Inhaltsverzeichnis

Einführung

Diese Arbeit setzt sich aus sechs Kapiteln zusammen. In den ersten zwei Kapiteln werden Grids ausgiebig besprochen und vertieft, während die letzten vier Kapitel die (parallele) Programmentwicklung für Grids anhand der Fallstudie in einer MPI-Umgebung des Globus Toolkits behandeln.

Im ersten Kapitel wird eine historische Einführung in das Thema "Grid Computing" gegeben und einige daraus resultierende Definitionen des Begriffes "Grid" betrachtet. Weiter werden dann Grids nach ihren primären Anwendungsgebieten in drei Kategorien eingeteilt: Computational Grids, Data Grids und Access Grids. Anschließend werden einige ehemalige und gegenwärtige Grid-Projekte vorgestellt sowie andere verteilte Systeme mit Grids verglichen. Den Abschluss dieses Kapitels bilden die Besprechung mehrerer kommerzieller Grid-Lösungen sowie zwei Beispiele für Forschungsprojekte zu Grids.
Das zweite Kapitel beschäftigt sich mit den Aufgaben und der Umsetzung der Grid-Middleware, welche den Kern des Grids formt. In diesem Zusammenhang werden fünf Middleware-Projekte vorgestellt und dabei im Rahmen der Fallstudie genauer auf das "Globus Toolkit" eingegangen, da dieses die Grundlage für alle weiteren Kapitel bildet.
Im dritten Kapitel wird als Beispiel für parallele Programmierung mit Grids die MPI-Schnittstelle - bzw. deren Implementierung MPICH-G2 für das Globus Toolkit - ausführlich vorgestellt.
Dies führt im vierten Kapitel zu einer umfassenden Beispielinstallation des Globus Toolkits 2.4 mit MPICH-G2 unter Linux, wobei durch Testläufe von Beispielprogrammen ein kurzer Eindruck vermittelt werden soll, wie diese Grid-Umgebung zu nutzen ist. Anschließend werden die Grundlagen des LDAP-Verzeichnisdienstes erklärt, da im Globus Toolkit mittels dieser Struktur die Informationen über Ressourcen im Grid zugänglich gemacht werden. Letzteres wird mit einem Beispiel demonstriert.
Im fünften Kapitel werden noch theoretische Grundlagen für die Entwicklung eines Programms zur Primzahlberechnung behandelt. Dieses ist als Beispiel für die Umsetzung eines komplexen mathematischen Problems als parallele Grid-Anwendung zu sehen. Es

werden unter anderem die mathematische Problemstellung der Primzahlberechnung genau untersucht und Anwendungsgebiete – wie zum Beispiel in der Kryptographie – aufgezeigt. Anschließend werden Probleme – wie zum Beispiel die Wahl der Datenstrukturen - und Lösungsansätze bei der praktischen Primzahlberechnung diskutiert.
Schließlich geht das sechste Kapitel auf die Konkretisierung des Primzahlprogramms mit den im vorigen Kapitel besprochenen Methoden ein. Die Umsetzung geschieht für das im vierten Kapitel samt MPICH-G2 installierte Globus Toolkit. Es werden dabei die im dritten Kapitel vorgestellten MPI-Funktionen benutzt. Danach wird der verwendete Algorithmus im PRAM-Modell untersucht und die so gewonnenen Ergebnisse mit den Benchmarks des Programms verglichen.

Im Anhang sind die vorgestellten Quelltexte der in dieser Arbeit verwendeten Programme angegeben.

1. Grid Computing

In diesem Kapitel sollen die Hintergründe und die praktischen Realisierungen von Grids erörtert werden. Die Grundlage hierfür wird hauptsächlich die von Ian Foster et al. formulierte visionäre Vorstellung und deren Konkretisierung sein. Vor diesem Hintergrund sollen einige verteilte Systeme auf Grid-Eigenschaften hin untersucht werden. Anschließend wird auf einige große Grid-Projekte eingegangen, die hauptsächlich für die Bewältigung von Forschungsprojekten oder für die Forschung an Grids selbst eingesetzt werden - sogenannte *Test-Grids* (von dem englischen *Testbeds*). Den Abschluss bildet ein kurzer Überblick über die kommerziellen Anbieter von Grids, und es werden einige Forschungsprojekte zu Grids vorgestellt, an denen momentan gearbeitet wird. Viele der hier verwendeten Informationen gehen praktisch nicht weiter als bis 1998 zurück, was den modernen Charakter und die Aktualität von Grids unterstreicht.

