zum Teil wurde neben dem beschriebenen biologischen noch weitere Vorbilder aus der Physik oder Psychologie hinzugezogen. Eine Einteilung der Typen ergibt sich zunächst durch die mögliche Einteilung in feed-forward oder rekurrente Netzwerke. Sie kann auch nach der Anzahl der Schichten oder der Lernmethode - überwacht oder selbstorganisiert - vorgenommen werden.
Aufgrund der Vielfalt der Netzwerktypen [L15] können diese nicht vollständig beschrieben werden, beispielhaft sollen die folgenden Abschnitte einen Einblick geben. Dabei wird zur Einführung das Perceptron als sehr einfaches zweischichtiges KNN herangezogen und eine Erweiterung dieses Konzepts zu einem mehrschichtigen KNN erläutert. Dem gegenübergestellt werden die Self-organizing Maps, mit denen die Lernmethode des nicht-überwachten Lernens gezeigt werden kann.

Das Perceptron

Das Perceptron wurde 1958 von Frank Rosenblatt entwickelt [L13] und ist zwar ein sehr einfaches Netzwerk, kann aber bis heute als Grundlage künstlicher Neuronaler Netze gesehen werden.

Das Perceptron ist ein feed-forward Netzwerk ohne verdeckte Schichten. Da die Eingabeschicht keine Lernfunktionen hat, handelt es sich also um eine Schicht von Neuronen,

als Eingangssignale sind die reellen Zahlen erlaubt. Als Aktivierungsfunktion wird eine Schwellwertfunktion verwendet (x_i sind die n Eingangswerte, w_i die Gewichte):

$$f(x) = \begin{cases} 0 & \text{für } \sum_{i=1}^{n+1} x_i w_i < 0 \\ 1 & \text{für } \sum_{i=1}^{n+1} x_i w_i \geq 0 \end{cases} \tag{4.1}$$

Bei dieser Darstellung wurde bereits ein *Bias* mit einbezogen, der zum einen dazu dient, den Schwellwert modifizieren zu können. Zum anderen ermöglicht er auch das Lernen des Nullvektors als Eingabe. Der Bias ist in Abb. 4.9 zu sehen, er hat in der Eingabeschicht immer den Wert 1 und kann dann wie die anderen Eingabewerte über die Gewichte verändert werden.

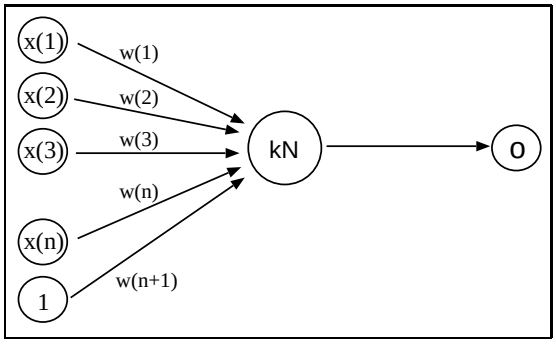


Abbildung 4.9: Darstellung eines Perceptrons mit Bias-Neuron

Die Perceptron-Lernregel besagt, dass bei noch nicht korrekter Bearbeitung eines Eingabevektors (also wenn z.B. das erwartete Ergebnis eine 1 ist, das Netz aber die 0 ausgibt) die (zu Beginn zufällig verteilten) Gewichte nach folgenden Formeln angepasst werden:

$$w_i^{neu} = w_i^{alt} + \Delta w_i \tag{4.2}$$

$$\Delta w_i = \alpha(t - o)x_i \tag{4.3}$$

Dabei sind w_i die Gewichte und Δw_i die Gewichtsänderung, die erzielt werden soll, α eine Lernrate, durch die die Lerngeschwindigkeit beeinflusst werden kann, t die erwartete Ausgabe des Lernvektors, o die tatsächliche Ausgabe und die x_i sind wiederum die Eingangswerte.
Die Lernrate α muss sorgfältig gewählt werden, ist sie zu groß, kann Gelerntes wieder verlernt werden, ist sie zu klein, sind viele Lernschritte nötig. Minsky und Papert bewiesen 1969 in ihrem Buch *Perceptrons* [L9], dass die Perceptron-Lernregel konvergiert, sofern eine Lösung existiert.
Ein Perceptron-Netzwerk kann als Klassifikator dienen, um Eingaben auf die Werte 0 und 1 abzubilden. Es können auch die boolean'sche Funktionen UND sowie ODER trainiert werden. Diese Eigenschaft hat in den 1970er Jahren zu einem zwischenzeitlichen Ende der Euphorie um KNN geführt, da Minsky und Papert ebenfalls in *Perceptrons*

nachwiesen, dass das XOR-Problem hingegen nicht mit einem Perceptron gelernt werden kann, da die Funktion nicht separierbar ist. Man wusste, dass man durch Hintereinanderschalten von mehreren Perceptrons, also einem mehrschichtigen KNN, auch Multilayer-Perceptron genannt, dieses Problem würde Lösen können, hatte jedoch für diese Arten von Netzen noch keinen Lernalgorithmus. Dies ändert sich 1986 mit dem Backpropagation-Algorithmus und führte zu einem erneuten Aufschwung der Forschung an KNN [L14].

Der Backpropagation-Algorithmus

In Abbildung 4.10 ist ein solches Multilayer-Perceptron (MLP) mit einer verdeckten Schicht zu sehen.

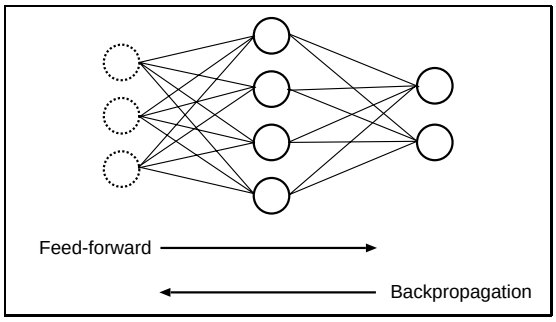


Abbildung 4.10: Darstellung eines Multilayer-Perceptrons, das über Backpropagation trainiert wird.

Als Lernalgorithmus für ein MLP kann wie beschrieben der Backpropagation-Algorithmus genutzt werden. Er arbeitet zunächst wie beim einfachen Perceptron. Es wird aus dem Ergebnis des Trainingslauf und der tatsächlich gewünschten Ausgabe der Fehler berechnet. Dazu wird hier der quadratische Fehler verwendet:

$$F = \sum_i (t_i - o_i)^2 \tag{4.4}$$

Der Fehler wird dann aber wieder über die Ausgabe- zur Eingabeschicht zurückpropagiert. Die Gewichte werden jeweils abhängig von ihrem Einfluss auf den Fehler geändert, so dass der Gesamtfehler reduziert wird:

$$\Delta\omega_{ij} = -\epsilon \frac{\delta F}{\delta \omega_{ij}}$$

Die Gewichtsänderung ergibt sich aus der Minimierung der Fehlerfunktion F nach den Gewichten ω. Es muss also die Kombination von Gewichten gefunden werden, die den niedrigsten Fehlerwert ergibt. Der Algorithmus beruht auf der sogenannten Methode des steilsten Abstiegs (Gradientenverfahren), daher ist eine differenzierbare Aktivierungsfunktion - die ja in der Berechnung der o_i und damit auch in der zu differenzierenden Fehlerfunktion steckt - wie die Sigmoidfunktion Voraussetzung für den Algorithmus.

Self-Organizing Maps

Die Self-Organizing Maps (SOM) oder auch Kohonen-Maps nach ihrem Entwickler Teuvo Kohonen [L7] sind ein Typ Neuronaler Netze, in dem das Lernen ohne Überwachung, also ohne Lehrer oder Bewerter stattfindet.
Es handelt sich im Prinzip um ein zweischichtiges feed-forward Netzwerk, die Eingabeschicht hat auch die gleiche Aufgabe wie in den bisherigen Beispielen. Die zweite Schicht ist hingegen anders aufgebaut, da innerhalb dieser Schicht jedes Neuron einen genau festgelegten Platz hat. Die zugrunde liegende Idee ist die Erkenntnis aus der Biologie, dass im Gehirn Eingangssignale so abgebildet werden, dass ähnliche Reize nahe beieinander liegen. Die zweite Schicht bildet also eine *Karte*, s. Abb. 4.11. Jedes Neuron in

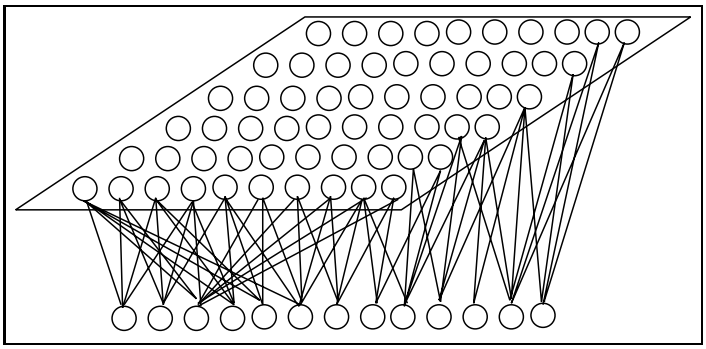


Abbildung 4.11: Darstellung einer Self-Organizing Map - die vollständige Vernetzung zwischen Eingabeschicht und Kohonen-Karte wird im Bild nur angedeutet

dieser Schicht enthält einen Vektor mit Gewichten, der vor dem Trainieren des Netzes mit zufälligen Werten belegt ist. Die Größe dieses Vektors entspricht dem Eingabevektor. Das Training geschieht, indem die Gewichte der Neuronen in der Kohonen Map mit dem Trainingsvektor verglichen werden. Als Ähnlichkeitsmaß wird in der Regel das euklidische Distanzmaß verwendet:

$$\sqrt{\sum_{i=1}^{n}(x_i - w_{ij})^2} \tag{4.5}$$

Das Neuron, das diesen Vergleich am Besten besteht, wird als *Best Matching Unit* (BMU) bezeichnet. Die Gewichte der BMU werden angepasst und auch die Nachbarn dieses Knotens werden so verändert, dass sie dem BMU mehr entsprechen. Für die Anpassung der Gewichte wird die *Hebbsche Lernregel* verwendet, die besagt, dass Verbindungen zwischen Neuronen verstärkt werden sollen, wenn diese gleichzeitig aktiv sind. Auch diese Lernregel entstammt biologischen Beobachtungen. Sie sieht wie folgt aus:

$$\Delta w_{ij} = \alpha \cdot a_i \cdot o_j \tag{4.6}$$

Dabei ist Δw_{ij} die Änderung der Verbindungsstärke der Neuronen, α wiederum die Lernrate, a_i die Aktivierung von Neuron i und o_j Ausgabe von Neuron j, das mit Neuron i

verbunden ist.

Auf diese Weise wird das Netzwerk mit einer grossen Anzahl an Lernvektoren trainiert. Dieses Training endet, sobald die Karte stabil ist. Dann kann ein neuer Vektor, der dann aber lückenhaft sein darf, an die Stelle in das Netz eingefügt werden, an die er am Besten passt. Auf diese Weise werden die fehlenden Komponenten des Vektors angenähert.

4.2.6 Modellierung eines KNN

Wie durch die vorangegangen Kapitel deutlich geworden, ergeben sich bei der Modellierung von KNN die folgenden Schritte:

- Zunächst müssen die Daten untersucht werden, wie können Sie dem Netz in der Eingangsschicht präsentiert werden?
- Was möchte der Entwickler erreichen, sollen die Daten z.B. klassifiziert werden, d.h. soll ein Muster erkannt werden, oder müssen die Daten bewertet werden, z.B. hinsichtlich ihrer Güte?

Daraus wird sich die Architektur des Netzwerkes ergeben, die Anzahl der Neuronen und der Schichten sowie die Definition der Aktivierungsfunktion.
Danach muss das Netzwerk trainiert werden, die Festlegung der Trainingsart sowie die Definition eines ausreichenden Trainings- und auch eines Testsatzes ist somit erforderlich. Dabei ergeben sich oft zwei Probleme: das Netz wird zu wenig trainiert und liefert nicht die gewünschten Ergebnisse oder es wird sogar zu sehr, d.h. zu speziell auf die Trainingssätze hin trainiert, so dass es das Gelernte nicht mehr generalisiert.

4.3 Unscharfe Werte

Sowohl bei Expertensystemen als auch bei Künstlichen Neuronalen Netzen kann es erforderlich sein, dass das System mit unscharfen Werten arbeitet, wie z.B. der Aussage, dass ein Benchmark „sehr gut“ gelaufen ist. Als *unscharf* bezeichnet man somit ungenau bereitgestelltes bzw. ungenau bekanntes Wissen.

Durch die *Fuzzy-Logik* wird es ermöglicht, dieses ungenaue Wissen zu spezifizieren. Die Boolesche Logik (0,1) wird erweitert, so dass auch nicht konkrete Angaben wie „ein bisschen“ oder „sehr“ durch Zwischenwerte ausgedrückt werden können. Diese Logik ist jedoch abzugrenzen von dem Begriff der Wahrscheinlichkeit.

Auf der Fuzzy-Logik basierend gibt es die Fuzzy-Inferenz-Systeme, die es möglich machen, aus unscharfen Informationen mit der Fuzzy-Logik und vorgegebenen Regeln Schlußfolgerungen zu ziehen. Die Regeln sind dabei Wenn-Dann Konstrukte.
Durch die in 4.12 auf der nächsten Seite abgebildete *Fuzzyfunktion* werden Temperaturbereiche dargestellt, durch die ausgedrückt werden kann, dass ein Raum z.B. schon etwas

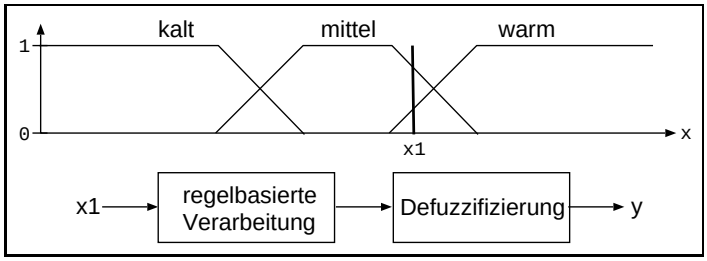


Abbildung 4.12: Ein Fuzzy-Inferenz-System mit Fuzzyfunktion.

warm ist, sich aber eher noch im Bereich der mittleren Temperatur befindet. Diese *fuzzyfizierte* Eingabe kann mit Regeln der Art „Wenn Temperatur=*warm* dann Aktion=*nach unten regeln*“ bearbeitet werden, das wird durch die Fuzzyfunktion ermöglicht.

4.3.1 Expertensysteme und Unschärfe

Am Beispiel über die Diagnose von Masern unter Abschnitt 4.1.1 auf Seite 27 ist zu sehen, dass auch in Expertensystemen die Fähigkeit vorhanden sein muß, Fälle zu definieren, in denen regelbasierte Schlußfolgerungen mit einer Unschärfe versehen sind. Es ist klar ersichtlich, dass die Diagnose *Masern* gestellt werden kann, sie aber nicht völlig sicher ist, da die Symptome Fieber und Ausschlag auch auf andere ähnliche Krankheiten passen wie z.B. Windpocken. Um Unschärfen dieser Art behandeln zu können, wird das System mit Regeln aus der *Fuzzy-Logik* zusätzlich zur Prädikatenlogik aufgebaut.

4.3.2 Neuro-Fuzzy-Systeme

Um in künstlichen Neuronalen Netzen mit unscharfen Daten zu arbeiten, können Fuzzy-Inferenz-Systeme (FIM) mit den KNN verbunden werden, man spricht dann von Neuro-Fuzzy-Systemen [L2]. Die Kombination von FIM mit KNN kann auf zwei Arten vorgenommen werden:

1. Systeme mit kooperativer Kopplung
   Hier arbeiten KNN und FIM unabhängig voneinander. Das Neuronale Netz optimiert z.B. Parameter des KNN. Dabei kann zum einen ein FIM mit unscharfen Mengen vorgegeben werden. Die Regeln für dieses FIM werden dann jedoch vor dem Einsatz mit einem KNN erlernt. Um im Beispiel der Temperaturregelung zu bleiben, sollte das System durch die Trainingsdaten z.B. lernen, dass bei kalten Temperaturen die Temperatur nach oben geregelt werden muss.
   Zum anderen kann ein FIM vorgegeben werden, in dem die Regeln bekannt sind, dann müssen mit Methoden der KNN die Grenzen der unscharfen Mengen abgesteckt werden.
   Es ist auch möglich, Parameter oder Verbindungen eines KNN mit einem FIM einzustellen. Die Gewichte oder die Eingabewerte eines solchen KNN können unscharf sein.

2. Hybride Systeme
   Hier sind die beiden Systeme FIM und KNN miteinander verwoben, es ist also nicht nur ein System vor das andere vorgeschaltet wie beim Beispiel der Temperaturregelung. Es entsteht ein System mit Lernfähigkeit und Regeln, die nachvollzogen werden können.

4.4 Künstliche Intelligenz – Kurzzusammenfassung

In diesem Kapitel wurde zunächst eine Einführung in den Begriff der Künstlichen Intelligenz gegeben und verschiedene Definitionen vorgestellt.
Daraufhin wurden Verfahren der Künstlichen Intelligenz beschrieben, die für das geplante Benchmarking-Prognose-System in Frage kommen. Es handelte es dabei zum einen um Wissensbasierte Systeme, in denen durch Fakten und Regeln dem System eine Wissensgrundlage gegeben wird, anhand der Entscheidungen getroffen werden können. Zum anderen wurden Künstliche Neuronale Netze erläutert, bei denen Entscheidungen aus einem vorhandenen Datensatz heraus gefunden werden, der die Regeln implizit enthält. Als dritter Punkt wurde eine Einführung in Fuzzy-Syteme gegeben, um eine Möglichkeit zu bieten, mit unscharfen Daten zu arbeiten.
Eine Entscheidung für eine der Methoden wird im anschließenden Kapitel erfolgen.

5 Auswahl eines Verfahrens

Eines der im Kapitel 4 beschriebenen Verfahren aus der Künstlichen Intelligenz soll Grundlage für das Benchmarking-Prognose-Modell werden. Um das geeignete Verfahren auszuwählen, sollen die Methoden in diesem Kapitel miteinander verglichen werden. Danach werden noch einmal die Anforderungen an das zu modellierende System aufgezeigt, um dann zu einer Entscheidung hinsichtlich des Verfahrens zu kommen.

5.1 Allgemeiner Vergleich der Verfahren

Es soll noch einmal kurz aufgezeigt werden, wodurch sich die einzelnen betrachteten Verfahren auszeichnen:

- Wissensbasierte Systeme simulieren die Arbeit eines Experten. Sie sind im Wesentlichen gekennzeichnet durch einen grossen Wissensdatenbestand, das dem System mitgegeben wird. Es existieren Regeln, über welche die einzelnen Wissenselemente miteinander in Beziehung gesetzt werden können. Auf diese Weise ist es in diesem System möglich, Schlussfolgerungen vorzunehmen. Die Wissensdatenbank ist also explizit durch das Expertenwissen aufgebaut und daraus ergibt sich die Problemlösung. Dieser Ansatz ist vergleichbar mit dem Ansatz der Performance Prediction, die in Kapitel 3 angesprochen wurde. Durch die zu schaffenden Regeln wird ein Modell für die Benchmarks entwickelt.

- Künstliche Neuronale Netze nehmen sich biologische Zusammenhänge zum Vorbild. Bei Neuronalen Netzen ist zu Beginn kein Wissen innerhalb des Netzwerkes vorhanden. Das Wissen gelangt in das System durch die Lernfälle, mit denen das Netzwerk trainiert wird. Nach dem Lernprozess ist die Problemlösung möglich. Diese Methode ist mit der Möglichkeit von stochastischen Auswertungen von Benchmarkläufen vergleichbar, das Neuronale Netz übernimmt die mathematische Abbildung.

- Ergänzend ist zu erwähnen, dass Fuzzy-Logik sowohl in Expertensystemen wie auch in Neuronalen Netzen genutzt werden kann, um mit unscharfen Daten arbeiten zu können.

In der folgenden Tabelle werden die Eigenschaften und Vor- und Nachteile der Systeme im vergleichenden Überblick gezeigt:

Expertensysteme mit Ontologien	**Künstliche Neuronale Netze**
Experten*wissen* ist im System vorhanden	Kein *Wissen* direkt im System
Es sind ein *Regel*system und *Zusammenhänge* zwischen den Daten vorhanden	Die *Regeln* und *Zusammenhänge* stecken implizit verteilt in den Lernbeispielen
Der *Wissenserwerb* ist evtl. schwierig (komplex,schwer fassbar)	*Wissenserwerb* nicht nötig
Lernfähigkeit: Es werden keine neuen Zusammenhänge erkannt, das System weiss nicht mehr als der Experte	*Lernfähigkeit*: Zusammenhänge werden selbstständig erkannt, evtl. auch bisher unbekannte
Lernfähigkeit: Das System ist lernfähig über die Wissenerwerbskomponente	*Lernfähigkeit*: Kein direktes Wissen in das System einbringbar
Schlußfolgern ist durch die vorhandenen Regeln möglich	*Schlußfolgern* ist keine Eigenschaft eines KNN
Die Lösungsfindung kann *nachvollzogen* werden	Die Lösungsfindung ist nicht *nachvollziehbar*
Generalisierungsfähigkeit: je nach Umsetzung stärker oder schwächer	*Generalisierungsfähigkeit*: Das aus der Trainingsmenge erlernte Wissen kann auf die Testmenge übertragen werden
Fehlertoleranz: Eine fehlerhafte Regel wird vom System nicht entdeckt	*Fehlertoleranz*: über die Menge an Trainingsdaten werden Fehler in den Daten geglättet
An neue Gegebenheiten kann das System evtl. durch Ergänzung der Regeln *angepasst* werden	An neue Gegebenheiten ist das System evtl. nur *anpassbar* durch einen erneuten Lernprozess
Einsatzgebiete: • Diagnosesysteme • Steuer- und Regelungstechnik • Optimierung/Prognose/Simulation • Planung • Therapie • ...	*Einsatzgebiete*: • Bildverarbeitung und Mustererkennung • Robotik • Optimierung/Prognose/Simulation • Zeitreihenanalyse (z.B. Aktien) • Data-Mining • ...

5.2 Die Verfahren unter Berücksichtigung des vorliegenden Problems

5.2.1 Die Anforderungen aus dem Benchmarking

Aus dem Kapitel 2 *Benchmarking* hatte sich ergeben, dass ein System gesucht wird, mit dem Zusammenhänge erkannt werden können zwischen einem Programm mit seinen Laufparametern und dem Rechner, auf dem es laufen soll. Als verbindenes Element waren die Messwerte aus Benchmarking-Läufen identifiziert worden.
Als entscheidend hatte sich zudem herausgestellt, dass Verbindungen zwischen den beiden Klassen nicht als 0 (nicht vorhanden) oder 1 (vorhanden) dargestellt werden dürfen, sondern dass diese Verbindungen auch unterschiedlich stark ausgeprägt sein können. Als Problem wurde die Integration der Laufzeitparameter in ein solches System angesehen.

5.2.2 Vergleich der Verfahren unter Berücksichtigung des Benchmarkings

Es wird sich um ein Simulations- bzw. Prognosesystem handeln. Dies sind beides Einsatzgebiete der behandelten Verfahren.

Für ein Expertensystem spricht, dass das Wissen, das über die Aktivitäten des JSC zum Thema Benchmarking bereits vorhanden ist, genutzt werden kann. Es kann in das System eingebracht werden, und die Hoffnung wäre, dass die Arbeit des Experten simuliert werden kann. Darin liegt aber auch direkt ein Nachteil des Verfahrens, es wird sehr schwierig sein, das Wissen für ein solches System zu formulieren. Die Abhängigkeiten sind komplex und basieren zum Teil aus Schätzungen bzw. Erfahrungswerten, die schwer zu fassen sind. Auch die Einbringung der Laufzeitparameter würde die Komplexität der Regeln deutlich erhöhen. Das Wissen läßt sich vermutlich auch nicht durch externe Experten ergänzen, um eine breitere Wissensbasis zu bekommen (und evtl. auch Fehler korrigieren zu können), da insbesondere Benchmark-Experten aus der Industrie ihr Wissen nicht vorbehaltlos weitergeben werden. An der Stelle wäre es auch wünschenswert, wenn das System eine Hilfestellung wäre, indem es Zusammenhänge selber erkennen kann und eine hohe Fehlertoleranz hat, was bei den Expertensystemen nicht der Fall ist. Vorteilhafte Eigenschaften der Expertensysteme sind allerdings noch, dass die Lösung nachvollzogen werden kann und damit auch wieder durch den Experten überprüfbar wird. Zudem können neue Bedingungen, wie z.B. eine veränderte Rechnerarchitektur, in das System eingepflegt werden, wenn auch, wie oben beschrieben, davon auszugehen ist, dass die Regeln dafür nicht immer einfach zu bestimmen sind.

Bei den Neuronalen Netzen kann das vorhandene Wissen leider nicht eingebracht werden. Es besteht jedoch die Hoffnung, dass durch eine große Basis von Benchmarkingdaten ein solches System gut trainiert werden könnte, so dass es die Zusammenhänge selbst lernt. In dieser Datenbasis finden sich dann auch die Laufzeitparameter, wobei auch hier die Komplexität des Trainings durch diese Parameter stark zunimmt. Zudem besteht aber die Möglichkeit, dass das System über das Expertenwissen hinaus Zusammenhänge er-

kennt. Wie oben deutlich geworden, ist auch die Fehlertoleranz ein großer Pluspunkt.
Ein Nachteil ist, dass ein Neuronales Netz nicht gut auf veränderte Gegebenheiten reagieren kann. Es muß zwar nicht neu entwickelt werden, aber ein erneutes Training ist in der Regel erforderlich.
Insbesondere kann ein Künstliches Neuronales Netz keine Aussagen treffen über völlig unbekannte Zusammenhänge, beim Benchmarking also die Aufgabe z.B. das Verhalten einer komplett neuen Rechnerarchitektur vorauszusagen. An dieser Stelle ist allerdings auch der Wissensaufbau für ein Expertensytem noch einmal extrem erschwert.

Die Etablierung der Verbindungen zwischen den erarbeiteten Klassen *Programme* und *Rechner* sind ebenfalls für die Entscheidung wichtig. Es geht dabei z.B. darum, wie stark ein Programm vom Netzwerk des Rechners abhängig ist.
Bei den Ontolgien sind diese Verbindungen fest und können nicht graduell eingetragen werden. Daher würde man die beiden Klassen getrennt voneinander aufbauen und müsste die Abhängigkeiten in zusätzlichen Regeln des Expertensystems spezifizieren. Damit müssten die Regeln evtl. sehr speziell auf die einzelnen Programme oder zumindest Programmkerne abgestimmt werden und das System würde sehr komplex und zudem an Allgemeingültigkeit verlieren. Es gibt auch Arbeiten über probabilistische Ontologien, bei denen also direkt in der Ontologie die Verbindungen mit einer Wahrscheinlichkeiten versehen sind [W23]. In dieser Arbeit würden aber statt Wahrscheinlichkeiten gewichtete Verbindungen benötigt werden.
Bei den Künstlichen Neuronalen Netzen werden die Verbindungen zwischen den Klassen gar nicht explizit geschaffen. Ein solches System sollte sich so verhalten, dass es aus den Trainingsdaten lernt, dass z.B. Programme mit starker Netzabhängigkeit auf einem Rechner mit schnellem Netzwerk eine gute Performance haben. Dabei sei angemerkt, dass dieser Kausalzusammenhang vom Netz nicht explizit erkannt wird, sondern nur die Gewichte im Lernprozess entsprechend verändert werden.

5.3 Kurzzusammenfassung und Auswahl eines Verfahrens

In diesem Kapitel wurden die Methoden der Wissensbasierten Systeme und der Künstlichen Neuronalen Netze miteinander verglichen.
Es hat sich gezeigt, dass beide Methoden für das vorliegende Problem unterschiedliche Vorteile und auch Nachteile mitbringen. Der große Vorteil der Expertensysteme liegt in der Möglichkeit, das vorhandene Wissen direkt einzubringen und zu nutzen. Es ist aber wie beschrieben bei Neuronalen Netzen so, dass sie ihr Wissen verteilt über die Lernbeispiele erhalten. Dadurch kann durch gezielte Auswahl der Trainingsdaten auch einem Neuronalen Netz Wissen implizit mitgeteilt werden. Zudem haben die Neuronalen Netze den entscheidenen Vorteil, dass die Verbindungen zwischen den Klassen nicht starr in das System eingehen. Auf diesen Überlegungen basierend, wird die Entscheidung getroffen, eine entsprechende Benchmarking-Prognose mit Hilfe der Künstlichen Neuronalen Netze zu modellieren.

6 Testreihen

Die vorangegangenen Kapitel haben zu der Entscheidung geführt, für das Benchmarking-Prognose-System Künstliche Neuronale Netze einzusetzen.
Um das Konzept entwickeln zu können, werden zuerst die Möglichkeiten der Verbindung von Neuronalen Netzen mit der Benchmarking-Prognose anhand von verschiedenen Tests untersucht.
Dazu werden zunächst KNN-Simulatoren betrachtet, um einen für die geplanten Tests geeigneten zu finden. Danach werden einige Grundlagen für die Tests erläutert und Informationen aus der Theorie der KNN ergänzt. Daraufhin folgen die Testreihen, die die Grundlage der Konzeption bilden sollen.

6.1 Simulator für Künstliche Neuronale Netze

Um mit Neuronalen Netzen arbeiten zu können, wird ein Werkzeug benötigt, dass diese Netze erzeugen und mit verschiedene Trainingsalgorithmen trainieren kann. Daher ist als erstes zu untersuchen, ob existierende Software für Neuronale Netze genutzt werden kann. Das gefundene Werkzeug kann dann entweder auch innerhalb des konzipierten Softwaresystems genutzt werden oder es dient nur zur Vereinfachung des Testprozesses und in der Konzeption wird eine Eigenentwicklung vorgeschlagen.

Nach einer entsprechenden Recherche wurde eine Auswahl an Produkten näher untersucht:

- MemBrain
- JavaNNS
- Matlab
- FANN-Bibliothek

MemBrain [W24] wurde betrachtet, da es in Internetforen empfohlen wird und einen guter Einstieg in die Entwicklung von Neuronalen Netzen zu bieten schien. Es hat eine graphische Benutzeroberfläche und beinhaltet verschiedene Formen des Backpropagation-Lernalgorithmus sowie einen Algorithmus für unüberwachtes Lernen. Es ist jedoch nur für Windows erhältlich und erscheint somit für die eher Unix-geprägte HPC-Landschaft nicht geeignet.
JavaNNS [W25] ist der Nachfolger des sehr verbreiteten KNN-Simulators SNNS [W26] (Stuttgart Neural Netzwork Simulator). Auch hier findet sich eine graphische Benutzeroberfläche, über die das Netz aufgebaut werden und das Training gesteuert werden kann.

Es ist hingegen nicht einfach möglich, das Tool ohne die Oberfläche im Batchmodus zu betreiben.
Matlab [W27] ist ein Softwarepaket zur Lösung von numerischen Problemen und liefert mit der Neural Network Toolbox eine Vielzahl an Möglichkeiten für die Erzeugung von verschiedenen Typen an Neuronalen Netzen und dem Training dieser Netze. Es handelt sich um ein sehr ausgereiftes Produkt, ist jedoch im Gegensatz zu den anderen Werkzeugen kommerziell und nicht frei erhältlich. Die OpenSource-Variante Octave [W28] verfügt noch nicht über ein ausgereiftes Modul für die Arbeit mit Neuronalen Netzen.
Die FANN-Bibliothek (Fast Artificial Neural Network Library) [W29] ist eine in der Programmiersprache C geschriebene Bibliothek, über die mehrschichtige Neuronale Netze simuliert werden können. Sie hat Schnittstellen zu verschiedenen anderen Programmiersprachen, u.a. findet sich hier auch die Sprache Perl, in der auch die Jülicher Benchmarking Umgebung JuBE verfasst ist. Es existiert eine graphische Benutzeroberfläche, die jedoch nicht so ausgereift ist wie bei den vorhergehenden Produkten. Dem steht aber gegenüber, dass Neuronale Netze und ihre Trainingsdaten sehr leicht über Konfigurationsdateien angesprochen werden können. Daher scheint die FANN-Bibliothek einen guten Einstieg in die Arbeit mit Neuronalen Netzen zu bieten.
Die im folgenden beschriebenen Testreihen wurden daher mit der FANN-Bibliothek durchgeführt, die auch im zu entwickelnden Softwaresystem als Bestandteil dienen könnte.

6.2 Grundlagen der Untersuchung

Für das Konzept werden zwei Schwerpunkte - die Datenbasis und die Netzwerkarchitektur - eine Rolle spielen, die durch die Tests näher beleuchtet werden sollen. Die Datenbasis bildet die Eingabe des KNN und enthält das Wissen, das das KNN daraus extrahiert. Dieser Ablauf ist in Abb. 6.1 noch einmal schematisch dargestellt.

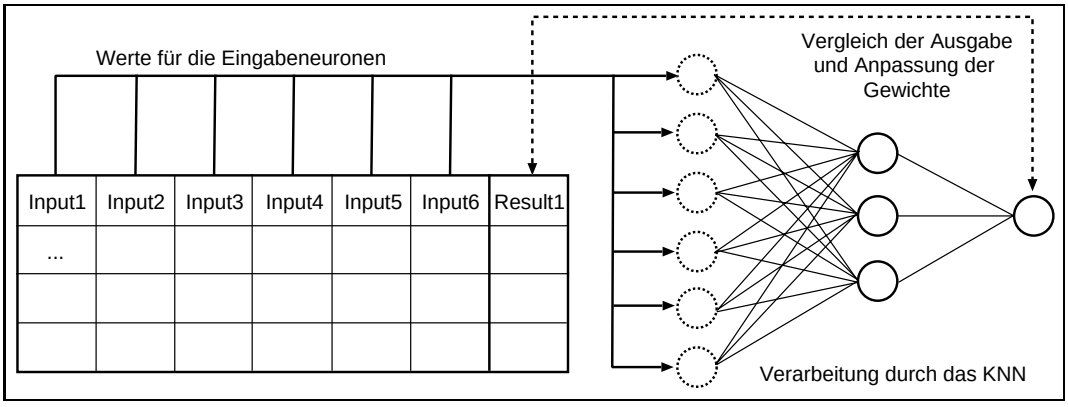


Abbildung 6.1: Das Training eines KNN basiert auf der zur Verfügung stehenden Datenbasis, die über die Eingabeneuronen in das Netz eingeht. Die Anpassung der Gewichte geschieht über den Abgleich mit der gewünschten Ausgabe.

6.2.1 Datenbasis

Beim ersten Schwerpunkt handelt es sich um die Datenbasis, also die vorliegenden Benchmark-Messungen. Im Kapitel 2 wurden die Benchmark-Suiten beschrieben, aus denen die Trainingsdaten für die Neuronalen Netze gewonnen werden. Es ist wichtig zu wisssen, welche Daten genau vorliegen und wie sie dem Netz in der Eingangschicht präsentiert werden können. Eventuell wird sich herausstellen, dass für ein aussagekräftiges Bild noch Daten ergänzt werden müssen, um z.B. die Abhängigkeit eines Programms von bestimmten Komponenten eines Rechners besser einordnen zu können. Hier können Low-Level-Benchmarks hilfreich sein.
Für die Eingangschicht des Netzes ist zu entscheiden, wie die Parameter dem Netz übergeben werden, also ob die vollen Parameter eines Benchmarklaufs in der Eingabeschicht abgebildet werden und das Netz dadurch alle Zusammenhänge lernen kann oder die Programme vorklassifiziert werden und dem Netz Eingaben wie „stark kommunikationsabhängig“ präsentiert werden. Der zweite Weg wird vermutlich der gangbare sein, da der volle Parametersatz inklusive aller Laufzeitparameter ein sehr komplexes Netzwerk hervorriefe, für das entsprechend viele Trainingsdatensätze benötigt würden. Die Attribute für die Eingangschicht sind somit herauszufinden.
Zur Datenbasis gehört auch das Ergebnis des Neuronalen Netzes, also ob z.B. eine Bewertung oder Klassifikation vorliegt. Bei einer Bewertung hätte man Aussagen wie „Programm P1 läuft gut auf Rechner R1“ als Ergebnis des Netzes. Bei der Klassifikation können genauere Aussagen getroffen werden, wie die Simulation einer Laufzeit oder des Durchsatzes, auch detailliertere Ausgaben durch mehrere Ausgabeneuronen sind vorstellbar.
Nach der Präsentation der Daten in Eingabe- und Ausgabeschicht ist auch wichtig herauszufinden, wieviele Trainingsdaten überhaupt für eine gegebene Netzwerk- bzw. Eingangsvektorgröße benötigt werden und ob diese Datensätze orthogonal zueinander sein müssen.

6.2.2 Netzwerkarchitektur

Durch die Tests soll eine den Daten angemessene Netzwerkarchitektur gefunden werden. Darunter fällt die Festlegung auf ein Lernverfahren, die Anzahl der Neuronen und Schichten sowie die Definition der Aktivierungsfunktionen und der Trainingsdauer.
Durch die Literatur-Recherche hat sich herausgestellt, dass bei der Funktionsapproximation ein mehrschichtiges Perzeptron (MLP, s.Kap.4) am häufigsten verwendet wird und gute Ergebnisse liefert [L10]. Daher werden die ersten Tests mit diesem Netzwerktyp gefahren, um durch Modifikation der Neuronen- und Schichtenanzahl die optimale Netzstruktur herauszufinden.

Weitere Fragen hinsichtlich der Netzwerkarchitektur sind auch, wie lange ein Training laufen muß, damit die Datensätze korrekt gelernt werden. Dabei ist auf der anderen Seite zu beachten, dass ein Netz nicht übertrainiert wird, sondern seine Generalisierungseigenschaft behält (siehe Abschnitt 6.3.1).

6.3 Theorie

Zum Verständnis des Trainings der Neuronalen Netze und potentieller Schwierigkeiten sind noch weitere Informationen aus der Theorie der Neuronalen Netze hilfreich, die im allgemeineren Kapitel 4 zur KI noch nicht eingeführt wurden.

6.3.1 Overfitting

Unter Overfitting versteht man, dass ein KNN bei zu langem Training seine Generalisierungseigenschaft verlieren kann und nur die Trainingsdaten auswendig lernt.
Um dieses Verhalten während des Trainings zu erkennen, sollte neben dem Trainingsdatensatz ein Referenzdatensatz, dessen Daten nicht trainiert werden, abgeprüft werden. Der Fehler dieses Datensatzes wird in der Regel höher als der des Trainingssatzes sein, sollte aber im Verlauf des Trainings ebenfalls sinken. Hat man einen Punkt erreicht, an dem der Fehler des Trainingssatzes gleich bleibt oder weiter sinkt, der Fehler des Referenzsatzes aber wieder ansteigt, hat man ein Anzeichen, dass das Netz bereits auswendig lernt und nicht mehr generalisiert.

6.3.2 Trainingsende

Dies führt zur Frage, wann ein Training beendet werden sollte. In der Regel wird man beim überwachten Training den Fehler zu den Originaldaten verwenden, z.B. in Form der Fehlerquadratsumme, der unter einen bestimmten Wert fallen soll. Parallel dazu müssen die bzgl. Overfitting aufgeführen Überlegungen berücksichtigt werden.
Um die Endkonfiguration zu bestätigen, ist es häufig sinnvoll, ein wiederholtes Training durchzuführen, wobei unterschiedliche Gewichtsinitialisierungen beim Start vorliegen sollten.
Zudem kann man eine *Kreuzvalidierung* durchführen, bei der die vorhandenen Daten für den Eingabevektor wiederholt in disjunkte Mengen von Trainings- und Referenzdaten unterteilt werden. Auf diese Weise werden alle Daten auch einmal als Referenz benutzt.

6.3.3 Backpropagation

Der Backpropagation-Algorithmus wurde bereits in Kapitel 4 beschrieben. Da es sich beim Backpropagation-Algorithmus um ein Gradientenverfahren handelt, müssen bei der Verwendung die typischen Probleme eines solchen Verfahrens berücksichtigt werden. Es handelt sich dabei um folgenden mögliche Verhaltensweisen [W30]:

- Konvergenz gegen lokale Minima
- Stagnation auf flachen Plateaus
- Überspringen bzw. Verlassen guter Minima
- Oszillation in steilen Schluchten

Strategien, die zur Vermeidung eingesetzt werden können, sind:

- Mehrfaches Training mit unterschiedlichen Gewichtsinitialisierungen, um verschiedene Startpunkte zu haben.
- Nutzung unterschiedlicher Lernraten: Die Lernrate bestimmt die Geschwindigkeit des Trainings. Ist sie zu groß, ist die Gefahr, dass gute Minima übersehen werden, vorhanden, ist sie jedoch zu klein, kann das Training unter Umständen sehr viel Zeit in Anspruch nehmen.
- Nutzung eines Momentums: Dabei handelt es sich bereits um eine Erweiterung des klassischen Backpropagation-Algorithmus, jeder Gewichtsänderung wird ein Anteil der im Lernschritt zuvor erfolgten Änderung hinzuaddiert. Dadurch wird der Stagnation auf flachen Plateaus und auch der Oszillation in Schluchten entgegengewirkt. Gute Minima werden jedoch häufiger verlassen.

6.3.4 Resilient Propagation

Ein Parameter des Backpropagation-Algorithmus ist die Lernrate. Diese Rate wird fest vorgegeben für die ganze Dauer des Algorithmus unabhängig von der Position auf der Fehlerkurve. Darin sind einige der oben beschriebenen Probleme begründet.
Bei Resilient Propagation (Rprop) [L5, W31] handelt sich um einen adaptiven Algorithmus, d.h. die Lernrate passt sich dynamisch an: Es werden größere Schrittweite bei gleichbleibendem Gradienten und kleinere bei einer Richtungsänderung gewählt.
Der Rprop-Algorithmus konvergiert für gewöhnlich schneller als Backpropagation, es kann jedoch auch aufgrund von Unstetigkeiten am Minimum der Approximation zum Überspringen des Minimums kommen.

6.4 Tests

Um die Möglichkeiten der KNN in Verbindung mit Benchmarking auszutesten und ihr Verhalten besser kennenzulernen, wurde eine Reihe von Tests durchgeführt, auf deren Grundlage das Konzept entwickelt werden soll.

Die erste Frage, die sich bei der Art der Tests stellte, war die Auswahl der Datenbasis. Grundsätzlich sind zwei verschiedene Ansätze möglich:

- Künstliche Datenbasis
  Mit einer künstlichen Datenbasis ließen sich alle Fragen bzgl. der Datenbasis durchtesten, man könnte also z.B. für jede Netzarchitektur einen beliebig großen Datensatz erzeugen, ergänzt durch einen entsprechenden Referenzdatensatz.
  Es bliebe dabei aber die Frage offen, ob sich die KNN mit realen Benchmark-Daten auf die gleiche Weise verhielten oder ob vielleicht Schwierigkeiten, die evtl. mit realen Daten aufträten, bei den künstlichen Daten nicht bemerkt würden. Zudem besteht das Problem, dass beim Aufbau des künstlichen Datensatzes, die Zusammenhänge in den Daten, also das Wissen, geschaffen werden müssen. Das ist zum einen gut, da das Netz, das dieses Wissen wiederum extrahiert, so auch überprüft

werden kann. Zum anderen ist aber zu bedenken, dass die Zusammenhänge eines realen Benchmarks evtl. viel komplexer sind als ein stringent geplanter, künstlicher Benchmark.

- Realer Applikations-Benchmark
  Mit einem realen Benchmark hat man zwar vermutlich keine ideale, da für verschiedene Tests passend erzeugte, Datenbasis, aber das ist ja auch eine Schwierigkeit, mit der das System in der Realität arbeiten muss.
  Probleme können allerdings für die Tests bzw. das Training eines Netzes aufkommen, wenn z.B. die Datenlage nicht ausreichend ist.

Es wurde die Entscheidung getroffen, mit den Daten von existierenden Applikations-Benchmarks zu arbeiten, um das System nicht an der Realität vorbei zu konzipieren.

6.4.1 Tests mit PEPC

Die ersten Tests wurden mit den Daten des Benchmarks *PEPC* durchgeführt. PEPC (Pretty Efficient Parallel Coulomb-solver) ist sowohl Teil der DEISA als auch der PRACE Benchmark-Suite [W32, W33, L19].
Es handelt sich um einen parallelen Baum-Code zur effektiven Berechnung von langreichweitigen Wechselwirkungen von Coulomb Kräften in N-Körper Systemen und findet Anwendung in der Molekulardynamik, Laser-Plasma-Interaktion sowie der Astrophysik. PEPC ist auf IBM Power 4 und Power 6 sowie CRAY-Systemen und auf die Blue Gene-Architektur portiert. Von diesen Systemen existieren eine Reihe von Benchmark-Messungen, die für die folgenden Tests genutzt werden konnten.