1.1 Der Begriff des "Grid Computing"

Die Bezeichnung *Grid* (englisch für Gitter) ist etwa Mitte der neunziger Jahre in Anlehnung an den Begriff *Electrical Power Grid* (englisch für Stromnetz) entstanden, hinter dem sich die Vision einer überall verfügbaren, sich dem Bedarf anpassenden Infrastruktur verbirgt. Zu dem uns praktisch immer und überall verfügbaren Stromnetz gehören die verschiedensten Kraftwerke – egal ob mit Kohle, Wasser- oder Kernenergie betrieben –, die je nach Netzbedarf zugeschaltet oder heruntergefahren werden. Der Verbraucher merkt davon lediglich, dass er praktisch immer genug Strom hat, um seine momentan benutzten Geräte zu betreiben.
Ein solcher Zugriff auf Ressourcen bei Bedarf wurde schon früh prophezeit; z. B. stammt von Len Kleinrock aus dem Jahre 1969 folgende Aussage: "We will probably see the spread of 'computer utilities', which, like present electric and telephone utilities, will service individual homes and offices across the country" [177].

1.1.1 Eine erste Definition des Grid

Ian Foster und Carl Kesselman fassten in dem von ihnen 1998 herausgegebenen Buch "The Grid: Blueprint for a New Computing Infrastructure" [165] diese intuitive Idee genauer und gaben den Grids damit eine allgemein anerkannte Definition. Nach diesen beiden Autoren sind Grids eine Kombination von Hardware und Software, die zusammen eine einheitliche Infrastruktur bilden, welche Rechenkapazität und andere *Ressourcen* – wie zum Beispiel Festspeicher – zur Verfügung stellt.
Um das Analogon vom Stromnetz wieder aufzugreifen, werden an diese Infrastruktur noch einige Forderungen gestellt:

- die zur Verfügung gestellten Ressourcen sollen überall verfügbar sein
- die Verfügbarkeit der Ressourcen soll sehr zuverlässig sein
- es stehen einheitliche Schnittstellen zur Verfügung (eine wichtige Grundlage dafür, dass verschiedene Produkte von Firmen miteinander harmonieren und so – wie beim Stromnetz – gemeinsam eine einzige Infrastruktur bilden)
- es muss einen preiswerten Zugriff darauf geben

Letzteres ist offensichtlich eine visionäre Forderung, denn selbst das Stromnetz hat seine Zeit gebraucht, bis sich jeder in der westlichen Welt einen Anschluss leisten konnte – von den Milliarden Menschen, die auch heute noch ohne Strom sind, ganz zu schweigen.
Diese Definition ist weniger als Entscheidungshilfe für Fragen wie "Ist dies ein Grid?" gedacht, sondern mehr als Leitfaden zu einer Zeit des Aufbruchs im Grid-Bereich – eine Vision, die ein angestrebtes Ziel umriss, das man innerhalb einiger Jahre erreichen wollte.

1.1.2 Konkrete Definition durch die "Three Point Checklist"

Nur kurze Zeit später, im Jahr 2000, verfasste Ian Foster mit Carl Kesselman und Steven Tuecke den auf dem Buch aufbauenden Artikel "The Anatomy of the Grid" [157], in dem auf die bisherigen Projekte im Bereich Grid Computing Bezug genommen wird und eine konkrete, an die Praxis angepasste Definition von Grids erfolgt. Diese Definition stellt unter anderem fest, dass sich Grid Computing in der koordinierten gemeinsamen Nutzung von Ressourcen und im Lösen von Problemen in dynamischen – durchaus mehrfach