Vorgehen

Die ersten Trainingsdaten wurden aus den Benchmarkläufen auf der IBM Power 4 und Power 6 Jump des JSC [W34] und der IBM Power 6 VIP [W35] des Rechenzentrums der Max-Planck Gesellschaft in Garching gewonnen. Bei diesen Benchmark-Messungen wurden Wechselwirkungen verschiedener Größe betrachtet, was sich durch die Anzahl der Partikel ausdrückt.
Aus den Läufen wurde der Eingangsvektor für ein kleines Neuronales Netz zusammengesetzt bestehend aus folgenden Daten:

- Anzahl Prozessoren: `ncpus`
  Die Anzahl der Prozessoren liegt bei den Daten zwischen 32 und 1024.

- Anzahl Knoten: `nodes`
  Die Anzahl der Knoten liegt zwischen 1 und 32.

- Anzahl Partikel: `npart`
  Die Anzahl der Partikel und damit die Problemgröße des Benchmarks betrug 2-, 3- bzw. 5 Millionen Partikel.

- Die Plattform: `platform`
  Die Plattform wird repräsentiert durch die Geschwindigkeit des Prozessors, codiert in Form des Prozessortaktes.

- Taskspernode: `tpn`
  Bei den Taskspernode handelt es sich um die Anzahl der Tasks, also der Prozesse, pro Knoten, in den Daten lagen die `tpn` zwischen 4 und 64.

- Threadspertask: `tpt`
  Die Threadspertask sind die Anzahl der Subprozesse pro Task, in den Daten sind die `tpt` jeweils 1.

Der Outputvektor besteht aus der:

- Belegzeit: `walltime`
  Die Belegzeit ist die Zeit, die der Benchmarklauf summiert über alle Prozessoren benötigt hat, um seine Rechnungen durchzuführen.

Für die Tests ist durch die `platform` ein Vertreter der in der Analyse in Kapitel 3 beschriebenen Klasse der *Rechner* vorhanden. Die Klasse *Programm* wird durch die Anzahl Partikel `npart` beschrieben und zusätzlich durch die Laufzeitparameter `ncpus`, `nodes`, `tpn` und `tpt`, die die Ausführungszeit des Benchmarks wesentlich bestimmen. Diese Ausführungszeit, also die `walltime`, ist ein Element der Klasse *Messung*.

Ein exemplarischer Auszug dieser Werte aus den Läufen sieht dann wie folgt aus:

```
...
ncpus=128 nodes=2 npart=5000000 platform=4.7 tpn=64 tpt=1
walltime=63.040
ncpus=64  nodes=2 npart=2000000 platform=4.7 tpn=32 tpt=1
walltime=67.710
ncpus=192 nodes=6 npart=5000000 platform=4.7 tpn=32 tpt=1
walltime=65.200
...
```

Die Werte werden für den Netzsimulator auf den Wertebereich von 0 bis 1 abgebildet, um Probleme mit zu kleinen oder großen Gewichten zu vermeiden. Dabei stellt 1 aber nicht das Maximum von 1024 Prozessoren dar. Dadurch werden auch Prognosen für größere Anzahlen an Prozessoren ermöglicht, die ansonsten aus dem Wertebereich fallen würden:

```
...
0.061 0.016 1.000 1.000 1.000 1.000
0.076
0.029 0.016 0.400 1.000 0.467 1.000
0.083
0.092 0.079 1.000 1.000 0.467 1.000
0.079
...
```

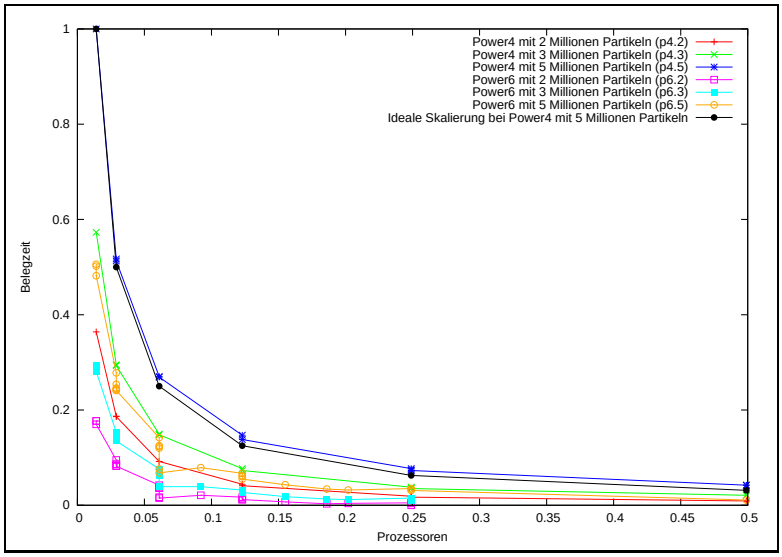


Abbildung 6.2: Die Kurvenschar der Skalierungskurven von PEPC für die Systeme Power4 und Power6, jeweils mit Läufen von 2, 3 bzw. 5 Millionen Partikeln. Exemplarisch ist die Kurve der idealen Skalierung für die Power4 mit 5 Millionen Partikeln eingezeichnet.

Die Original-Skalierungskurven von PEPC sind in Abb. 6.2 zu sehen. Man sieht das Verhalten des Simulationsprogramms, die Läufe auf den schneller getakteten Power6-Systemen sind wie zu erwarten schneller als die korrespondierenden Läufe auf Power4, die Läufe mit kleineren Partikelzahlen wiederum sind schneller als die größeren Simulationen. Insgesamt ist im Vergleich zur idealen Skalierung, die als Beispiel für Power4 mit 5 Millionen Partikeln eingezeichnet ist, ein gutes Verhalten zu erkennen.
Für einige Prozessorzahlen scheint es Unstetigkeitsstellen in den Kurven zu geben, z.B. bei der Kurve der Power6 für 5 Millionen Partikel bei 0.061. Bei den kleineren Sprüngen kann es sich um Abweichungen mehrerer Messungen handeln, größere Sprünge wie in diesem Beispiel werden verursacht durch eine andere Aufteilung der Prozessoren auf Knoten. Wenn mehr Knoten benutzt werden, kann es zu einer signifikant anderen Ausführungszeit kommen.

Technische Durchführung mit FANN

Das Training und der Test des Netzes wird über die FANN-Bibliothek durchgeführt, die durch entsprechende C-Programme angesprochen wird.

Training des Neuronalen Netzes

Ein Auszug aus einem einfachen Trainingsprogramm sieht wie folgt aus, es ist in der Programmiersprache C codiert:

```
1 ...
2  struct fann *ann = fann_create_standard(num_layers, num_input,
3                                  num_neurons_hidden, num_output);
4
5  fann_set_activation_function_hidden(ann, FANN_ELLIOT);
6  fann_set_activation_function_output(ann, FANN_ELLIOT);
7
8  fann_train_on_file(ann, "pepc.data",max_epochs,
9                           epochs_between_reports,desired_error);
10
11 fann_save(ann, "pepc.net");
12
13 fann_destroy(ann);
```

Zunächst wird eine Struktur für das Netz erzeugt (2), indem die Anzahl der Schichten angegeben wird sowie die Anzahl sowohl der Neuronen in der Inputschicht als auch der verdeckten Schichten und die Anzahl der Outputneuronen. Zudem werden Aktivierungsfunktionen für die verdeckte und die Ausgangsschicht festgelegt (5,6), in dem vorliegenden Beispiel handelt es sich um die Elliotfunktion (s. Abbildung 4.7(c) auf Seite 37).

Danach wird das Training durchgeführt (8). Es endet, sobald die Fehlerquadratsumme einen vorgegebenen Wert unterschreitet. Sollte dieser Wert nicht unterschritten werden, endet es nach einer maximalen Anzahl von Trainingsepochen. In einer Epoche wird das Netz genau einmal mit allen Trainingssätzen konfrontriert und entsprechende Gewichtsanpassungen vorgenommen. Das resultierende Netz wird mit seiner Konfiguration und insbesondere den resultierenden Gewichten gespeichert (11).

Test des Neuronalen Netzes

Der Test des resultierenden Netzes kann ebenfalls über FANN erfolgen:

```
1  ...
2  struct fann *ann = fann_create_from_file("pepc.net");
3  fp=fopen("pepc.test","r");
4  op=fopen("pepc.knn","w");
5
6  while (!feof(fp))
7    fscanf(fp,"%f %f %f %f %f %f",
8                &input[0],&input[1],&input[2],&input[3],&input[4],&input[5]);
9    calc_out = fann_run(ann, input);
10   fprintf(op,"%f %f %f %f %f %f %f\n",
11     input[0], input[1],input[2], input[3],input[4], input[5],calc_out[0]);
12
13 fann_destroy(ann);
```

Dazu wird das im Training erzeugte Netz aus dem Konfigurationsfile gelesen und das Netz entsprechend aufgebaut (2). Aus einer Datei werden Inputvektoren gelesen, diese Vektoren werden dem Netz vorgelegt, das daraus einen Ausgabewert erzeugt (9).

Auswertung der Tests

Die Auswertung der resultierenden Netze ist im Detail bei der Testbesprechung zu finden. Die Tests wurden im Regelfall durchgeführt, indem bekannte Daten dem Netz im Training vorenthalten wurden, so dass sie zum Test des Netzes herangezogen werden konnten. Die jeweils resultierenden Ergebnisse des KNN können so mit den Originaldaten verglichen werden, es ist möglich die Kurvenverläufe abzugleichen und Fehler zu den Originaldaten zu berechnen.

6.4.2 Netztests

Die ersten Tests sollen Aufschluss über das Verhalten der KNN geben und Aussagen ermöglichen, welche Netzarchitektur gut geeignet ist, d.h. die vorgegebenen Daten gut lernt.
Zur Notation der gewählten Architektur werden die Anzahl der Neuronen in den verschieden Schichten mit : getrennt voneinander angegeben, 6:3:1 steht also für ein feed-forward Netz mit einer verdeckten Schicht. In der Eingabeschicht befinden sich 6, in der verdeckten Schicht drei Neurone und es gibt ein Ausgabeneuron.

Die 6:x:1 Netz-Architektur

Das Ziel dieses ersten Tests ist zu verifizieren, ob das Lernen eines Benchmark-Datensatzes anhand eines mehrschichtigen feed-forward-Netzwerkes möglich ist und das Verhalten eines solchen Netzes besser kennenzulernen.

Für den Test standen Datensätze aus 113 Läufen des PEPC-Benchmarks zur Verfügung. Bzgl. der Menge von benötigten Trainingsdaten gibt es keine einheitlichen Aussagen. Dennoch muss gesagt werden, dass ein größerer Datensatz wünschenswert wäre, da die Daten sich teilweise auch recht ähnlich sind.
Das Netze wurde mit einer Zwischenschicht aufgebaut. Bei dieser Zwischenschicht wird die Anzahl der Neuronen variiert und die Güte anhand des resultierenden relativem Fehlers beurteilt sowie die resultierenden Kurvenverläufe graphisch gezeigt.
Als Lernalgorithmus wurde zunächst Resilient Propagation gewählt, da mit diesem Algorithmus die dynamische Anpassung der Lernrate erfolgt. Diese Entscheidung soll aber auch noch in Tests mit Backpropagation geprüft werden.
Als Aktivierungsfunktion wurde sowohl die Elliot- wie auch die Sigmoidfunktion genutzt. Das Netz wurde traininert, bis es eine Fehlerquadratsumme (ESS) von 0.00001 erreicht hat oder die als maximal eingestellten 500000 Epochen.
Anschließend wurde ein Test mit den Trainingsdaten durchgeführt, um zu sehen, ob die Skalierungskurve des Benchmarks gelernt wurde. Wichtig ist allerdings auch, wie sich ein nicht zum Training verwendeter Testdatensatz verhält, um einer Übertrainierung des Netzes vorzubeugen. Daher wurde eine kleine Menge der Trainingsdaten, 10 Werte, herausgenommen, um als Referenz zu dienen.
In der folgenden Abb. 6.3 sind die sich ergebenen Fehlerquadratsummen pro Netztopologie aufgeführt. Dabei ist für sowohl für die Elliot- wie auch für die Sigmoidfunktion

der ESS der Trainingsdaten am Ende des Trainings, also in allen Fällen nach 500000 Epochen, und auch der ESS des Referenzdatensatzes aufgeführt.

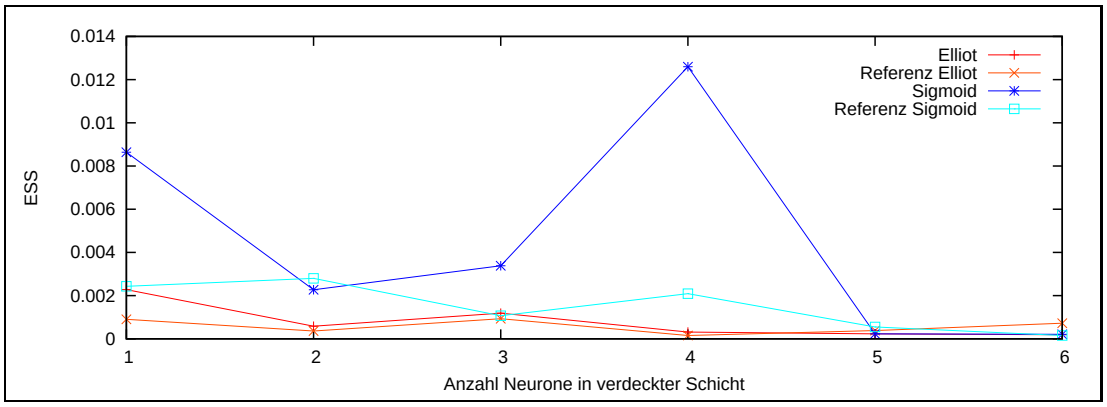


Abbildung 6.3: Die Entwicklung des ESS bei einem 6:x:1-Netz in Abhängigkeit der Anzahl der Neuronen in der verdeckten Schicht.

Es ist zu sehen, dass sich mit der Elliotfunktion in den meisten Fällen eine bessere Annäherung erzielen läßt als mit der Sigmoidfunktion. Der Fehler des Referenzdatensatzes bleibt in beiden Fällen niedrig, häufig sogar unter den trainierten Daten. Das wäre erstaunlich, liegt aber an der kleinen Menge Referenzdaten, zudem sind sich, wie bereits festgestellt, die Daten in diesem relativ kleinen Testbeispiel recht ähnlich, so dass auch diese Datensätze gut angenähert werden.

Am Beispiel des Datensatzes der Power4-Daten mit 5 Millionen Partikeln kann in Abb. 6.4 auf der nächsten Seite das Verhalten der verschiedenen Topologien gesehen werden. Dabei sei angemerkt, dass bei dieser und auch allen folgenden Ergebnisgraphiken der KNN zu beachten ist, dass das Netz im Grunde nur eine Punktewolke durch eine Funktion annähert. Diese Punkte werden für die Abbildungen durch die erwartete Kurvenschar der Skalierungskurven dargestellt.
In Abb. 6.4 ist gut zu erkennen, dass die beiden kleinsten Netze mit einem bzw. zwei Elementen in der verdeckten Schicht die schlechtesten Lernerfolge zeigen: Das 6:1:1 Netz weicht im ersten Teil stark ab, das 6:2:1 im zweiten. Mit Zunahme der Elemente in der verdeckten Schicht wird auch die ursprüngliche PEPC-Kurve immer besser angenähert. Dabei ist zu beachten, dass bei einem 6:6:1 Netz bereits sehr viele Verbindungen im Verhältnis zur Anzahl der Trainingsdaten vorhanden sind, so dass an dieser Stelle ein "auswendiglernen" wahrscheinlich ist. Ein größerer Referenzdatensatz würde die Menge an Trainingsdaten aber weiter verringern.
In der Abbildung ist weiterhin zu erkennen, dass die Annäherung an die zu lernende Kurve von unten geschieht. Das liegt daran, dass nicht nur mit dem Datensatz der Power4 und 5 Millionen Partikeln trainiert wurde, sondern mit allen vorhandenen Daten von Power4 und Power6, also jeweils für 2, 3 und 5 Millionen Partikel. Alle anderen Skalie-

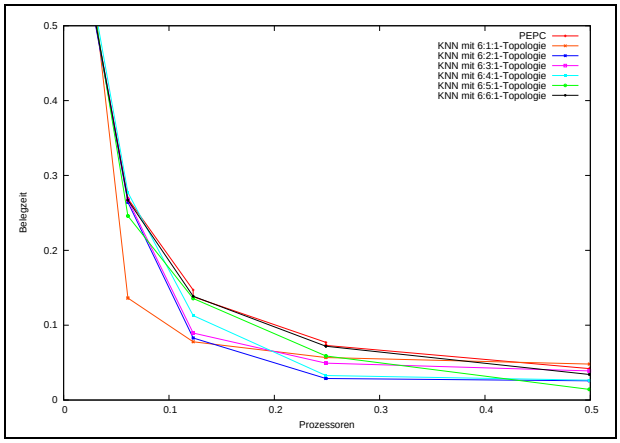
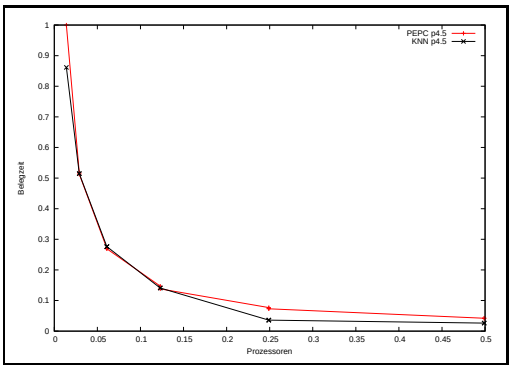


Abbildung 6.4: Lernerfolg verschiedener Topologien am Beipiel der Power4-Daten mit 5 Millionen Partikeln (Zoom): Mehr Neuronen in der verdeckten Schicht helfen bei der Annäherung an die Original-Kurve.

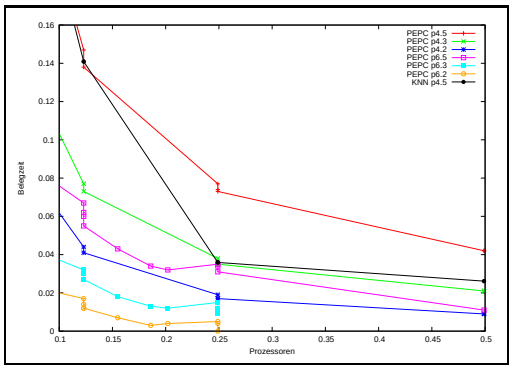
rungskurven liegen, wie in der Abb. der Original PEPC-Daten 6.2 auf Seite 56 zu sehen, unter der betrachteten Kurve, so dass das Training dieser Kurve z.B. von der Kurve der Power6 mit 5 Millionen Partikeln beeinflusst wird. Die Eingabevektoren dieser beiden Fälle unterscheiden sich ja im aktuellen Aufbau nur in einem Neuron, nämlich dem für die Prozessorgeschwindigkeit der Maschine.

Um das Vertrauen in die Ergebnisse zu bestärken, wurde das Training wiederholt mit verschiedenen Gewichtsinitialisierungen durchgeführt. In 6.5(a) auf der nächsten Seite ist noch einmal eine Kurve der Power4-Daten mit 5 Millionen Partikeln aufgeführt, die aus einem 6:3:1 Netz resultiert.
Sie zeigt noch eine etwas bessere Annäherung als die vorhergehende Simulation bei dem 6:3:1-Netz, aber wie zu erwarten prinzipiell das gleiche Verhalten, die Abweichung findet für größere Prozessorzahlen nach unten statt, die Werte werden offensichtlich von den Power6-Daten “abgelenkt“. Das ist in der vergrößerten Abb. 6.5(b) auf der nächsten Seite gut zu erkennen, wo alle Kurven, deren Werte beim Training benutzt wurden, aufgezeichnet sind.

In Abb. 6.6(a) auf der nächsten Seite sind die gesamten Skalierungskurven dargestellt. Die Kurvenverläufe sind im Vergleich zu Abb. 6.2 auf Seite 56 insbesondere für kleinere Prozessor-Werte recht gut angenähert, der Lauf der Kurven für größere Prozessor-Werte ist jedoch noch nicht zufriedenstellend. Alle Kurven nähern sich dort einander an. In Abb. 6.6(b) ist dieses Verhalten noch deutlicher zu erkennen, die Kurven wurden logarithmisch dargestellt und direkt den Original-PEPC Werten gegenübergestellt. Zur Übersichtlichkeit sind in der Graphik nur die Power4 Werte aufgeführt.
Dass die größeren Läufe nicht so gut angenähert werden, kann darin begründet liegen, dass mehr Trainingsdaten für kleinere als für größere Läufe vorlagen.

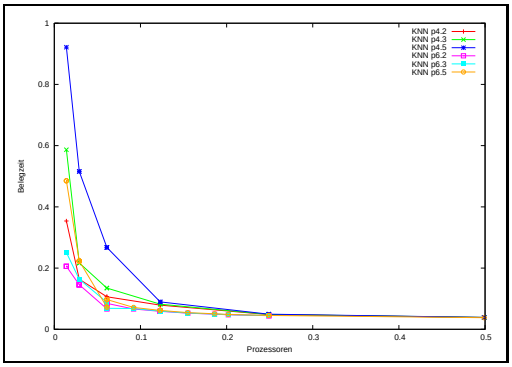


(a) Gelernte Skalierungskurve der Power4 mit 5 Millionen Partikeln

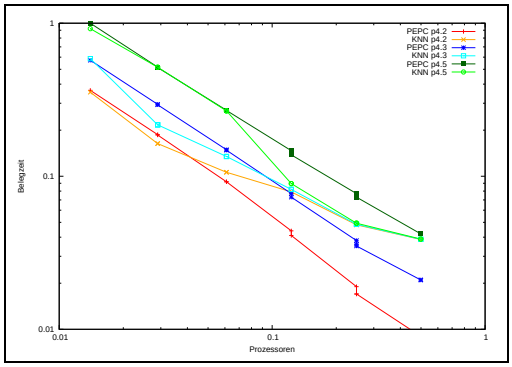


(b) Zoom inkl. Abbildung der kompletten Original-PEPC-Kurvenschar

Abbildung 6.5: Lernerfolg für Power4 mit 5 Millionen Partikeln: In (a) ist eine gute Annäherung zu sehen, die aber im zweiten Kurventeil nach unten abweicht, der Zoom in (b) macht deutlich, dass die Annäherung von den Power6-Trainingsdaten abgelenkt wird.



(a) Gelernte Skalierungs-Kurvenschar



(b) Darstellung der Power4-Trainings- und Testdaten in logarithmischer Form

Abbildung 6.6: Die Ergebniskurven des trainierten Netzes sind für kleinere Prozessorwerte zufriedenstellend, für größere laufen sie jedoch zusammen, was nicht der Realität entspricht. Dieses Verhalten wird in der logarithmischen Darstellung noch deutlicher.

Beispiel eines resultierenden Netzes

Nach beendetem Training ergibt sich eine Konfiguration des Netzes mit festgelegten Gewichten für jede Verbindung. Dieses Netz kann dann genutzt werden, um Anfragen zu stellen, d.h. unbekannte Eingabevektoren auf das Netz zu geben und das Ergebnis am Ausgabeneuron abzulesen.
Als Beispiel für eine solche Endkonfiguration ist in Abb. 6.7 das Netz für die Topologie 6:3:1 graphisch dargestellt. Zu beachten ist, dass sich sowohl in der Eingabeschicht wie auch in der verdeckten Schicht jeweils ein Bias-Neuron befindet (siehe Beschreibung im Abschnitt 4.2.5), das FANN automatisch hinzufügt. Zur Lesbarkeit sind die Verbindungen zwischen der Eingabe- und der verdeckten Schicht farbig gekennzeichnet und die zugehörigen Gewichte über dem jeweiligen Eingabeneuron notiert. Größere Gewichte wurden durch eine höhere Strichdicke markiert. Die Gewichte zum Ausgabeneuron wurden direkt an die zugehörige Verbindung geschrieben.
Es ist zu sehen, dass das erste verdeckte Neuron im Wesentlichen von der Anzahl CPUs,

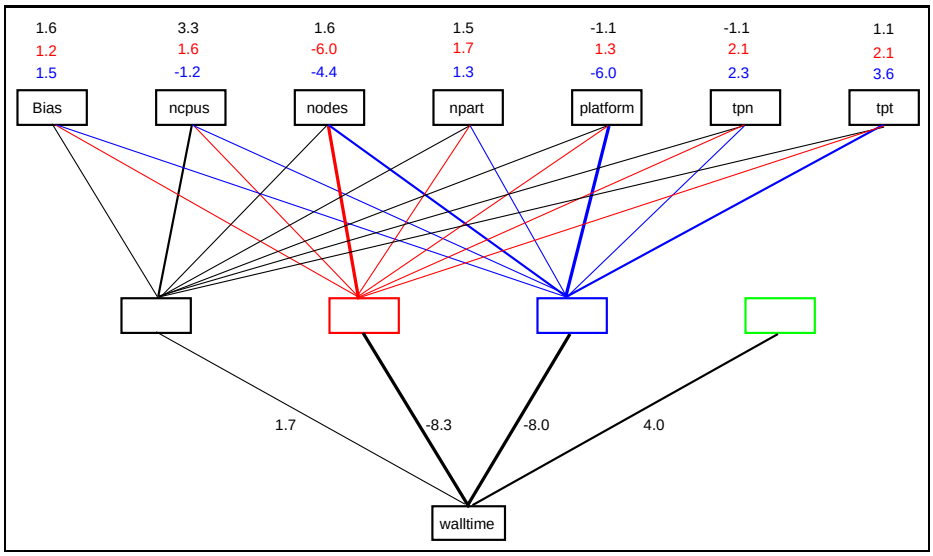


Abbildung 6.7: Resultierendes Netz nach einen Trainingslauf mit der FANN-Bibliothek.

das zweite verdeckte Neuron von der Anzahl Nodes geprägt wird. Das dritte verdeckte Neuron wird von der Anzahl Nodes und der Plattform am stärksten beeinflusst. Das vierte Neuron ist als Bias-Neuron nicht mit der Eingabeschicht verbunden. Die Ausgabe wiederum wird hauptsächlich vom zweiten und dritten verdeckten Neuron geprägt.

Vergleich mit JavaNNS

Um sich mit der Entwicklung nicht zu abhängig von einem Werkzeug zu machen und die Ergebnisse der FANN-Bibliothek zu verifizieren, wurde das Beispiel der 6:3:1-Architektur auch exemplarisch mit dem Programm JavaNNS getestet.

Dabei ist aber zu beachten, dass nicht der exakt gleiche Aufbau durchgeführt werden konnte. FANN fügt automatisch in jede Schicht ein Bias-Neuron ein während das bei JavaNNS nicht der Fall ist. Der Netzaufbau wird bei JavaNNS über die graphische Benutzeroberfläche gestaltet und es wurde keine Möglichkeit gefunden, Bias-Neurone zu definieren. In der Eingabeschicht könnte man einfach ein zusätzliches Neuron, das immer mit 1 im Eingabevektor belegt ist, einfügen, das ist in der verdeckten Schicht aber nicht möglich. Man kann dort zwar ein Neuron einfügen, das nur mit der Ausgabeschicht verbunden ist, dieses aber nicht mit 1 initialisieren. Daher wird es im Training nicht weiter beachtet, d.h. das Gewicht der Verbindung nicht verändert. Aus diesem Grund wurde hier ein einfaches 6:3:1-Netz ohne Bias aufgebaut, das in Abb. 6.8 zu sehen ist. Beim Aufbau wurden die Neuronen für jede Schicht erzeugt, mit der Aktivierungsfunktion Elliot versehen und feed-forward verbunden.

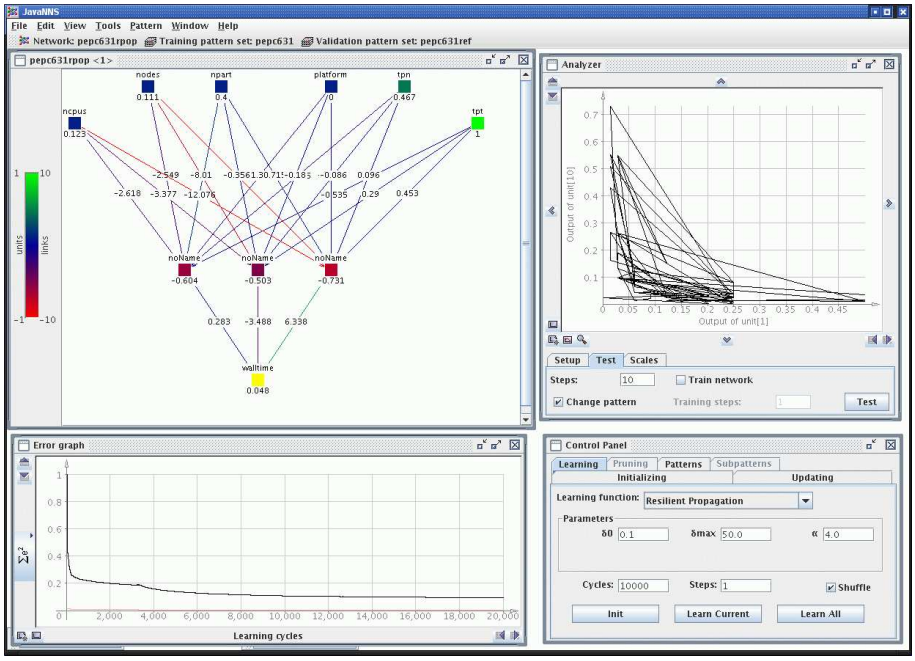


Abbildung 6.8: Oberfläche mit resultierendem Netz nach einem Trainingslauf des Simulators JavaNNS.

Innerhalb der graphischen Oberfläche von JavaNNS lassen sich verschiedene Sichten mit Informationen über das Netz öffnen:
Unten rechts sieht man das *Control Panel* über das sich das Training steuern läßt. Es ist zu erkennen, das wie bei FANN mit Resilient Propagation gelernt werden soll und verschiedene Parameter, wie z.B. die Anzahl Epochen *Cycles*, sind einstellbar.
Links daneben findet sich eine Lernkurve, der *Error graph*. Man sieht, dass die Fehlerquadratsumme angezeigt wird, die zunächst stark sinkt und dann beginnt sich einzupendeln.
Oben links ist das resultierende Netz zu sehen: Die Farben sind hier anders zu deuten, als bei der Graphik 6.7 zum FANN-Netz. Wie in der Legende auf der linken Seite ange-

zeigt, läßt sich an der Farbe der Neuronen die aktuelle Aktivierung des Neurons ablesen, die sich zwischen -1 und 1 befinden kann. An der Farbe der Verbindungen zwischen den Neuronen kann die Größe des Gewichtes der Verbindung abgelesen werden, die selber nicht direkt begrenzt ist, aber durch den Farbverlauf von -10 oder kleiner bis 10 oder größer angezeigt wird.
Schaut man sich die Verbindungen genauer an, ist zu erkennen, dass das erste verdeckte Neuron deutlich von den CPUs und den Nodes geprägt wird, auch die Partikelanzahl nimmt noch Einfluss. Das zweite und dritte Neuron werden ebenfalls größtenteils von den CPUs und Nodes beeinflusst.

Ein direkter Vergleich zum resultierenden Netz von FANN ist anhand der Gewichte nicht möglich. Das war auch nicht zu erwarten. Denn zum einen verhalten sich die Gewichte schon bei zwei gleichen Läufen eines Tools nicht exakt gleich, da sowohl die Initialisierung der Gewichte als auch die Reihenfolge der Präsentation der Trainingsdaten vom Zufall bestimmt wird. Zum anderen kommt die Bias-Problematik hinzu, insbesondere das Bias-Neuron der verdeckten Schicht hatte bei FANN durchaus Einfluss auf das Ausgabeneuron, so dass die Gesamtverteilung dort anders aussieht.
Das Training eines KNN kann im Grunde nicht direkt aus der Verteilung der Gewichte, sondern muss vom Ergebnis her beurteilt werden. Das heisst, es wird vom Verhalten des Fehlers während des Trainings und vom Fehler des Referenzdatensatzes beurteilt, zudem kann man sich auch Daten aus dem Parameterraum, also zum Beispiel die Skalierungskurven anschauen und mit den Originaldaten vergleichen wie es auch schon in Abb. 6.4 auf Seite 60 vorgenommen wurde.
JavaNNS bietet dazu den *Analyzer*, der oben rechts zu sehen ist. In dem angezeigten Fall wurde das erste Neuron, die CPUs, und das Ausgabeneuron, die Belegzeit, gegeneinander aufgetragen, also sollte man die PEPC-Skalierungskurven erhalten. JavaNNS verbindet die resultierenden Punkte automatisch mit Graden, so dass ein relativ wirrer Eindruck entsteht, da die Daten dem Netz in einer zufälligen Reihenfolge präsentiert werden. Zudem handelt es sich im Grunde nicht um eine Skalierungskurve, die das Netz gelernt hat, sondern um 6 Kurven (Power4 und Power6 jeweils für 2,3,5 Mill. Partikel). Eine ähnliche Punktewolke entsteht auch, wenn man sich die FANN-Ergebnisse ohne das Vorwissen der verschiedenen Kurven anschaut, s. Abb. 6.9 auf der nächsten Seite. In der Graphik sind die Ergebnisse des KNN rot dargestellt und darunter noch in grün die Original-PEPC-Punkte. Im Vergleich der beiden Tools sieht man, dass beide den Extrempunkt auf der Zeit-Achse bei 1 nicht ganz erreichen. Bei beiden Tools akkumulieren alle Kurven beim Maximum der Prozessoranzahl von 0.5 in einem Punkt. Die Darstellung erschwert aber einen genaueren Vergleich.
Da die Abb. 6.9 aber auch die PEPC-Originalwerte enthält, sind hier noch Vergleichspunkte aufzeigbar. Man sieht, das weniger rote Punkte vorhanden sind, das KNN hat also einige Werte gemittelt, bei denen in den Trainingsdaten gleiche Eingabevektoren zu anderen Ergebnissen führten, wo also zwei gleiche Benchmarkläufe nicht die exakt gleiche Laufzeit hatten. Ausserdem ist zu beobachten, dass das Netzwerk einen Ausreisser, der bei x=0, y fast 0 zu sehen ist, ignoriert hat. Das hingegen scheint dem Netz unter JavaNNS nicht gelungen zu sein.

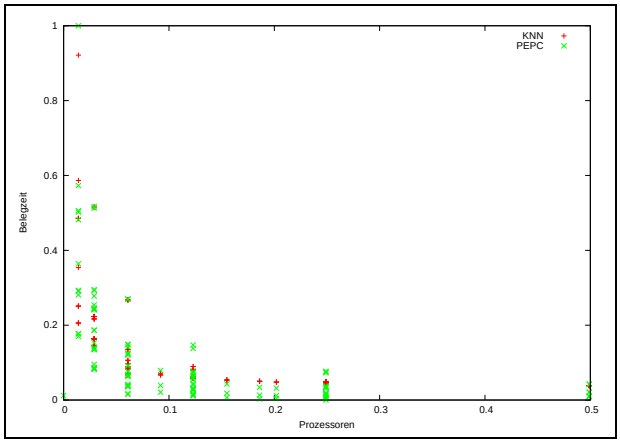


Abbildung 6.9: Punktewolke des FANN-Ergebnisnetzes: Das Netz hat keine direkte Information, dass es sich bei den Trainingsdaten um Kurvenscharen handelt.

6.4.3 Prognosetests

In den bisherigen Tests wurden nur Daten gelernt und wiedergegeben. Nun sollen erste Prognosen basierend auf den Daten von PEPC gemacht werden. Dazu wird jeweils ein zusammengehöriger Teil der Daten aus dem Training ausgeschlossen, um diese Daten durch das Netz zu prognostizieren und damit gleichzeitig seine Generalisierungsfähigkeit zu prüfen. Schematisch ist dieses Vorgehen in Abb. 6.10 dargestellt.

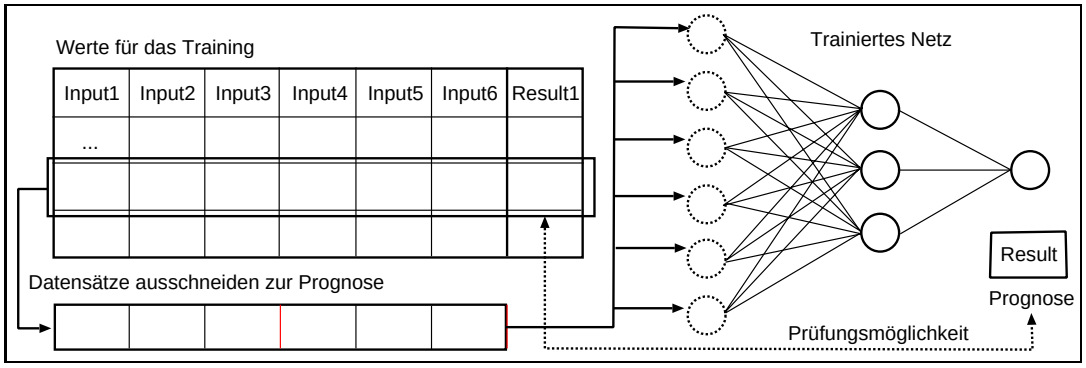
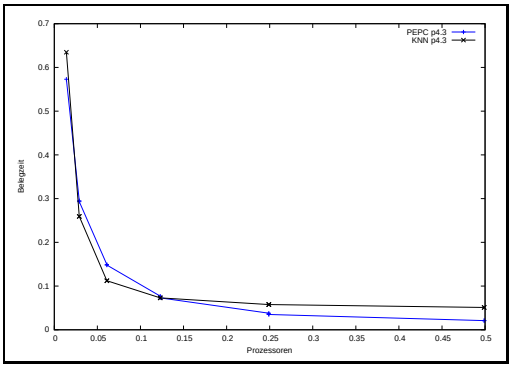


Abbildung 6.10: Für Prognosetests wird ein Teil der Trainingsdaten herausgeschnitten. Nach erfolgtem Training können diese Daten das Netz durchlaufen, so dass eine Prognose dieser Werte vorgenommen wird. Diese Prognose kann anhand des bekannten Ergebnisses geprüft werden.

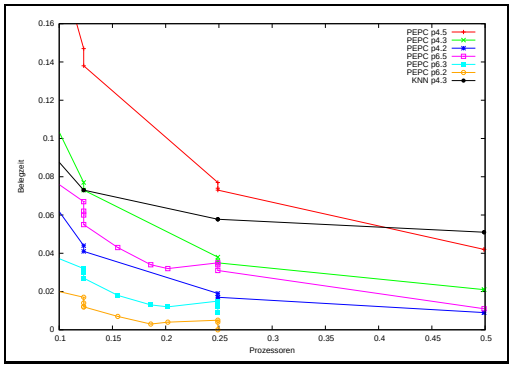
Für die folgenden Prognose wurde ein 6:3:1 Netz trainiert. Dieser Aufbau wurde gewählt, da mit dieser Topologie, wie z.B. in Abb. 6.5(a) auf Seite 61 zu sehen, bereits eine gute Wiedergabe der Trainingsdaten erfolgen konnte. Das Netz wurde mit allen Daten ausser den Daten für Power4 mit 3 Millionen Partikeln trainiert. Diese Datensätze wurden

dann als Testbeispiele zur Prognose herangezogen. Die Entscheidung fiel im Gegensatz zu den vorhergehenden Beispielen, bei denen meist die Daten der Power4 mit 5 Millionen Partikeln näher beleuchtet wurden, auf die Daten mit 3 Millionen Partikeln, da es für den ersten Prognosetest nicht sinnvoll erschien, einen Extremwert zu prognostizieren, den diese Kurve als langsamste bildet. Bzgl. der Daten mit 3 Millionen Partikeln ist eine gute Annährung an die tatsächlichen Werte zu erhoffen, da das Netz für die Power6 den Verlauf der entsprechenden Partikelanzahl kennt und von der Power4 die umgebenden Partikelgrößen von 2 und 5 Millionen beim Training mitberücksichtigt wurden. Es wurde wieder mit der Elliotfunktion in allen Schichten gearbeitet, da diese in den Tests zuvor jeweils die besseren Ergebnisse lieferte.

In Abb. 6.11(a) ist die Kurve der Prognose zu sehen. Der Kurvenverlauf nähert sich



(a) Prognose der Skalierungskurve der Power4 mit 3 Millionen Partikeln



(b) Zoom inkl. Abbildung der kompletten Original-PEPC-Kurvenschar

Abbildung 6.11: Prognose der Power4 mit 3 Millionen Partikeln: Die Abweichung der Prognose für größere Prozessorzahlen nach oben liegt in den fehlenden Referenzwerten begründet.

zunächst gut mit leichten Abweichungen der Originalkurve an, weicht dann aber für größere Werte nach oben ab. Das ist darin begründet, dass von der Power6 für 2 und 3 Millionen Partikeln leider keine Daten für die größte Prozessoranzahl vorlagen. Wie in Abb. 6.11(b) zu sehen ist, enden die Kurven bereits bei dem Prozessor-Wert von 0.25, was 512 CPUs entspricht. Die Prognose hat darüberhinaus also keine direkten Referenzpunkte mehr, sie scheint sich eher an der Kurve für Power4 mit 5 Millionen Partikeln zu orientieren oder dem Trend der Power6 für 3 Millionen Partikeln zu folgen, deren Kurve ebenfalls nicht abflacht.

Wie verhält sich ein KNN generell, wenn ihm unbekannte Eingabewerte präsentiert werden?
Um dieses Verhalten näher zu untersuchen, wurde ein entsprechendes Netz mit Power6-Daten trainiert, wobei für die Maschinenkennung nicht die reale Taktfrequenz, sondern ein fiktiver, mittlerer Wert von 0.5 benutzt wurde. Anschließend wurden die Power4-Daten abgefragt, wobei ebenfalls nicht die reale Taktfrequenz, sondern variierende Ma-

schinenkennungen um 0.5 herum von 0.1 bis 1.0 in Schritten von 0.1 benutzt wurden. In der Graphik 6.12 ist exemplarisch der Verlauf für die Partikelanzahl von 3 Millionen aufgeführt. Die Kurve mit der Maschinenkennung von 0.5 entspricht direkt den Power6-

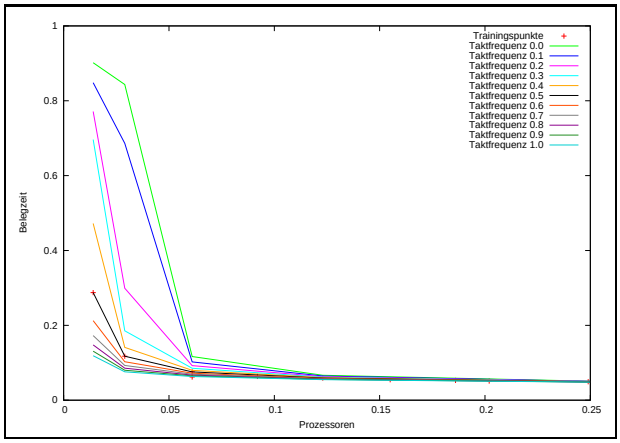


Abbildung 6.12: Ein Verhaltenstest für unbekannte Stützstellen macht das nicht-lineare Verhalten des Netzes deutlich.

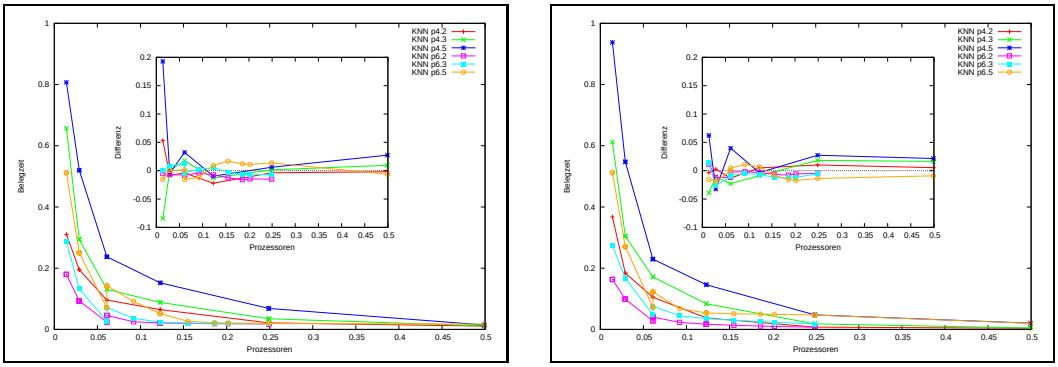
Daten, für die übrigen Werte wird die Kurve je mehr die Kennung abweicht immer stärker auseinandergezogen. Wie in dem Beispiel auch zu sehen, bedeutet eine niedrigere Kennung nicht unbedingt niedrigere Werte, das hängt von den Gewichten des Netzes ab, die ja auch negativ sein können und damit den Effekt umkehren.
Völlig unbekannte Werte lassen sich mit einem Neuronalen Netz nicht vorhersagen. Führt man z.B. ein Training mit Power6-Daten durch, um daraus Power4 Daten zu prognostizieren, kann man höchstens einen angenäherten Kurvenverlauf erwarten. Die Höhe der Kurve ist nicht linear abzubilden und muss mindestens durch einen Wert der Power4 bestimmt werden.