instanzierten – *virtuellen Organisationen* realisiert. Um zum Kern dieser Aussage vorzustoßen, muss zuerst diskutiert werden, was hinter den benutzten Schlagwörtern steckt.
Wie bei der Definition zwei Jahre zuvor ist der Begriff der Ressource sehr allgemein zu sehen. Dabei kann es sich um Rechenkapazität, Speicherkapazität in jeglichem Sinne, Software, Daten oder auch angeschlossene Geräte handeln. Ein wesentlicher Bestandteil der konkretisierten Definition ist die *Koordination*. Diese beinhaltet, dass der Anbieter von Ressourcen klar festlegt, was genau freigegeben ist und wer unter welchen Bedingungen darauf zugreifen kann. Zu beachten ist dabei, dass demnach keine zentralisierte Kontrollinstanz vorgesehen ist. Dafür gibt es aber die sogenannten *virtuellen Organisationen*. Dies sind Gruppen von Einzelpersonen oder Institutionen, die untereinander Zugriffs- und Nutzungsrechte der von ihnen zur Verfügung gestellten Ressourcen formuliert haben. Die in der Praxis daraus entstehenden Probleme sind zum einen die Konsistenz einer sich dynamisch verändernden virtuellen Organisation (denn das System muss sich auf immer neue Nutzer und neue Ressourcen mit neuen Nutzungsbedingungen einstellen) und zum anderen die Sicherstellung der Qualität der erbrachten Leistung der auf dem System laufenden Dienste. Letzteres beinhaltet im Besonderen die Konsistenz von Daten und Berechnungen, wenn z. B. ein benutzter Rechner nicht mehr zur Verfügung steht. Diesen Aspekt nennt man *Quality of Service* – zu deutsch etwa *Leistungsgüte*.

Versucht man aus der Definition des Jahres 2000 konkrete Anforderungen zu formulieren, so bleiben drei wesentliche Kriterien übrig, auf die im Einzelfall geprüft werden kann, ob es sich bei einem verteilten System um ein Grid handelt. Foster nennt sie "Three Point Checklist" [177]:

- die Koordinierung von Ressourcen, die nicht unter zentraler technischer Kontrolle stehen
- die Benutzung von standardisierten und offenen Protokollen sowie Schnittstellen
- die Sicherstellung der Leistungsgüte durch auf die jeweilige Anwendung optimierte Ressourcenauswahl

Nach der ersten, sehr allgemein gehaltenen Definition – oder besser: Vision – stehen nun direkte Anforderungen an ein verteiltes System, anhand derer sich entscheiden lässt, ob es sich um ein Grid handelt oder nicht. Im weiteren Verlauf der Arbeit werden bei Bewertungen von bestehenden Grids diese drei Punkte überprüft.

1.1.3 Alternative Definitionen

Natürlich ist die von Ian Foster et al. oben ausgeführte Definition nicht die einzige und erste, aber es ist die am meisten zitierte – dank Fosters Bekanntheitsgrad und der hohen Verbreitung der von ihm federführend betreuten Grid-Middleware "Globus Toolkit" (siehe Kapitel 2).
Insbesondere stellt sich ein Vertreiber eines verteilten Systems mit dem Wort "Grid" im Namen unter einem Grid wohl immer sein spezielles System vor, wie z. B. Suns "Grid Engine", die – für sich allein genommen – zumindest nach Foster wegen ihrer zentralisierten Kontrolle der Hosts kein Grid ist [177] (siehe auch Abschnitt 1.5.1.3).
Verallgemeinert kann man sagen, dass gerade in der Anfangszeit des Grid-Computing viele kommerzielle Produkte – insbesondere Middleware für Cluster (siehe auch 1.5.1.1) – sich mit der neuen Bezeichnung "Grid" schmücken, obwohl sie praktisch keine Charakteristika besitzen, wie sie am Anfang des Kapitels vorgestellt wurden.
Als Alternative zum verbreiteten Ansatz ein Grid über seine statischen Eigenschaften – wie zum Beispiel über bestehende Komponenten – zu definieren, charakterisieren Zsolt Nemeth und Vaidy Sunderam ein verteiltes System über seine Laufzeit-Semantik aus der Sicht einer Applikation [21]. Die Definition wird mittels der ASM (Abstract State Machine) formuliert, ein mathematisch fundiertes Modell, das zum Systemdesign und zur Analyse benutzt wird. Die ASM definiert neun *Regeln*, die von allen Prozessen auf einem vermeintlichen Grid – gemäß Definition – eingehalten werden müssen. Die genaue – sehr umfangreiche – Definition dieser Regeln kann in der entsprechenden Veröffentlichung nachgeschlagen werden.
Die im ASM Modell formulierten Ausdrücke zu den Regeln lassen sich formal nur sehr umständlich nachweisen; das Konzept der drei Punkte in 1.1.2 wirkt ansprechender. Jedoch liegt es in der Natur mathematischer Formulierungen, dass sie eine endgültige