6.4.4 Prüfung des Trainingsalgorithmus

Es war zu sehen, dass sowohl bei der Wiedergabe von gelernten Daten wie auch bei der Prognose die Annäherung an die Originaldaten noch nicht ideal bzw. je nach Netztopologie mehr oder weniger gut war. Insbesondere war das Verhalten der Kurven nicht gut, wenn die Originalkurven sehr eng zusammenliegen, also z.B. für große Prozessor-Werte.

Da die Wahl des Trainingsalgorithmus natürlich einen großen Einfluss auf die Testergebnisse hat, wurde die Wahl von Resilient Propagation (Rprop) wie geplant überprüft und ergänzende Tests mit Backpropagation durchgeführt. Die Ergebnisse eines Tests mit der 6:3:1-Topologie sind in der Abb. 6.13(a) auf der nächsten Seite zu sehen.
Im Vergleich zu den Original-PEPC-Werten (Abb. 6.2 auf Seite 56) ist zu sehen, dass der Maximalwert der oberen Kurve bei 1 zwar auch nicht wiedergegeben wird, aber insbesondere die Problematik, die Rprop mit den Werten für größere Prozessorzahlen hat, treten hier viel weniger stark auf. Die Kurven nähern sich zwar auch einander an,

akkumulieren aber nicht in einem Punkt, was bei Rprop sogar schon recht früh geschieht.



(a) Training mit der Elliotfunktion

(b) Training mit der Sigmoidfunktion

Abbildung 6.13: Trainingsergebnisse mit Backpropagation: Die Ergebnisse insbesondere für größere Prozessorwerte haben sich verbessert. Zum Vergleich mit den Original-PEPC-Werten sind die Fehlerkurven mit aufgezeichnet.

Die Abb. 6.13(a) stammt aus einem Lauf mit der Elliotfunktion. Zum Vergleich wurde auch noch einmal mit der Sigmoidfunktion trainiert (in der rechten Abb. 6.13(b) zu sehen).
Bei Rprop waren die Unterschiede zwischen den beiden Übertragungsfunktionen relativ groß, das ist hier nicht mehr der Fall. Mit der Sigmoidfunktion läßt sich der Extremwert der oberen Kurve besser erreichen, jedoch nähern sich die Kurven wieder stärker einander an.
Um den Vergleich der Werte untereinander sowie mit den Original PEPC-Werten zu erleichtern, wurden zusätzlich die Differenzen der KNN-Werte zu PEPC aufgezeichnet. Es ist zu sehen, dass die größten Abweichungen bei den aussen liegenden Kurven bestehen. Zudem ist gut der beschriebene Unterschied der beiden Netze zu erkennen, also z.B. die größere Abweichung des Elliot-Netzes für kleine Prozessorzahlen.

Damit hat sich Backpropagation für diese Trainingsaufgabe als der günstigere Algorithmus erwiesen.

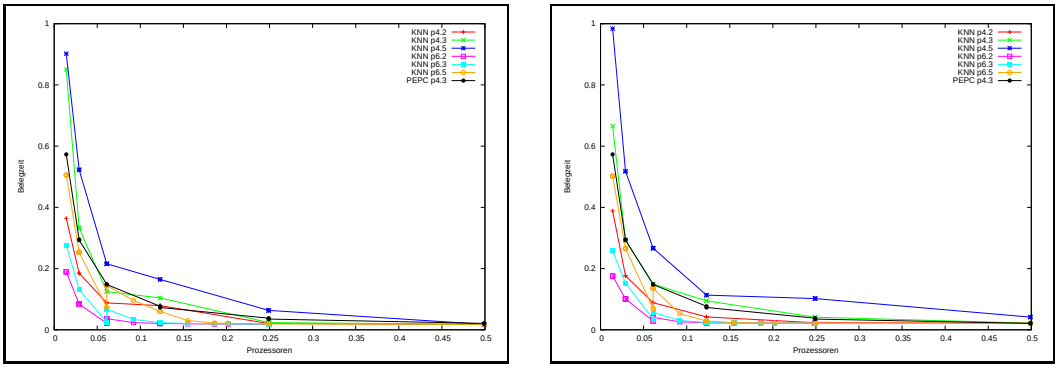
Lernrate und Momentum

Die Lernrate ist bei den Tests mit Backpropagation ein fest eingestellter Parameter. Die Default-Lernrate von FANN ist 0.7, ein Momentum ist nicht eingeschaltet. Mit dieser Konfiguration wurden die beschriebenen Tests zu Backpropagation durchgeführt. Um die Fehlerkurve noch feingranularer abzutasten und damit den möglichen Probleme von Backpropagation entgegenzuwirken, wurden zusätzlich Tests mit einer Lernrate von 0.2 und ebenso mit eingeschaltetem Momentum von 0.2 und 0.7 durchgeführt.
Die Testergebnisse verbesserten sich dadurch aber nicht weiter, der Backpropagation-

Algorithmus scheint also nicht auf eines der beschriebenen Probleme wie z.B. das Überspringen des Momentums gestoßen zu sein.
Allerdings konnte bei diesen wiederholten Tests beobachtet werden, dass die Ergebniskurven nahezu konstant blieben, was bei RProp noch nicht ganz der Fall war. Der Algorithmus scheint für das vorliegende Trainingsproblem also auch besser zu konvergieren.

Prognose mit Backpropagation

Die Kurven wurden in den zuletzt beschriebenen Tests bereits sehr gut angenähert, es soll aber auch für diese Konfiguration noch die Generalisierungseigenschaft überprüft und damit eine Prognose durchgeführt werden.
Als Beispiel wird wieder der Datensatz der Power4 mit 3 Millionen Partikeln genutzt, der aus dem Training entfernt und anschließend gemeinsam mit den gelernten Daten abgefragt wird. Der Test wurde sowohl mit der Aktivierungsfunktion Elliot wie auch Sigmoid durchgeführt, s. Abb. 6.14.

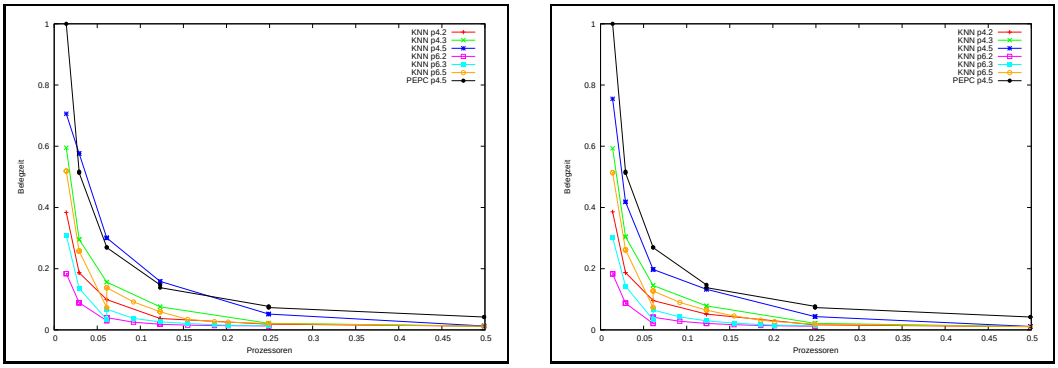


(a) Training mit der Elliotfunktion (b) Training mit der Sigmoidfunktion

Abbildung 6.14: Prognose mit Backpropagation für Power4 mit 3 Millionen Partikeln: Die Prognose mit Elliotfunktion nähert sich zunächst zu stark der Funktion mit 5 Millionen Partikeln an, verbessert sich dann aber. Die Prognose mit der Sigmoidfunktion ist sehr gut gelungen.

Über die Ergebnisgraphiken der beiden KNN ist jeweils die Original PEPC-Kurve für den zu prognostizierenden Datenteil gelegt.
Die Prognose mit Hilfe des Netzes mit der Elliotfunktion nähert sich zunächst zu stark der Funktion mit 5 Millionen Partikeln an, verbessert sich dann aber. Die Prognose mit der Sigmoidfunktion ist sehr gut gelungen.
Natürlich ist auch zu erkennen, dass bei der recht kleinen Gesamt-Datenmenge die für die Prognose zuvor entfernten Daten auch das Ergebnis der übrigen Kurven beeinflussen.

Abschließend soll mit Backpropagation auch noch die Prognose des Extremwerts bei 5 Millionen Partikeln der Power4 gezeigt weden. Die sich ergebenden Kurven sind in



(a) Training mit der Elliotfunktion

(b) Training mit der Sigmoidfunktion

Abbildung 6.15: Prognose mit Backpropagation für Power4 mit 5 Millionen Partikeln: Auch diese Funktion, die den Extremwert der Kurvenschar bildet, kann mit einem Backpropagation-Netz prognostiziert werden.

Abb. 6.15 zu sehen. In 6.15(a) ist eine sehr gute Prognose zu sehen, die allerdings bei niedrigen Prozessorzahlen zu stark abflacht. Die Sigmoidfunktion hingegen bewirkt eine bessere Annäherung des Kurvenverlaufs, liegt aber insgesamt zu niedrig.

6.4.5 Ergänzende Tests

In diesem Abschnitt sollen noch exemplarisch einige Tests aufgeführt werden, die mit zur Beurteilung der KNN für das Benchmarking allgemein bzw. für die Tests mit PEPC beitragen, aber nicht mehr im Detail erläutert werden sollen.

Dazu zählen zum einen erweiterte Netztests:

- Tests mit weiteren Trainingsalgorithmen, z.B. dem Backpropagation-Algorithmus in der Batch-Variante, bei dem die Gewichtsveränderung nicht nach jeden Trainingsbeispiel, sondern nach einem Satz von Trainingsbeispielen akkumuliert vorgenommen wird. Diese Variante lieferte ähnlich gute Ergebnisse wie die inkrementelle Version und könnte damit z.B. auch genutzt werden im Falle einer Parallelisierung des Trainings.

- Tests mit weiteren Übertragungsfunktionen neben der Sigmoid- und der Elliot-Funktion: Sigmoid_Stepwise und Gaussian resultierten nicht in guten Ergebnissen, mit Gaussian waren die PEPC-Kurven zumindest in 500000 Epochen nicht zu erlernen.

- Tests mit kleineren Netzen, um die Beiträge der einzelnen Neuronen besser einschätzen zu können, z.B. Tests mit 5:x:1-Topologie unter Entfernung der Nodes. Es ergab sich jedoch, dass die Nodes doch deutlich Einfluss auf die Ergebnisse haben, da ein Teil der Rechnungen auf verteilten Nodes stattgefunden haben, selbst wenn sie mit ihrer Anzahl an CPUs auf einem Knoten hätten laufen können. Dadurch

haben sich durch verlängerte Kommunikationszeiten auch die Laufzeiten signifikant verändert, was das Netz nur durch dieses Neuron lernen kann.

- Tests mit größeren Netzen: Es wurden Läufe hinzugenommen, die mit Profilern instrumentiert waren und dadurch längere Laufzeiten hatten oder auch mit besonders optimiertem Compiler und dadurch schneller waren. Um dem Netz eindeutigere Informationen zu geben, wurde die jeweilige Art des Profilers oder auch des Compilers auf Neurone abgebildet und das Netz dadurch um 5 auf 11 Eingabeneurone erweitert. Auf diese Weise konnte auch diese Information gelernt werden.

- Tests mit anderer Topologie: z.B. 6:3:3:1, also mit zwei verdeckten Schichten. Dieses Netz hat natürlich wesentlich mehr Verbindungen als die dreischichtigen Netze. Das hat zur Folge, dass die Daten tatsächlich sehr gut gelernt werden. Dem steht aber gegenüber, dass der Aufwand für das Training des Netzes erheblich ansteigt. Das ist bei der vorliegenden Menge an Trainingsdaten noch nicht sehr deutlich, längere Tests haben aber gezeigt, dass sich die Trainingsdauer auch verdoppeln kann.

Zu den Netztests hinzu kamen noch weitere Tests zum Verhalten von Zwischenwerten zwischen Stützstellen, dies entspricht im Grunde einer Prognose. Zudem wurden noch Fehlerkurven für Trainings- und Testdaten genauer betrachtet, ein Beispiel einer solchen Fehlerkurve ist in Abb. 6.16 zu sehen. Den Referenzdatensatz bilden die Daten der Power4 für 5 Millionen Partikel, die aus dem Training entfernt wurden. Es handelt sich also um das Fehlerverhalten dieser Prognose. Der Fehler der Trainingsdaten sinkt sehr

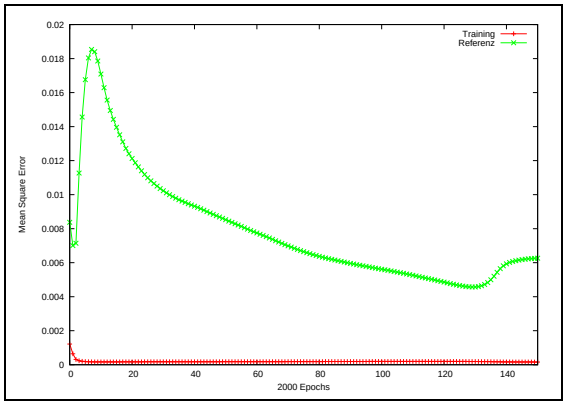


Abbildung 6.16: Fehlerverhalten für Trainings- und Referenzdaten: Der Fehler des Trainingsdatensatzes sinkt sehr früh stark ab, der Fehler des Referenzdatensatz sinkt langsamer, in dem Moment, in dem er wieder ansteigt, sollte das Training beendet werden.

schnell, der Fehler der Referenzdaten liegt, wie zu erwarten, über den Trainingsdaten, sinkt nach einem anfänglichen Peak aber auch. Gegen Ende des Ausschnitts ist zu sehen, dass dieser Fehler wieder ansteigt, sich das Netz also zu sehr den Trainingsdaten anpasst. Hier sollte das Training beendet werden.

Um die obigen Ergebnisse zur Prognose zu untermauern, wurden noch weitere Tests zur Prognose gemacht, so wurden z.B. auch die 3 Millionen Werte komplett prognostiziert, die vom KNN korrekt zwischen die Kurven für 2 und 5 Millionen beider Prozessortypen gelegt wurden. Es wurden zudem noch Prognosen an der Extremstelle von 1024 Prozessoren gemacht, die jedoch nicht so gut funktionieren. Es wurde schon oben gesehen, dass aufgrund der vorhandenen Datenlage dieser Punkt schwierig zu lernen ist. Diese Tatsache bleibt auch mit dem Backpropagation-Algorithmus bestehen.

6.4.6 Bewertung der Testreihe mit PEPC

Insgesamt haben die Tests gezeigt, dass sich die Benchmarking-Werte von PEPC durchaus mit Hilfe von Neuronalen Netzen erlernen lassen und auch Prognosen möglich sind. Es hat sich die Wahl für die feed-forward Architektur mit verdeckten Schichten bestätigt, wobei eine verdeckte Schicht schon ein gutes Training ermöglichte und sich der Backpropagation-Algorithmus als die beste Wahl herausstellte. Die Wahl der Aktivierungsfunktion fiel auf die Elliot- bzw. Sigmoidfunktion, wobei sich keine der beiden als die eindeutig bessere herausgestellt hat.
Natürlich lassen sich auch noch Schwächen erkennen. Es wurde deutlich, wie wichtig eine gute Datenbasis für das Lernen eines KNN ist, denn sein Wissen basiert allein auf dieser Grundlage.
Die präsentierten Netze wurden mit allen für diesen Benchmark zur Verfügung stehenden Daten trainiert. Die Projekte DEISA und PRACE sind noch nicht abgeschlossen, es werden also noch mehr Daten hinzukommen, insbesondere in PRACE werden mit der Zielrichtung Petascale-Computing größere Messungen vorgenommen werden. Die Hoffnung ist, dass insbesondere auch die Werte für die hohen Prozessorzahlen mit dieser erweiterten Datenbasis noch besser gelernt werden können.

Doch in Hinblick auf ein generelles Konzept sind natürlich noch Erweiterungen erforderlich. Der Kennung des Benchmarks PEPC wurde in dieses 6:x:1-Netz nur über die Anzahl der Partikel eingebracht. Ein anderer Benchmark könnte natürlich nicht auf der Basis dieses Trainings prognostiziert werden, auch dann nicht, wenn es sich um einen Partikel Code handelt. Es sind, wie schon in der Analyse des Benchmarking-Prozesses beschrieben, weitere Informationen nötig, um den Benchmark zu beschreiben bzw. ein generelles Netz zu erstellen.
Daher soll als nächstes das Kommunikationsverhalten mit in das Netz eingebracht werden. Dazu wird aber zunächst ein anderer Benchmark betrachtet, nämlich der Intel MPI Benchmark (IMB), ein Low-Level-Benchmark zur Messung der Kommunikationsgeschwindigkeit eines Rechners.

6.4.7 Tests mit IMB

In den meisten parallelen Programmen, so auch in PEPC, wird die Message Passing Interface-Bibliothek (MPI) genutzt, um Nachrichten zwischen den beteiligten Prozessoren zu versenden. Bei diesen Nachrichten kann es sich z.B. um den Austausch von In-

formationen zwischen Randpunkten bei einer Gebietszerlegung handeln. In diesem Fall würden in der Regel physikalisch benachbarte Prozessoren miteinander kommunizieren. Es ist aber auch denkbar, dass ein Prozessor eine Nachricht gleichzeitig an alle anderen Prozessoren senden muss, z.B. bei einem Master-Server Algorithmus, oder dass sogar alle Prozessoren gleichzeitig eine Information an alle Prozessoren schicken, wenn z.B. zu einem bestimmten Zeitpunkt jeder Prozessor die Rechenergebnisse der anderen benötigt. Diese beiden Kommunikationsarten werden als globale Kommunikation bezeichnet, im Gegensatz zur Punkt-zu-Punkt Kommunikation im ersten Fall.
Es gibt also verschiedene Möglichkeiten zu kommunizieren, exemplarisch seien die folgenden dazugehörigen Funktionen des MPI-Standards genannt:

- `MPI_Send` und `MPI_Recv` dienen zur Punkt-zu-Punkt Kommunikation, d.h. zwei Prozessoren kommunizieren miteinander. Diese Varianten sind blockierend, die Prozessoren warten auf die Beendigung des Nachrichtenaustauschs. Entsprechend existieren die nicht-blockierenden Varianten `MPI_Isend` und `MPI_Irecv`.

- `MPI_Bcast` realisiert die beschriebene Situation, dass ein Prozessor eine Nachricht an alle anderen Prozessoren versendet, also einen *Broadcast* vornimmt.

- `MPI_Gather` realisiert die umgekehrte Funktionalität zu `MPI_Bcast`, ein Prozessor sammelt die Daten aller anderen Prozessoren ein.

- `MPI_Allgather` sammelt ebenfalls die Daten der beteiligten Prozessoren ein, allerdings sammeln hier alle Prozessoren die Daten der anderen, so dass eine Multi-Globale Kommunikation entsteht.

- `MPI_Alltoall` ist ebenfalls eine Multi-Globale Operation, wobei aber nicht jeder Prozessor die gesamten Daten der anderen Prozessoren empfangen möchte, sondern nur einen für ihn bestimmten Teil.

Zu den zuletzt genannten Operationen gibt es noch die Varianten `MPI_Gatherv`, `MPI_Allgatherv` und `MPI_Alltoallv`, die unterschiedliche Datengrößen zulassen.

Die Geschwindigkeit dieser Kommunikationsarten auf einem Rechner ist entscheidend für die Geschwindigkeit einer Anwendung, die MPI nutzt. Um auszumessen, wieviel Zeit die einzelnen Funktionen auf einem Rechner benötigen, wurde der Intel MPI Benchmark (IMB) entwickelt. Es handelt sich beim IMB um einen Low-Level-Benchmark, der eine Reihe von Kernen von MPI-Kommunikationsfunktionen enthält, die auf diese Weise einzeln ausgemessen werden können.
IMB enthält unter anderen folgende Benchmark-Kerne für verschiedene Kommunikations-Schemata:

- *PingPong* zur Ausmessung von Punkt-zu-Punkt Kommunikation,

- *Bcast* zur Ausmessung von Broadcast-Kommunikation,

- *Gather* zum Ausmessen der globalen Funktion zum Sammeln von Daten,
- *Allgather* und *Alltoall* sowie die entsprechenden Vektorvarianten für die Multi-Globalen Funktionen.

IMB ermöglicht es, diese Benchmarks mit mehr als einer Prozessgruppe zu starten. Beim *PingPong* wird eine Nachricht zwischen zwei Prozessoren hin und her gesendet. Startet man diesen Kern mit mehr als zwei, beispielsweise vier, Prozessoren, werden zwei Gruppen mit je zwei Prozessoren erzeugt, die jeweils untereinander parallel zu den anderen Prozessoren den *PingPong*-Benchmark durchführen. Diese Gruppenversionen der Benchmarks sind von den einzelnen Kernen zu unterscheiden, da natürlich auch diese parallele Arbeit auf einem Knoten die Zeiten beeinflussen kann. Bezeichnet werden die Gruppenbenchmarks analog zu den einzelnen Benchmarks, aber mit einem vorangestellten *Multi-*, also z.B. *Multi-PingPong*.
Zudem gibt es verschiedene Möglichkeiten der Konfiguration von IMB. Die wichtigste Einstellung neben der Auswahl der Anzahl an beteiligten Prozessoren ist die Angabe der Nachrichtenlänge, die jeweils verschickt werden soll. Es wird einen erheblichen Unterschied in der Dauer der Kommunikation machen, ob eine sehr kurze Nachricht von einem Byte oder die maximal mögliche Einstellung von 4 MByte verschickt werden muss.

Tests für ausgewählte IMB-Kerne

Es lagen eine Reihe von Messungen mit IMB vor, die auf Power6 vorgenommen wurden. Für die Tests mit KNN und IMB wurden drei Operationen herausgegriffen, die auch in PEPC genutzt werden, um eine Grundlage für die Verbindung der beiden Benchmarks zu haben.
Die Läufe wurden auf 128, 256 und 512 Prozessoren durchgeführt, wobei die Nachrichtengrößen 8192 Byte, 16384 Byte und 32768 Byte betrugen.
Ein Auszug, der aus den Läufen auf Power6 herausgefilterten Daten sieht wie folgt aus:

```
Multi-PingPong 64 2 8192 1000 10.78
Multi-PingPong 64 2 8192 1000 10.64
Multi-PingPong 64 2 16384 1000 16.81
Multi-PingPong 64 2 16384 1000 18.44
```

Die erste Spalte beinhaltet den Benchmark-Kern, die zweite die Anzahl parallel laufender Gruppen, die dritte die Anzahl Prozessoren pro Gruppe, die vierte die Nachrichtengröße, die fünfte die Anzahl an Wiederholungen der Kommunikation (um das Gewicht von Ausreissern zu relativieren) und die sechste die benötigte Zeit. Für jeden Kern lagen 2670 Messwerte vor.

Aus diesen Daten wurden jeweils Trainingsdaten für die verschiedenen Kerne gefiltert und wie schon bei PEPC auf den Bereich von [0,1] skaliert. Dann wurde ein Netz aufgebaut mit den beschriebenen Spalten als Eingabevektor, wobei die letzte Spalte, also die Belegzeit, wieder die Ausgabe des Netzes bilden soll.

Mit dieser Konfiguration wurde eine Reihe an Tests analog zu den PEPC-Tests durchgeführt. Das bedeutet, dass die bekannten Parameter durchgetestet wurden:

- Anzahl an versteckten Neuronen
- Aktivierungsfunktionen Sigmoid und Elliot
- Trainingsalgorithmen Resilient Propagation und Backpropagation
- Länge des Trainings
- Lernrate und Momentum
- Anzahl verdeckter Schichten

Die beste Konfiguration soll hier jeweils aufgeführt werden.

Bei den Tests mit *PingPong* ergab sich, dass die Trainingsdaten mit einer Reihe von Netzen gut gelernt werden konnten. *PingPong* ist gegenüber den anderen Kernen auch etwas einfacher, da immer zwei Prozessoren kommunizieren und das Neuron der Prozessoranzahl somit konstant ist. In Abb. 6.17 ist das Netz mit 5:3:1-Topologie mit Backpropagation-Training und Sigmoid-Aktivierungsfunktion zu sehen. Dargestellt sind die Trainingsdaten und die trainierte Wiedergabe des KNN. Die drei Balken markieren die Anzahl an Gruppen, also die Messungen mit 128, 256 bzw. 512 Prozessoren. Innerhalb dieser Balken gibt es jeweils drei Schwerpunkte, die durch die unterschiedlichen Zeiten für die verschiedenen Nachrichtengrößen entstehen und vom KNN auch entsprechend durch einen Ergebniswert erkannt werden.

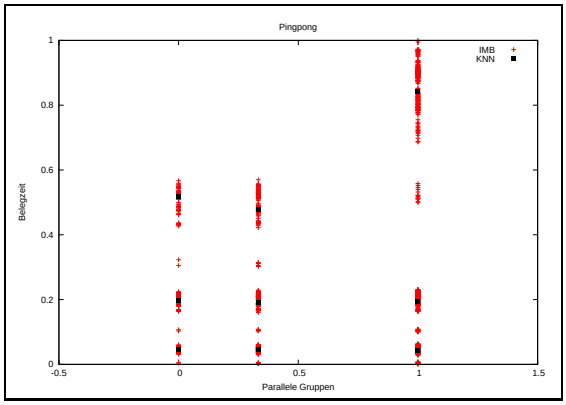
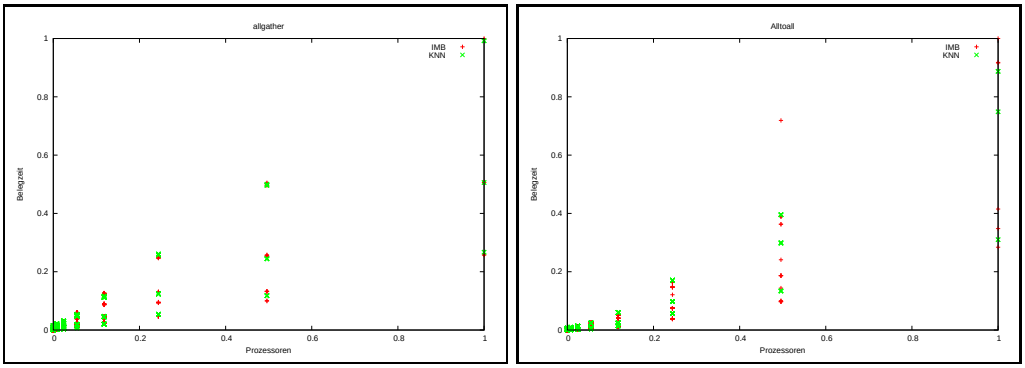


Abbildung 6.17: Punkt-zu-Punkt Kommunikation *PingPong*: In der Datenbasis befanden sich drei Prozessorgrößen, die Schwerpunkte der Läufe pro Prozessoranzahl werden von den verschiedenen Nachrichtengrößen verursacht und werden vom KNN erkannt.

Bei den Tests mit *Allgather* und *Alltoall* stellte sich jeweils als bestes Netz ein 5:3:1-Netz mit Backpropagation-Training und Sigmoid-Aktivierungsfunktion heraus.



(a) Lernerfolg für *Allgather* (b) Lernerfolg für *Alltoall*

Abbildung 6.18: Multi-Globale Kommunikation mit *Allgather* und *Alltoall*: Bei *Allgather* sind die drei Kurven der unterschiedlichen Nachrichtengrößen erkennbar, *Alltoall* fächert sich aufgrund der parallel kommunizierenden Gruppen stärker auf.

Bei *Allgather* sind drei Kurven für die unterschiedlichen Nachrichtengrößen erkennbar. Es hätte auch erwartet werden können, dass sich die Kurven noch mehr aufsplitten, da hier zusätzlich zu der Nachrichtengröße noch die Anzahl an Gruppen eine Rolle spielt. Das scheint sich bei *Allgather* aber nicht auf die Laufzeiten auszuwirken. Bei *Alltoall* hingegen ist diese größere Streuung zu sehen, der Lernerfolg ist auch nicht so groß wie bei *Allgather*, wobei der oberste Wert unter 0.5 auf der Prozessorenachse als Ausreisser identifiziert wurde. Die Darstellung ist hier schwierig, da tatsächlich drei Nachrichtengrößen kombiniert mit 8 Gruppenkombinationen (2,4 bis 256 parallel Gruppen) dargestellt werden müssten, also 24 Kurven. Daher sind zur Übersichtlichkeit nur die Originaldaten den gelernten Daten gegenübergestellt worden.

Test für einfache globale Kommunikation

Es wurde deutlich, dass die parallel laufenden Gruppen zu einer großen Streuung der Daten führen können. Da bei PEPC keine globale Kommunikation parallel auf mehreren Gruppen verwendet wird, werden dort nur die Messungen mit jeweils einer Gruppe benötigt. Das Trainingsergebnis einer solchen Konfiguration ist in Abb. 6.19 auf der nächsten Seite am Beispiel *Allgather* aufgeführt.
Es ist zu sehen, dass die Kurven sehr gut gelernt werden konnten. Dabei muss gesagt werden, dass die zu lernende Funktion durch die Trainingsdaten mit 9 Messungen pro Kern (drei Prozessorgrößen mit je drei Nachrichtengrößen) sehr klar definiert ist.

Test für die Kombinationen von IMB-Kernen

Abschließend wurde noch ein Test unternommen, die verschiedenen Kerne gemeinsam in einem Netz zu trainieren. Dabei hilft ein Neuron mit der Angabe des Kernes zu erkennen, dass die Eingabewerte nicht widersprüchlich sind, sondern tatsächlich unterschiedliche Quellen haben. Der Lernerfolg ist ein Abb. 6.20 auf der nächsten Seite zu sehen.

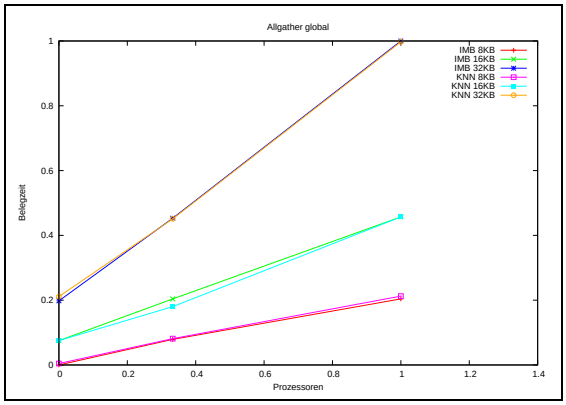


Abbildung 6.19: Globale Kommunikation mit *Allgather*: Es existieren keine parallel kommunizierenden Gruppen, sondern es wird über die volle Prozesseranzahl mit *Allgather* kommuniziert. Die Kurven konnten vom KNN sehr gut gelernt werden.

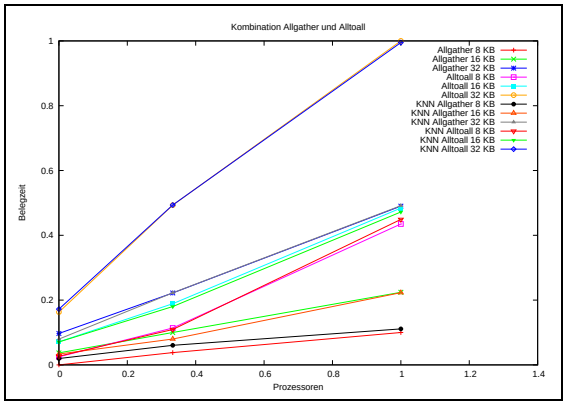


Abbildung 6.20: IMB Kombination *Alltoall und Allgather*: Die Benchmark-Kerne wurden anhand eines zusätzlichen Neurons für den Benchmark-Typ in einem Netz trainiert.

Damit ist also schon ein einfacher Test unternommen worden, mit einem Netz verschiedene Benchmarks zu erlernen, realisiert allerdings durch ein spezielles Neuron. Ziel ist es aber, die unterschiedlichen Benchmarks durch ihre Eigenschaften zu erkennen. Dazu hätte man hier statt des direkten Benchmark-Neurons zwei Neurone einführen können, die für den Prozentsatz an *Alltoall-* bzw. *Allgather*-Kommunikation stehen. In diesem Beispiel ergibt sich dadurch aber noch kein großer Unterschied, das dadurch größere Netz hätte sogar noch mehr Speicherkapazität. Anders sieht es aus, wenn man einen IMB-Kern in Zusammenhang mit einem komplexeren, also nicht ausschließlich Kommunikationsabhängigen, Benchmark wie PEPC kombiniert.

6.4.8 Bewertung der Testreihe mit IMB

Die vorgenommenen Tests mit IMB haben gezeigt, dass diese ebenfalls von einem KNN gelernt werden können. Das war nach den vorhergehenden Tests mit den komplexeren Daten von PEPC auch zu erwarten. Eine Schwierigkeit hätte die doppelte Information von parallel laufenden Gruppen und Nachrichtengröße sein können, was sich aber nicht als Problem erwiesen hat.

Damit hat sich gezeigt, dass IMB als ein Basis-Benchmark für die Messung von Grundwerten einer Maschine auch in einem Künstlichen Neuronalen Netz nutzen kann. Es erscheint also möglich ein Gesamtbild einer Maschine über Low-Level-Benchmarks darzustellen bzw. einem KNN bekannt zu machen.

Die angesprochene Kombination von PEPC und IMB soll nun im folgenden Abschnitt vorgenommen werden.

6.4.9 Tests mit der Kombination von PEPC und IMB

Ziel der Kombination von PEPC und IMB ist zum einen der Test der KNNs bzgl. komplexerer Zusammenhänge. Zum anderen ist es möglich, dass durch die zusätzliche Information der Kommunikationsabhängigkeit eine genauere Prognose für PEPC erreicht werden kann.

Vorgehen

Es liegen Datensätze für die bereits beschriebenen Eingabevektoren für PEPC und IMB vor, sie sind in Abb. 6.21 noch einmal dargestellt. Diese beiden Vektoren müssen nun zu einem gemeinsamen Vektor zusammengefügt werden.

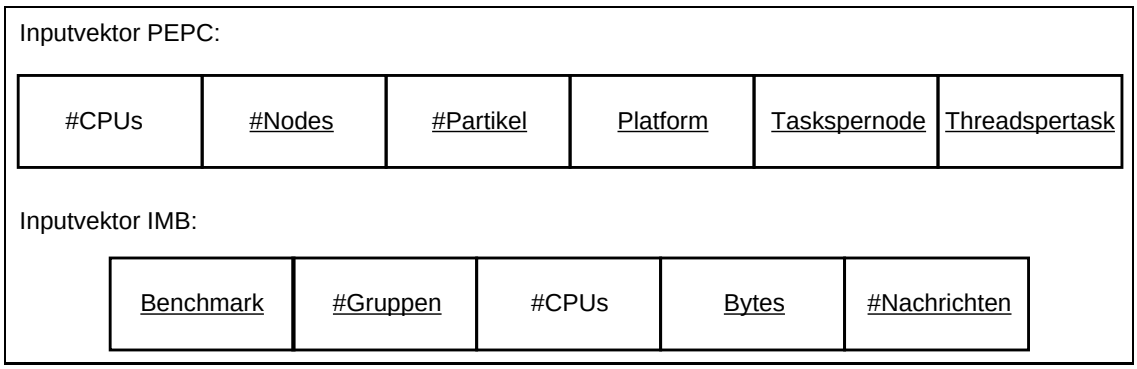


Abbildung 6.21: Darstellung der Eingabevektoren der KNN von PEPC und IMB aus den vorhergehenden Testreihen.

Werden die Vektoren verglichen, fällt zunächst einmal fällt auf, dass beide Eingabevektoren nur die Anzahl an CPUs als gemeinsame Kategorie haben. Die Frage ist daher

wie aus diesen beiden Datensätzen ein gemeinsamer Eingabevektor für ein kombiniertes Neuronales Netz erzeugt werden kann.
Dazu werden im Folgenden die einzelnen Werte nach ihren Klassen sortiert betrachtet:

Klasse Rechner Zur Klasse *Rechner* zählt bei PEPC das Neuron für die Plattform. Ein analoges Neuron wurde bei den Tests mit IMB nicht genutzt, da nur mit den Werten einer Plattform, IBM Power6, gearbeitet wurde. Es ist aber kein Problem, dieses Neuron auch für die IMB-Werte mit aufzunehmen, um die Kombinationstests für verschieden Plattformen durchführen zu können.

Klasse Programm Zur Klasse *Programm* zählen zum einen die Laufzeitparameter der Benchmark-Programme, also neben der Anzahl der CPUs, die beide Benchmarks nutzen auch die Anzahl der Nodes, Taskspernode und Threadspertasks von PEPC sowie auch die Größe und Anzahl der Nachrichten und die Anzahl der parallel kommunizierenden Gruppen bei IMB. Zum letztgenannten Punkt, der Gruppenanzahl, wurde schon festgestellt, dass dieser Wert im Zusammenhang mit PEPC keine Rolle spielt, da dort keine Gruppen parallel kommunizieren. Das Neuron wird also keinen Mehrwert für die Prognose von PEPC bringen und kann somit weggelassen werden. Die Nodes, Taskspernode und Threadspertasks könnten für IMB ergänzt werden, sofern sie benötigt werden. Die Größe und Anzahl der Nachrichten, die bei PEPC verschickt werden, können durch Werkzeuge (wie z.B. der High Perfomance Monitor von IBM) während des Programmlaufs ermittelt und somit auch für PEPC in das Netz aufgenommen werden.
Zum anderen wird in dieser Klasse das eigentliche Programm genauer beschrieben. Bei IMB gab es dafür ein eigenes Neuron, die Kombinationstests von PEPC und IMB werden zunächst nur mit einem IMB-Kern durchgeführt, so dass dieses Neuron nicht mehr notwendig ist. Sofern ein zweiter Kern hinzugefügt wird, sollte dieser über seine Kommunikationsart beschrieben werden.
Bei PEPC lag zur Beschreibung des Programms die Anzahl der Partikel vor, also im Grunde eine Problemgröße. Dieses Neuron ist daher entscheidend, da die Laufzeit natürlich von der Problemgröße abhängt. Eine direkte Entsprechung gibt es dazu allerdings bei IMB nicht, die Frage ist also, wie ein Neuron in ein solches kombiniertes KNN eingebunden werden kann, das nur bei einem der beiden Benchmarks eine Rolle spielt.
Dazu kommen verschiedene Möglichkeiten in Betracht. Die erste Idee war, das Neuron aufzunehmen und für den Benchmark bei dem es eigentlich keine Rolle spielt, also in diesem Fall IMB, zu variieren. Das bedeutet, dass die Werte, die bei PEPC auftreten (2-,3-,5-Millionen), auch für IMB angegeben werden, wobei aber jeweils die gleiche Belegzeit zugeordnet wird. Alternativ könnte auch ein Maximal- und ein Minimalwert mit gleicher Belegzeit spezifiziert werden. Auf diese Weise soll dem Netz mitgeteilt werden, dass dieses Neuron für diesen Benchmark keine Rolle spielt. Die Idee wurde jedoch verworfen, da es z.B. einen Benchmark geben kann, der den Werten von IMB ähnlich sein kann, für den dieses Neuron jedoch sehr wohl eine Rolle spielt. Zudem erscheint es nicht günstig, die Eingabewerte auf diese Weise künstlich zu verändern.
Da die Anzahl der Partikel ja letzlich für eine Problemgröße steht, ist es interessant, die

Problemgröße von IMB festzustellen. Es wäre also besser, nicht diese für PEPC spezifische Angabe zu benutzen, sondern einen eher generischen Wert. Dafür würde sich z.B. die Anzahl der insgesamt ausgeführten Floating-Poing Operationen (MFlop) anbieten. Für PEPC kann dieser Wert ebenfalls über Werkzeuge während des Programmlaufs ermittelt werden. Da IMB ausschließlich die entsprechende Kommunikation durchführt, wäre die Last, also die MFlop-Rate, gleich 0 und kann so als beschreibender Wert des Benchmarks in das Netz aufgenommen werden.

Klasse Messung Zur Klasse *Messung* gehört die Belegzeit der Programme auf dem Rechner, die in den bisherigen Tests für das Ausgabeneuron genutzt wurde, um Skalierungskurven zu erhalten. Das soll auch bei der Kombination der Fall sein.

Die beschriebenen Ergebnisse sind in Abb. 6.22 noch einmal schematisch dargestellt.

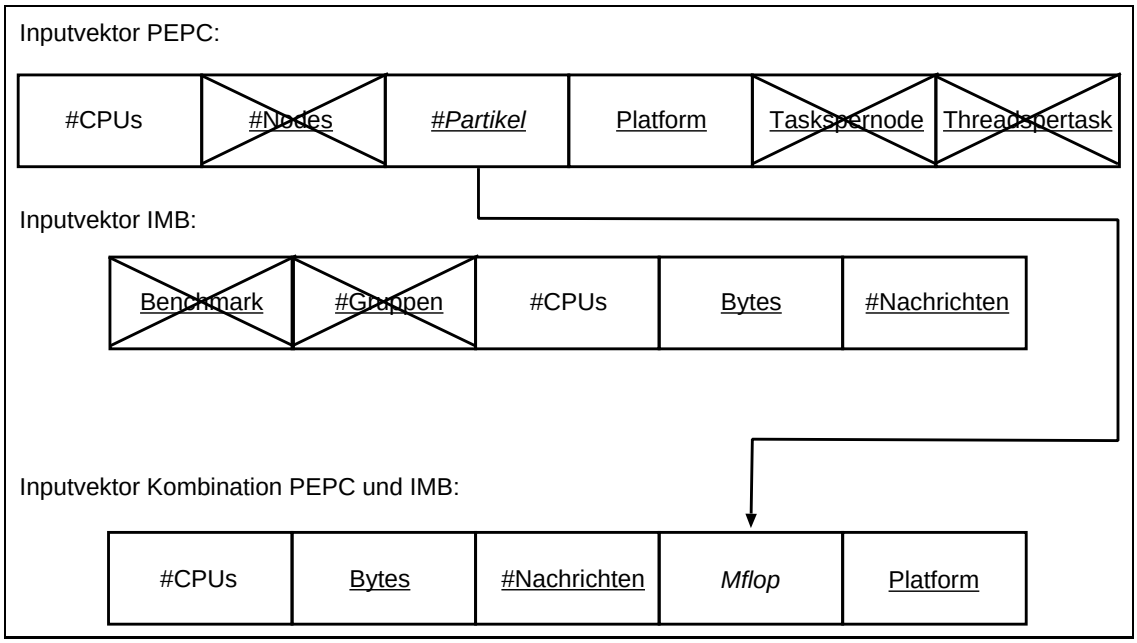


Abbildung 6.22: Der Eingabevektor für die Kombination von PEPC und IMB entsteht aus den einzelnen Eingabevektoren: Nicht benötigte Neurone wie die Anzahl der Gruppen können weggelassen werden, andere Werte wie die Nachrichtengröße müssen beim Benchmark, bei dem sie fehlen, hier PEPC, ergänzt werden. Benchmark-spezifische Werte wie die Anzahl Partikel müssen auf generische Werte abgebildet werden.

Zudem ist in Abb. 6.22 zu sehen, dass einige Laufzeitparameter von PEPC, nämlich die Anzahl der Nodes sowie die Taskspernode und Threadspertasks, ebenfalls nicht in den kombinierten Eingabevektor übernommen wurden.
Das ist begründet in der Datenlage, die für einen Kombinationstest zur Verfügung steht. Es werden für die Trainingsdaten die Ergebnisse von instrumentierten Läufen für PEPC benötigt, um an Werte wie den Kommunikationsanteil oder die MFlop-Rate zu gelan-

gen. Solche detaillierten Rechnungen wurden erst innerhalb des PRACE-Projektes vorgenommen, sie liegen daher nur für Läufe auf IBM Power6 mit 5 Millionen Partikeln vor. Zudem gab es dort nur Läufe bei denen Anzahl CPUs und Anzahl Nodes korrelierten und auch Taskspernode und Threadspertasks konstant waren. Daher geben diese Werte keine zusätzliche Information und werden bei den hier gezeigten Untersuchgungen weggelassen.

Lernerfolg

Für die Kombination standen somit 11 Läufe von PEPC auf Power6 mit 5 Millionen Partikeln zur Verfügung, die mit den IMB-*Allgather* Läufen verknüpft werden konnten. Das Ergebnis eines solchen Trainings ist in Abb. 6.23 dargestellt.

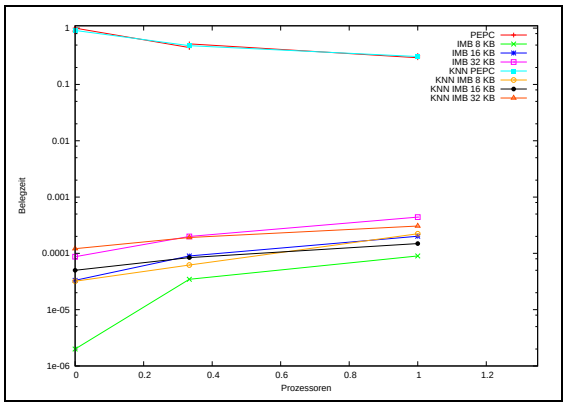


Abbildung 6.23: Lernerfolg der Kombination von PEPC und IMB in einem KNN. Sowohl die PEPC- wie auch die einzelnen IMB-Kurven der unterschiedlichen Nachrichtengrößen werden vom KNN erkannt.

Im oberen Bereich ist die PEPC-Kurve für 5-Millionen Partikel auf Power6 zu sehen, im unteren die drei IMB-Kurven für *Allgather*. Es handelt sich um ein 6:10:1 Netz. Die Anzahl der Neuronen in der verdeckten Schicht wurde im Vergleich zu den vorhergehenden Tests mit PEPC erhöht, um auch die IMB-Werte besser anzunähern. Das gelingt mit dieser Netzkonfiguration, obwohl die große Spannbreite der Daten ein Training erschwert. Der Ausreisser der IMB Kurve mit 8MB wird von Netz sogar als solcher erkannt, wobei auch die Kurven der anderen Nachrichtengrößen für den ersten Punkt etwas nach unten abgelenkt werden.

Der Lernerfolg besteht somit darin, dass auch diese Kurven, die unabhängig voneinander sind, vom Netz zugeordnet werden können und das KNN somit auch in der Lage ist, eine Kombination von Benchmarks zu erlernen.

Prognosetest

Interessant wird es jetzt sein, ob die zusätzliche Information des IMB-Benchmarks eine Hilfe bei der Prognose von PEPC-Werten bietet.

Da, wie beschrieben, bzgl. der Kommunikation instrumentierte Messungen nur für Power6 vorliegen, werden diese wie oben als Trainingswerte genommen. Um eine Vorhersage der Power4-Werte vornehmen zu können, wird, wie unter Abschnitt 6.4.3 auf Seite 65 erläutert, noch mindestens ein Wert der Power4 benötigt, um einen Referenzwert für die Höhe der Kurve zu haben. Das Verhalten soll aus den Power6-Werten gelernt werden.
Im Vergleich zu den vorhergehenden Tests kann nun aber noch die Information des IMB-Benchmarks hinzugenommen werden, so dass über das Verhalten von IMB auf der Power4 und dem Wissen über den Kommunikationsanteil von PEPC eine genauere Prognose möglich werden sollte. Daher werden die Trainingsdaten durch die IMB-Werte für Power6 und nun zusätzlich für Power4 ergänzt. Die IMB-Power4 Werte wurden für den Test als dreifach langsamer als Power6 angenähert.
Zusammengefasst wird somit das Training mit folgenden Werten vorgeonmmen:

- Instrumentierte Power6-Läufe für 5 Millionen Partikel
- Ein Power4 Wert mit 5 Millionen Partikel für 256 CPUs
- IMB-Werte von *Allgather* für Power4 und Power6

Allgather wurde gewählt, da in PEPC zu einem größeren Teil ebenfalls auf diese Art kommuniziert wird.
Das Ergebnis dieses Tests ist in Abb. 6.24 zu sehen. Bei dieser Darstellung wurde auf die logarithmische Skala verzichtet, um den Verlauf der PEPC-Kurven besser zu erkennen, zwischen denen die Prognose vorgenommen wird. Es ist zu sehen, dass der vorgegebene

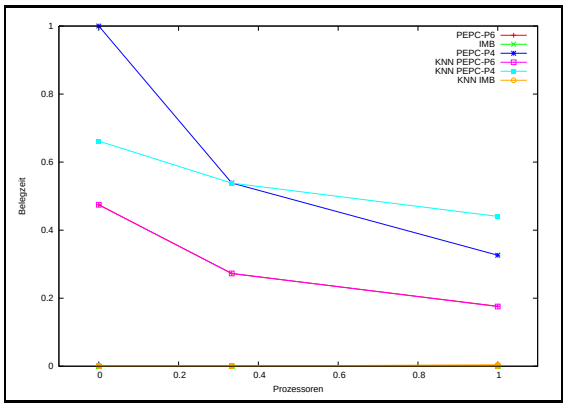


Abbildung 6.24: Prognose für Power4 aus der Kombination von PEPC und IMB: Der Kurvenverlauf orientiert sich zwar am Referenzwert der Power4 und liegt oberhalb der Power6-Kurve, nimmt aber nicht den steilen Verlauf der originalen Power4-Kurve.

Wert der Power4 bei 256 Prozessoren gelernt wurde und die zu prognostizierenden Werte

analog zur Kurve der Power6 angenähert wurden, nur etwa entsprechend dem Faktor langsamer, den auch der Lauf mit 256 Prozessoren langsamer war. Verglichen mit den Originalwerten müsste die Kurve aber steiler verlaufen als die Power6-Kurve.
Das ist genau das Lernverhalten, das auch ohne die Information von IMB zu erwarten wäre. Die zusätzlichen Trainingswerte scheinen also keine Verbesserung zu bringen, das Netz setzt sie nicht zueinander in Beziehung. Woran liegt das und kann dieses Verhalten verbessert werden?

Dazu soll einmal ein Blick in die Daten der beiden Benchmarks geworfen werden:

```
IMB
#CPUs       Bytes    #Msg    MFlop Platform    Belegzeit
128.000    8192.0    1000      0.0      4.7     0.00248480
128.000   16384.0    1000      0.0      4.7     0.00607985
128.000   32768.0    1000      0.0      4.7     0.01193797
256.000    8192.0    1000      0.0      4.7     0.00623330
256.000   16384.0    1000      0.0      4.7     0.01222515
256.000   32768.0    1000      0.0      4.7     0.02409390
512.000    8192.0    1000      0.0      4.7     0.01221858
512.000   16384.0    1000      0.0      4.7     0.02429991
512.000   32768.0    1000      0.0      4.7     0.05021884
128.000    8192.0    1000      0.0      1.7     0.00648480
...

PEPC
#CPUs       Bytes    #Msg    MFlop Platform    Belegzeit
128.00        4.0   14546      513      4.7     93.29
256.00        4.0    7981      407      4.7     53.56
512.00        4.0    4118      293      4.7     34.62
256.00        4.0    7981      407      1.7     105.91
```

Es ist zu erkennen, dass die beiden Benchmarks aus Sicht des KNN nur die Anzahl der CPUs und die Plattform gemeinsam haben, das Kommunikationsverhalten unterscheidet sich sehr, sowohl in der Größe wie auch in der Anzahl der verschickten Nachrichten. Ebenso verhält es sich mit der MFlop-Rate, die bei einem zu 100% aus Kommunikation bestehenden Benchmark wie IMB 0 beträgt.
Das Netz sollte anhand des Verhaltens von IMB natürlich lernen, dass ein Benchmark mehr Zeit benötigt, je größer die zu versendenen Nachrichten sind. Die Nachrichtengröße ist bei PEPC aber konstant und zudem wesentlich kleiner als bei den IMB-Messungen. Die Schlussfolgerung wäre an der Stelle also eher eine Vorhersage von wesentlich schnelleren Läufen für Power4, was aber von den anderen Trainingswerten verhindert wird.

In der Realität sollten an dieser Stelle für eine bessere Prognose natürlich keine weiteren Läufe des zu prognostizierenden Benchmarks gemacht werden, um etwa für größere

Byte-Werte mehr Trainingsdaten zu erhalten. In dem Fall würde keine Prognose mehr benötigt werden. Weitere Läufe eines Low-Level-Benchmarks wie IMB liegen aber häufig vor oder können durchgeführt werden. Es würden dann IMB-Messungen benötigt, die mehr Information für PEPC beinhalten, wenn sie nicht schon in einer entsprechend großen Benchmark-Datenbank vorliegen.

Zudem wurde bei den Untersuchungen festgestellt, dass PEPC leider zur Zeit noch Probleme mit der Lastbalance hat und viel Zeit in Barriers verbracht werden muss. Die Kommunikation mit *Allgather* macht zwar einen größeren Prozentsatz der tatsächlichen Kommunikation aus, der weitaus größte Teil sind aber die angesprochenen Barriers, die hier nicht einfließen können.

Weiterhin muss beachtet werden, dass die Kommunikation nur ein weiterer Baustein für eine Prognose einer realen Anwendung wie PEPC ist. Wie in Kapitel 3 zur Problemanalyse ausgeführt, muss neben der Kommunikation für einen Anwendungs-Benchmark z.B. auch noch das Speicherverhalten oder die Zugriffe auf Dateien berücksichtigt werden.
Aus der aktuell vorhandenen Datenlage heraus wurde die Entscheidung getroffen, die Testreihe mit PEPC und IMB durchzuführen, in Zukunft wird auch eine größere Datenbasis z.B. bzgl. Low-Level-Benchmarks zu I/O zur Verfügung stehen, die eine detailliertere Untersuchung erlauben werden.

Abhängige Daten

Eine weitere wichtige Schlussfolgerung bzgl. der Nachrichtengröße ist aus diesem Prognosetests zu ziehen und wurde schon angedeutet: Die vorliegenden Daten des Eingabevektors bedingen einander. Würde PEPC mit einer größeren Nachrichtenanzahl getestet werden, müssten gleichzeitig die Anzahl an MFlops sinken oder aber natürlich die Laufzeit ansteigen, wofür das Netz entsprechende Trainingsdaten benötigt. Es könnte zudem ein zusätzliches Neuron eingeführt werden, dass den Anteil an Kommunikation ausdrückt.
Es wurden entsprechende Tests vorgenommen, die dieses Verhalten des Netzes zeigen. Es nähert sich, wie aus der Logik des Netzes zu erwarten, immer den Werten an, denen sein Inputvektor zu entsprechen scheint, bei grösseren Nachrichten in diesem Fall also den IMB-Werten, wo entsprechend z.B. über die MFlop-Rate gegengesteuert werden muss.
Die Herausforderung besteht daher darin, einen ausreichenden Trainingssatz bereitzustellen, durch den das Netz die verschiedenen Fälle erlernen kann.

6.4.10 Bewertung der Testreihe mit PEPC und IMB

Es wurde gezeigt, dass ein Erlernen von verschiedenen Benchmarks anhand Ihrer Kennwerte grundsätzlich möglich ist. Das ist eine wesentliche Grundlage für ein Konzept, bei dem ja nicht die Benchmarks durch eigene Neurone gekennzeichnet werden sollen, wie es noch bei dem ersten Kombinationstest mit verschiedenen IMB-Kernen der Fall war.
Es wurde aber auch deutlich, dass die Komplexität eines solchen Netzes sehr schnell

ansteigen kann.
Als wichtigste Erkenntnis ist zu nennen, dass eine hohe Anforderung an die Datenbasis gestellt werden muss, die alle Zusammenhänge, die das Netz für eine Prognose benötigt, in den Trainingsdaten enthalten muss.

6.5 Testreihen – Kurzzusammenfassung

Es wurde eine ausführliche Testreihe unter Nutzung von Künstlichen Neuronalen Netze für das Benchmarking durchgeführt. Durch die detaillierten Tests mit PEPC konnte das Training eines feed-forward Netzes mit überwachtem Lernen vorgestellt werden. Bei den Randbedingungen, die dabei beachtet werden müssen, handelt es sich im Wesentlichen um das Erhalten der Generalisierung. Die Topologie eines Netzes kann über die Anzahl an Neuronen und Schichten verändert werden. Je mehr Neurone sich in den verdeckten Schichten befinden, desto genauer können die Trainingsdaten erlernt werden. Die Gefahr ist aber, dass das Netz die gelernte Information nicht mehr verallgemeinern kann. Um dieses Verhalten zu verhindern, wurde bei allen Tests ein Referenzdatensatz mitgeführt, auch die gelungenen Prognosen bestätigen die vorhandene Fähigkeit zur Verallgemeinerung. Zudem kommt die Möglichkeit, verschiedenen Trainingsalgorithmen zu verwenden. Backpropagation hat sich bei diesen Tests besser verhalten als Resilient Propagation.
Diese Erfahrungen konnten schon in Tests mit IMB und die abschließenden Kombinationstests mit PEPC und IMB einfließen.

Somit ergibt sich auf der einen Seite ein positives Bild bzgl. der Entscheidung für die Prognose mit Neuronalen Netzen, auf der anderen Seite müssen die Schwierigkeiten, die insbesondere bei der Kombination der zwei Benchmarks PEPC und IMB zu Tage gekommen sind, berücksichtig werden: es hat sich gezeigt, dass zum einen die Streuung der Daten das Lernen beeinflusst. Es wurden mehr verdeckte Neurone benötigt, um die im Vergleich zu den PEPC-Läufen sehr kurzen IMB-Läufe, die noch dazu stärker aufgefächert waren, ebenfalls gut anzunähern. Zum anderen muss die Lernbasis möglichst vollständig das Verhalten der Benchmarks widerspiegeln, um nicht zu falschen Ergebnissen zu gelangen. Zum Beispiel waren in den Trainingsdaten des Kombinationstests nur PEPC-Messungen mit sehr kleinen Nachrichtengrößen und IMB-Messungen mit größeren Nachrichten. Wäre auf das mit diesen Daten trainierte Netz eine Abfrage hinsichtlich PEPC-Läufen mit größeren Nachrichten gemacht worden, hätte sich die Prognose an den kurzen IMB-Laufzeiten orientiert, obwohl die Laufzeiten eigentlich länger werden müssten. Das Netz muss also erkennen, dass die MFlop-Rate ebenfalls einen entscheidenen Beitrag liefern muss. Dafür benötigt es einen entsprechenden Trainingsdatensatz, der diese Fälle abdeckt. Um den Einfluss eines Eingabewerts eines Neurons lernen zu können, müssen daher mehrere Vergleichsmessungen vorliegen. Zudem müssen alle Komponenten, die Einfluss auf das Ergebnis haben, abgebildet werden. Ansonsten kann das Netz nicht erkennen, wodurch im Beispiel der Laufzeitprognose eine Änderung einer Belegzeit resultiert.

Weiterhin ist zu sagen, dass in der vorgenommenen Testreihe die Performance des Trainings sehr gut war, aber bemerkt werden konnte, dass ein größerer Trainingsdatensatz schnell zu längeren Laufzeiten führte.
In allen Tests wurden für die Eingabevektoren feste Werte benutzt. Dieses Vorgehen führte zu guten Ergebnissen, die in Kapitel 4 vorgestellte Möglichkeit der Fuzzymethoden wird für das Konzept somit vorraussichtlich nicht verfolgt werden müssen.

Diese aus der Testreihe resultierenden Überlegungen werden im folgenden Kapitel in den Entwurf des Benchmark-Prognose-Systems einfließen.

7 Konzeption

In diesem Kapitel soll basierend auf der Testreihe im vorhergehenden Kapitel das Konzept für das Softwaresystem entwickelt werden, mit dessen Hilfe Prognosen von Benchmarking-Ergebnissen möglich gemacht werden sollen.
Dazu wird zunächst noch einmal die Prognose eines einzelnen Benchmarks behandelt, um dann zu einem generelleren Konzept überzugehen.

7.1 Einfache Prognose eines Benchmarks

Zur Prognose eines einzelnen Benchmarks mussten die maßgeblichen Faktoren bestimmt werden, die Einfluss auf das Verhalten des Programms nehmen. Bei PEPC war das die Anzahl an Partikeln, die bestimmt, wie groß die Aufgabe ist, die das Programm zu lösen hat. Hinzu kommen dann die Laufzeitparameter, die Einfluss auf die Programmdauer haben und natürlich die Eigenschaften des Rechners, auf dem das Programm läuft. Im Beispiel mit PEPC war der Rechner durch den Prozessortakt gekennzeichnet.
Es war zu sehen, dass mit diesen relativ wenigen Eingaben schon recht gute Prognosen möglich waren. Die wesentlichen Schritte sind dabei die Folgenden:

- Die Festlegung des Ausgabeneurons, d.h. welche Abfrage soll das Netz beantworten können. In den Testfällen wurde dazu jeweils die Belegzeit des Benchmarks genutzt, um Skalierungskurven vorherzubestimmen.
- Die Festlegung der Eingabeneurone, also wie oben beschrieben, die bestimmenden Faktoren des Benchmarks.
- Eine Betrachtung der Datenlage, d.h. ist eine ausreichende Menge an Trainingsdaten vorhanden, zusätzlich ist eine Einteilung in Trainings- und Referenzdatensatz sinnvoll, um die Generalisierung des Netzes prüfen zu können.
- Die Bestimmung der Netzarchitektur und des Lernalgorithmus, in den Tests erwies sich hier ein feed-forward Netzwerk mit einer verdeckten Schicht als ausreichend.
- Darauf folgt das Training des Netzes, nach dem Anfragen an das Netz möglich werden.

Es ist allerdings zu beachten, welche Abfragen an ein solches Netz gestellt werden können. Gute Ergebnisse wurden erzielt nach einem Training mit Werten über zwei Plattformen, wobei die Prognose dann hinsichtlich einer Anzahl Partikel, also der Problemgröße des Benchmarks, gemacht wurde. Das ist eine durchaus sehr wertvolle Information für den Wissenschaftler, der ein Programm wie PEPC nutzt. Er kann so abschätzen, wieweit er

auf einem Rechner eine Simulation sinnvoll durchführen kann.
Es handelt sich dabei aber noch nicht um die Prognose des Verhaltens eines Programms auf einer neuen Plattform, die als besonders entscheidend angesehen wurde z.B. hinsichtlich der Anschaffung eines neuen Rechnersystems. Dazu würde zumindest ein Referenzwert auf der zu prognostizierenden Plattform benötigt, um die Höhe des Kurve festzulegen. Dann würde aber davon ausgegangen werden, dass sich die neue Plattform ansonsten wie die bekannte Plattform verhält. Es wird also noch mehr Information benötigt, z.B. über das Kommunikationsverhalten, die in das Netz integriert werden muss. Dazu können natürlich weitere Neurone bereitgestellt werden, doch das Netz muss auch das entsprechende Verhalten des Benchmarks zu jedem Neuron erlernen. Das heisst man würde entsprechende Läufe des Anwendungsbenchmarks auf dem System benötigen, die auf einem neuen System aber gerade in der Regel nicht möglich sind. Hinzu kommt, dass für eine wirkliche Prognose der Wunsch wäre, gar keine Läufe auf dem neuen System machen zu müssen, d.h. auch der Referenzwert, der in den Tests genutzt wurde um die Höhe einer Kurve festzulegen, soll nicht benötigt werden.
Das führt zu einem komplexeren Ansatz, der in dem folgenden Abschnitt ausgeführt wird.

7.2 Integration von Low-Level-Benchmarks

In der Testreihe in Kapitel 6 wurde bereits die Möglichkeit verfolgt, die Prognose eines Anwendungsbenchmarks durch die Informationen eines Low-Level-Benchmarks zu verbessern. Die beiden Benchmarks konnten auch gemeinsam gelernt werden, eine Verbesserung der Prognose wurde durch die Kombination von PEPC und IMB-*Allgather* jedoch noch nicht erreicht. Einer der beschriebenen Gründe dafür war, dass *Allgather* nur einen Teil der Kommunikation ausmacht und darüberhinaus, dass die Kommunikation als solche ja nur einen Teil der Belegzeit von PEPC verursacht. Hinzu kommen noch weitere Bereiche des Programms: Zum einen natürlich die Problemgröße des Programms, die beschrieben werden kann durch die MFlop-Rate, zum anderen weitere Basisfunktionen wie Speicherzugriffe oder I/O. Eine entsprechende Auflistung von Programmeigenschaften wurde schon im Kapitel 3 zur Problemanalyse dargelegt. Alle diese Programmteile verursachen einen Anteil an der Gesamtbelegzeit des Programms. Letzlich setzt sich ein Anwendungsprogramm also aus grundlegenden Funktionen zusammen, für deren Tests Low-Level-Benchmarks existieren.
Setzt sich also die Belegzeit eines Programms aus Low-Level Funktionen zusammen und können diese Funktionen auf einem Rechner über entsprechende Low-Level-Benchmarks getestet werden, so können diese Benchmarks die Anwendung ausreichend beschreiben und es ist kein Referenzwert auf der neuen Platform für den zu prognostizierenden Benchmark mehr erforderlich. Durch die Low-Level-Benchmarks werden also im Grunde die Rechnereigenschaften aus der Klasse *Rechner* abgebildet.

Diese Methode setzt aber zunächst voraus, dass Messwerte von Low-Level-Benchmarks auf einer neuen Plattform vorhanden sind. Das kann tatsächlich der Fall sein. Wenn die

Maschine beim Hersteller schon vorhanden ist, kann dieser die entsprechenden Messungen bereitstellen. Aber auch wenn dieser Fall nicht vorliegt, so wird der Hersteller doch Aussagen über das neue System machen, wie z.B. die Geschwindigkeit des Netzwerkes. Dann kann das Verhalten von in diesem Beispiel IMB wiederum von einem KNN prognostiziert werden. Das KNN wäre dann mit den Werten von Power4 und Power5 trainiert und könnte anhand der Herstellerangaben über die Power6-Konfiguration einen Verlauf der IMB-Kurven vorhersagen. Es handelt sich dann also um ein der Prognose für den Anwendungsbenchmark auf der neuen Plattform vorgeschaltetes KNN.
Ein solches vorgeschaltetes Netz kann auch für andere Werte des Eingabevektors des Hauptnetzes verwendet werden, die nicht direkt bekannt sind.

Es ist somit möglich, für die Konzeption eine vollständige Aufteilung des Benchmarks in die Elemente, aus denen sich die Gesamtlaufzeit zusammensetzt, vorzusehen. Die Low-Level-Benchmarks geben dann die Kenndaten dieser Elemente auf einem neuen System wieder und ergänzen den Trainingssatz des Anwendungsbenchmarks. Damit wird dem Prinzip der Aufgabenanalyse in Kapitel 3 gefolgt.

Dieser Ansatz orientiert sich an den Methoden der Performance Prediction (s. Kapitel 3), allerdings wird aus der Instrumentierung heraus kein direktes Rechenmodell für die Vorhersage entworfen, sondern das Neuronale Netz übernimmt diese Aufgabe. Abb. 7.1 auf der nächsten Seite soll diese Konzeption verdeutlichen.

Als problematisch könnte angesehen werden, dass bei PEPC beobachtet wurde, dass durch die Schwierigkeiten mit der Lastbalance Effekte auftreten, die nicht auf solche grundlegenden Funktionen abgebildet werden können. Das ist aber ein Fall, der jede Art der Prognose extrem erschwert und somit nicht auf diese Konzeption beschränkt ist. Hier müsste zunächst das Anwendungsprogramm entsprechend optimiert werden. Erst danach wäre es wirklich sinnvoll als Benchmark einzusetzen. Es geht in dieser Konzeption nicht um die Prognose allgemeiner Programme, sondern um Benchmark-Programme, die also Vergleiche zwischen Rechnern zulassen sollen. Effekte wie die beschriebene Inbalance wären dabei grundsätzlich ein Problem und ein gewisser Grad an Optimierung sollte daher vorausgesetzt werden.

Abschließend ist also festzustellen, dass für das Gelingen einer Prognose somit eine breite Datenbasis an Low-Level-Benchmarks notwendig ist, die das neue System beschreiben. Mit dieser Basis und den Werten eines Anwendungsbenchmarks auf bekannten Platformen ist eine gute Prognose des Programmverlaufs auf einer unbekannten Plattform zu erwarten.

7.3 Softwaresystem zur Prognose des Benchmarking

Durch die bisherigen Ausführungen in diesem Kapitel wurde die Datenbasis festgelegt, die für eine Prognose eines Benchmarks durch ein Künstliches Neuronales Netz bzw. eine

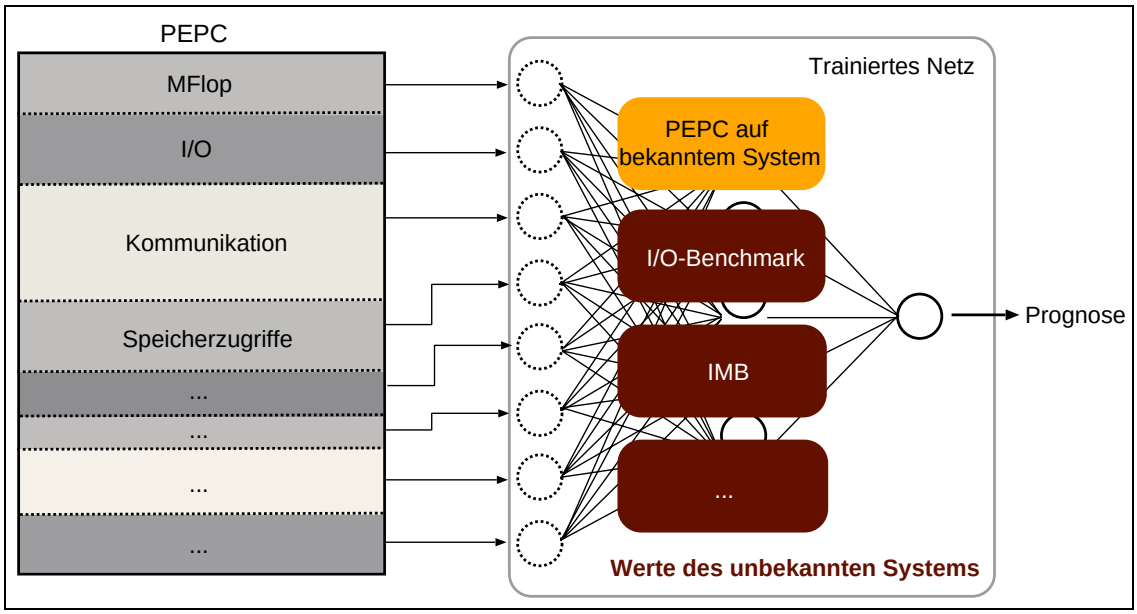


Abbildung 7.1: Konzeption des Benchmarking-Prognose-System: Die Prognose basiert auf den bekannten Läufen des zu prognostizierenden Benchmarks und den bekannten oder zuvor bestimmten Werten von Low-Level-Benchmarks auf dem neuen System.

Kombination von Neuronalen Netzen zur Verfügung stehen soll.
Nun soll abschließend das Softwaresystem, durch das die Prognose realisiert werden soll, konzeptionell dargelegt werden.

Es wird sich aus den folgenden Komponenten zusammensetzen:

Datenbereitstellung

Für die Prognose werden zunächst die Daten benötigt, mit denen das KNN trainiert werden kann. Darunter fallen nicht nur die Laufzeiten eines Benchmarks, sondern auch die Konfiguration, mit der diese Laufzeit erreicht wurde. Zur Konfiguration gehören die Elemente, die in der Problemanalyse in Kapitel 3 unter die Klasse *Programm* fielen, also z.B. die Laufzeitkonfiguration. Diese Informationen müssen in konsistenter Form vorliegen.
Das Konzept sieht daher vor, die Messreihen der vorliegenden Benchmarks in einer Datenbank zu speichern. Alle Attribute zu einem Lauf können so festgehalten werden. Ein weiterer Vorteil ist, dass auf die Daten so auch externen Partnern Zugriff gewährleistet werden kann. Im konkreten Fall der Anwendung im JSC bedeuted das, dass diese Datenbank auch von Teilnehmern der Projekte DEISA und PRACE gefüllt werden. Auf diese Weise kann eine große Datenbasis für das Training entstehen.

Preprocessing

Nachdem die Daten bereitgestellt worden sind, wird ein Programm benötigt, dass die Werte aus der Datenbank liest und für die Verarbeitung im KNN vorbereitet. Das bedeutet z.B., dass die Werte, wie es schon bei der Testreihe gemacht wurde, auf den Wertebereich [0,1] skaliert werden. Zudem müssen auch Anfragen an das Netz auf den entsprechenden Wertebereich übertragen werden.

Training

Der Kern der Prognose wird das Training des Neuronalen Netzes sein. Dafür gibt es die Möglichkeit, die entsprechende Architektur in einer Eigenentwicklung zu programmieren oder einen bestehenden Simulator zu nutzen. In der Testreihe hat sich die FANN-Bibliothek bewährt, alle Tests konnten gut und auch zügig mit der Bibliothek durchgeführt werden. Das führt zu der Entscheidung, dass diese Bibliothek auch als Komponente für das Softwaresystem genutzt werden kann.

Benutzerschnittstelle und Prognose

Schließlich soll die Möglichkeit gegeben werden, auf das System über eine graphische Benutzerschnittstelle zuzugreifen. Über diese Schnittstelle kann auf der einen Seite das Training des Netzes gesteuert werden. Das bedeutet, das auch Parameter wie die Netztopologie und die Trainingslänge über diese Oberfläche konfiguriert werden können. Auf der anderen Seite sollen über die graphische Oberfläche Anfragen an ein trainiertes Netz gestellt werden können. Auf diese Weise kann eine Prognose abgerufen werden.

Integrator

In Kapitel 2 *Benchmarking* wurde bereits die Jülicher Benchmarking-Umgebung JuBE beschrieben. Dort wurde erläutert, dass für diese Umgebung bereits weitere Planungen existieren. So ist eine graphische Benutzeroberfläche in Arbeit. Zudem soll eine Datenbank zur Speicherung der Benchmarkläufe, wie auch für die Prognose nötig, entwickelt werden. Auf diese Datenbank basierend ist zudem ein Modul zur Analyse von Benchmarkingläufen vorgesehen.
Es ist daher eine naheliegende Wahl, auch für die Benchmarkingprognose als Integrator JuBE vorzusehen. Bestärkt wird diese Entscheidung dadurch, dass JuBE inzwischen im Benchmarkingbereich bereits weite Verbreitung gefunden hat, es wird z.B. in den angesprochenen Projekten DEISA und PRACE verwendet. Die graphische Oberfläche kann ergänzt werden hinsichtlich der benötigten Bereiche für die Prognose. Die Datenbank kann sowohl von den Modulen für Analyse und Prognose genutzt werden. Die FANN-Bibliothek wird sich gut einbinden lassen, die Funktionen können über C-Programme angesprochen werden oder es wird direkt die Perl-Schnittstelle, die FANN bietet, von JuBE aus genutzt.
Auf diese Weise entsteht ein Framework, mit dem Benchmarks von der Konfiguration des Laufs auf einem Rechner, dem Lauf selber, der Speicherung der Benchmark-Daten und

schließlich sowohl der Analyse bestehender Rechner wie auch einer Prognose zukünftiger Rechner verwaltet werden können.

In Abb. 7.2 ist dieser konzeptionelle Aufbau des Systems zur Benchmarking-Prognose dargestellt.

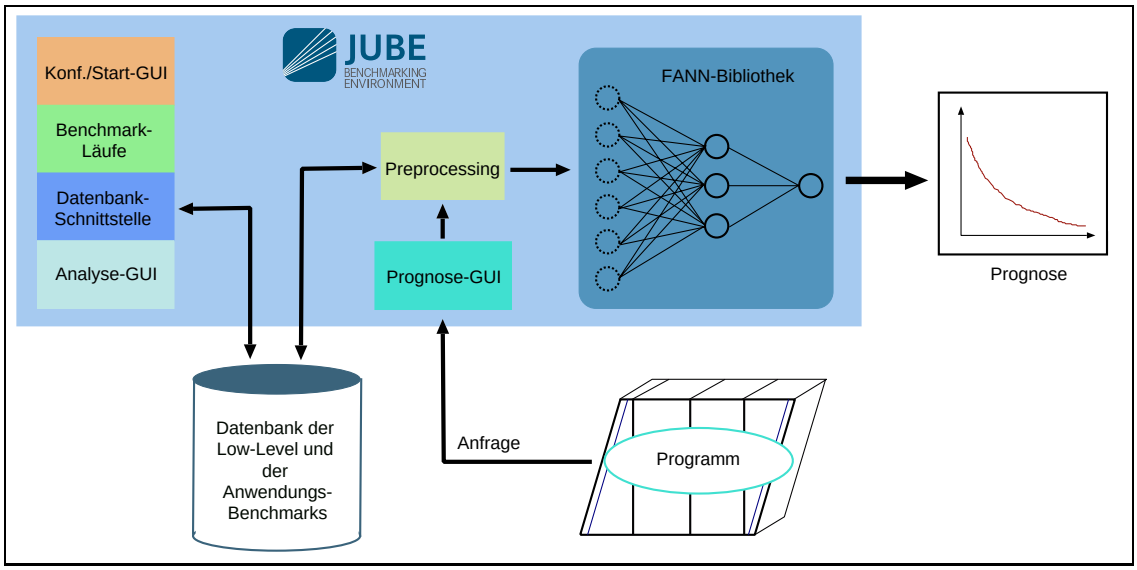


Abbildung 7.2: Software-Module des Benchmarking-Prognose-Systems: Das System besteht aus den Elementen der Datenbereitstellung durch Benchmarkläufe, die in einer Datenbank gespeichert werden, dem Preprocessing der Benchmark-Daten, um sie für das KNN aufzubereiten, dem FANN-Simulator für die Arbeit mit den KNN, und schließlich der Schnittstelle für die Prognose. Als Integrator für diese Module dient die Datenbank-Umgebung JuBE.

7.4 Konzeption – Kurzzusammenfassung

In diesem Kapitel wurde die Konzeption für das Softwaresystem zur Prognose von Benchmarking-Ergebnissen aufgebaut.
Die Konzeption beinhaltet, auf Basis einer Kombination von Anwendungs- und Low-Level-Benchmarks die Prognose durchzuführen. Dabei wird der Anwendungsbenchmark auf Basiskomponenten zurückgeführt, für welche die Low-Level-Benchmarks die Kenndaten liefern.
Die Software-Umsetzung besteht aus einem graphischen Interface, einer Datenbank-Anwendung und der Bibliothek FANN für Künstliche Neuronale Netze. Es wird vorgesehen, diese Komponenten in die Benchmarking-Umgebung JuBE zu integrieren.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Konzept für ein Benchmarking-Prognose-System entwickelt. Dazu wurden zunächst die Zusammenhänge erläutert, die im Benchmarking eine Rolle spielen. Es wurde festgestellt, dass für eine Prognose das Zusammenspiel von den Komponenten des Anwendungs-Benchmarks und des Rechners von entscheidender Bedeutung sind. Statt den Ansätzen der Performance Prediction durch mathematische Modellierung oder statistische Verfahren zu folgen, wurde die Entscheidung gefällt, Methoden der Künstlichen Intelligenz (KI) zu betrachten, um herauszufinden, ob auf diesem Weg eine Benchmarking-Prognose möglich gemacht werden kann.

Daher wurden Verfahren der KI theoretisch untersucht und in der Arbeit beschrieben, konkret wurden Expertensysteme mit Ontologien und Künstliche Neuronale Netze verglichen. Die Entscheidung fiel zu Gunsten der Künstlichen Neuronalen Netze, da deren prinzipieller Ansatz als vorteilhaft angesehen wurde. Er besteht darin, dass das Wissen für eine Prognose nur aus einer zu Grunde liegenden Datenmenge kommt und kein zusätzliches Wissen über Regeln abgebildet werden muss. Es war also entscheidend herauszufinden, ob die reine Informationsgewinnung aus den Benchmark-Daten für eine Prognose ausreicht. Daher folgte eine ausführliche Testreihe zu den Möglichkeiten von Neuronalen Netzen in Bezug auf die Benchmarking-Prognose. Dabei wurde zunächst anhand eines Anwendungs-Benchmarks das Training und die Prognosefähigkeit eines Netzes ausgetestet. Anschließend wurde diese Benchmark-Prognose durch Informationen eines Low-Level-Benchmarks ergänzt. Diese Tests wurden durchgeführt mit mehrschichtigen Neuronalen Netzen, wobei verschiedene Netztopologien und Trainingsalgorithmen verwendet wurden.
Bei der Testreihe hat sich gezeigt, dass eine gute Prognose auf Basis der Daten einer Reihe von Benchmarkläufen möglich ist, sofern das Verhalten des Benchmarks ausreichend in den Daten abgebildet ist. Das bedeutet, dass z.B. für die Prognose einer neuen Problemgröße zumindest zwei Problemgrößen bekannt sein müssen, damit durch das Netz ein Verhältnis erkannt werden kann.

Diese Testreihe wurde zur Grundlage der Konzeption. Das entwickelte Konzept sieht eine Kombination von Anwendungs- und Low Level-Benchmarks vor. Dabei wird davon ausgegangen, dass eine Anwendung zurückgeführt werden kann auf ihre Basiskomponenten, wie z.B. die Kommunikation. Durch die Kombination mit Low-Level-Benchmarks soll die Möglichkeit geschaffen werden, auch Prognosen für unbekannte Rechner-Plattformen vornehmen zu können. Für diese Plattformen liegen ja keine Referenzrechnungen des Anwendungsbenchmarks vor. Die Low-Level-Benchmarks können diese Informationslücke schliessen, indem sie für die einzelnen Elemente der Anwendung das Verhalten des un-

bekannten Rechners beschreiben. Die Werte der Low-Level-Benchmarks werden dabei vom Hersteller geliefert und können direkt verwendet oder anhand der Kennwerte des Rechners durch ein vorgeschaltetes Künstliches Neuronales Netz bestimmt werden.
Durch diese Kombination kann eine gute Prognose erwartet werden, da sowohl der zu prognostizierende Benchmark als auch das Rechnersystem, für das die Prognose erstellt werden soll, vollständig beschrieben sind.

Das Konzept sieht weiterhin vor, die Datenbasis aus verschiedenen Typen von Benchmarks in ein Software-System einzubinden, das aus den Komponenten einer Datenbank, der FANN-Library für Neuronale Netze und der Benchmarking-Umgebung JuBE als Integrator mit graphischer Schnittstelle bestehen soll.

In der Arbeit wurde beschrieben, dass bereits eine Reihe an Benchmark-Messungen vorliegen. Ein Teil davon konnte für die Testreihe genutzt werden, es wurde aber auch deutlich, dass insbesondere die Datenbasis für die Low-Level-Benchmarks noch weiter ergänzt werden muss. Dies geschieht bereits im europäischen Projekt PRACE, für die dabei wachsende Zahl an Benchmarking-Ergebnissen ist daher das Modul der Datenbank aufzubauen, um so eine konsistente Speicherung zu erlauben, auf die dann für die Prognosen zugegriffen werden kann.

Weiterhin wird eine Untersuchung sinnvoll sein, ob zur weiteren Verbesserung der Prognose eine statistische Vorbehandlung der Datenbasis und damit der Trainingswerte für ein Netz hilfreich sein kann. Auf diese Weise könnte doppelte oder widersprüchliche Information herausgefiltert und so die Prognose verbessert und auch beschleunigt werden.

In der Arbeit wurde ausschließlich mit mehrschichtigen, feed-forward Netzen gearbeitet, da diese Netze durch die Literaturrecherche als geeignet erschienen und sich ihr Einsatz in den Tests auch bestätigt hat. Es wäre aber dennoch durchaus sinnvoll noch weitere Netzwerk-Architekturen zu testen, z.B. noch flexiblere Netzwerke, die die Bestimmung der Anzahl von Neuronen und Verbindungen erleichtern bzw. auch je nach Problem anpassen, wie z.B. die Cascade Netzwerke oder die ART-Architektur.

Insgesamt hat sich gezeigt, dass die Künstlichen Neuronalen Netze ein lohnenswerter Ansatz zur Benchmarking-Prognose sind. Die vorgeschlagene Konzeption wurde anhand von Testreihen entwickelt und verifiziert. Als nächste Schritte sind die Anwendung der Konzeption auf eine größere Datenbasis und die Einbeziehung einer breiten Anwendergruppe geplant.

Der Nachteil der Intelligenz besteht darin,
daß man ununterbrochen gezwungen ist,
dazuzulernen.

George Bernard Shaw

Literaturverzeichnis

[L1] Beyer, Performance Prediction Tools für massiv parallele Programme, TU Chemnitz, 1996

[L2] Bothe, H.-H., Neuro-Fuzzy-Methoden - Einführung in Theorie und Anwendungen, Springer, 1998

[L3] Flynn, M. J., Some computer organizations and their effectiveness, IEEE Trans. Comput., Vol. C-21, pp. 948, 1972

[L4] Girona, Sergi, Performance Prediction and Evaluation Tools, Universitat Politecnica de Catalunya, 2003

[L5] Hasenauer/Merkl, Forest Tree Morality Simulation in Uneven-Aged Stands Using Connectionist Networks, Proceedings of the International Conference on Engineering Applications of Neural Networks, 1997

[L6] Helbig, Künstliche Intelligenz und automatische Wissensverarbeitung, Verlag Technik, 1996

[L7] Kohonen, Teuvo, Self-Organizing Maps; Springer Series in Information Sciences, 1995, 1997, 2001

[L8] Lilja, Measuring Computer Performance, Cambridge University Press, 2000

[L9] Minsky/Papert, Perceptrons: An Introduction to Computational Geometry; MIT Press (Juni 1969)

[L10] Poddig/Sidorovitch, Künstliche Neuronale Netze: Überblick, Einsatzmöglichkeiten und Anwendungsprobleme, in Handbuch Data Mining im Marketing

[L11] Reif, Gerald, Moderne Aspekte der Wissensverarbeitung, Technische Universität Graz, 2000

[L12] Rich, Elaine, Artificial intelligence, McGraw-Hill Verlag New York, 1983.

[L13] Rosenblatt, Frank, Principles of neurodynamics: Perceptrons and the theory of brain mechanisms, Spartan Books, 1962

[L14] Rumelhart/Hinton/Williams, Learning representations by back-propagating errors., Nature (London) 323, S. 533-536

[L15] Schöneburg/Hansen/Gawelczyk, Neuronale Netzwerke, Markt&Technik, 1990

[L16] Stanley J./Bak E., Neuronale Netze - Computersimulation biologischer Intelligenz, Systhema Verlag, 1991

[L17] Sure, Y., Ontologien, Institut AIFB, Karlsruhe

[L18] Turing, A.M., Computing machinery and intelligence. Mind, 59, 433-460.

[L19] Simpson A./Bull M./Hill J./Weiland M., PRACE Deliverable D6.2.1 - Preliminary report on application requirements, 2008

Webseitenverzeichnis

[W1] http://www.top500.org

[W2] http://de.wikipedia.org/wiki/Benchmark (Version 30. September 2008)

[W3] http://icl.cs.utk.edu/hpcc/

[W4] http://www.spec.org/cpu2006/

[W5] http://www.fz-juelich.de/jsc/jube

[W6] http://www.deisa.eu

[W7] http://www.fz-juelich.de/jsc/grid/

[W8] http://www.deisa.eu/science/benchmarking

[W9] http://www.prace-project.eu

[W10] http://www.prace-project.eu/news/first-candidate-applications-for-prace-petaflop-s-systems-identified

[W11] http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

[W12] http://www.spec.org

[W13] http://www.spec.org/cpu2006/Docs/readme1st.html

[W14] http://www.nas.nasa.gov/Resources/Software/npb.html

[W15] http://www.fz-juelich.de/jsc/scalasca/

[W16] http://domino.research.ibm.com/comm/research_projects.nsf/pages/hpct.index.html

[W17] http://lexikon.meyers.de/meyers/Ontologie

[W18] http://de.wikipedia.org/wiki/Ontologie_(Informatik)(Version 6. Oktober 2008)

[W19] http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/57/

[W20] http://semantic-web-grundlagen.de/w/images/6/6c/7-OWL_-_Syntax_und_Intuition.pdf

[W21] www.bioinf.mdc-berlin.de/∼schober/OntologieVortrag.ppt

[W22] http://protege.stanford.edu/

[W23] www.aifb.uni-karlsruhe.de/Lehre/Sommer2005/SemTech/stuff/probabilistic.pdf

[W24] http://www.membrain-nn.de/

[W25] http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/

[W26] http://www.ra.cs.uni-tuebingen.de/software/snns/

[W27] http://www.mathworks.de/products/neuralnet/

[W28] http://octnnettb.sourceforge.net/

[W29] http://leenissen.dk/fann/

[W30] Kriesel, David, Neuronale Netze http://www.dkriesel.com

[W31] http://de.wikipedia.org/wiki/Resilient_Propagation (Version 18. September 2008)

[W32] http://www.fz-juelich.de/jsc/pepc

[W33] http://www.deisa.eu/science/benchmarking/codes/pepc

[W34] http://www.fz-juelich.de/jsc/jump

[W35] http://www.rzg.mpg.de/computing/hardware/Power6/the-ibm-power6-system

Danksagung

Mein Dank gilt Herrn Prof. Dr. Volker Sander für die Möglichkeit, diese Arbeit unter seiner Leitung anzufertigen und sein Interesse, sowie für die Anregung, in das spannende Gebiet der Künstlichen Intelligenz einzusteigen.

Bei Herrn Wolfgang Frings möchte ich mich herzlich bedanken für die engagierte Betreuung meiner Arbeit, die wertvollen Diskussionen und insbesondere für die Bereitschaft, jederzeit ansprechbar zu sein.

Mein besonderer Dank gilt dem Jülich Supercomputing Centre für jede Unterstützung während des Studiums, namentlich meinen Abteilungsleitern Dr. Norbert Attig und Wolfgang Gürich.

Weiterhin bedanke ich mich beim Benchmarking-Team des JSC für die Bereitstellung von Benchmark-Daten und insbesondere bei Herrn Dr. Florian Janetzko für das Korrekturlesen der Arbeit.

Ein herzliches Dankeschön an meine Familie für ihre Unterstützung während des gesamten Studiums.

Jül-4298
Mai 2009
ISSN 0944-2952