Ja/Nein-Entscheidung ermöglichen. Die von Foster et al. verbal formulierten Bedingungen sind aufgrund von Sprachkunst und Interpretationgeschick dehnbar, was auch ihrem visionären Charakter entspricht, aber – wie man später sehen wird – nicht immer zu befriedigenden Entscheidungen führt.

1.1.4 Typische Merkmale von Grids

Im Folgenden gehen wir auf typische Merkmale von Grids ein, die sich in der Praxis bei bestehenden Grids herauskristallisiert haben.
Immer häufiger ist in der Praxis eine große geographische Ausdehnung zu beobachten, welche natürlich eine Fülle von Ressourcen mit sich bringt. Intuitiv ist ohnehin klar, dass für eine Sicherstellung an entsprechender Leistung für die Anwendungen (eine von Fosters Forderungen) auf eine bestimmte Ressourcenfülle zugegriffen werden muss. Der aus diesem weit verteilten Netz entstehende Kommunikationsaufwand ist naturgemäß sehr hoch, und wenn man noch berücksichtigt, dass bei solchen Strecken öffentliche Infrastrukturen wie etwa das Internet benutzt werden, muss noch ein beträchtlicher Overhead an Sicherheitsmaßnahmen hinzugerechnet werden. Viele Ressourcen (insbesondere Computer) werden bei großen Grids schon aus rein technischer Sicht nicht immer erreichbar sein und bilden so eine natürliche Dynamik im System, unabhängig von der gewollten Dynamik der in 1.1.2 definierten virtuellen Organisationen. Natürlich muss die sogenannte *Grid-Middleware* (siehe 1.2 und Kapitel 2) auf jeder einzelnen angeschlossenen Hardware zwischen den oft heterogenen Systemen (z. B. gänzlich unterschiedliche Betriebssysteme, Hardware und sonstige signifikante Abweichungen) und den Spezifikationen der geforderten einheitlichen Schnittstellen des Grids vermitteln. Es sind auch homogene Grids denkbar. Allerdings sind entsprechende homogene Strukturen, sogenannte *Cluster*, normalerweise ein Zeichen für reines verteiltes Rechen und bilden kein Grid an sich – sie können aber in ein Grid eingebunden werden. Zumindest die Einschränkung auf wenige unterschiedliche Systeme macht das Grid einfacher zu warten, was die Betreibung solcher Netze natürlich erheblich erleichtert. So setzt z. B. IBM für ihre e-Business Lösungen nur wenige grundlegende Architekturen ein – wie die eServer der p655, x335 und x345 Serie [89].

1.2 Umsetzung durch Grid-Middleware

Um ein Grid zu realisieren, sind vor allem zwei Aspekte wichtig. Zum einen muss ein vernetztes System bestehen, d.h. diverse vernetzte Hardwarekomponenten, die über entsprechende Software (Betriebssystem, etc.) und Hardware (Netzwerk) prinzipiell in der Lage sind miteinander zu kommunizieren. Bei dem heutigen Stand der Vernetzung, wo schon Drucker und andere Geräte mittels Standardlösungen über das Netzwerk angesteuert werden können, erfordert dies im Allgemeinen kein besonderes Augenmerk mehr. Um so wichtiger ist die Middleware, die aus dieser vorhandenen Struktur erst ein *verteiltes System* formt.

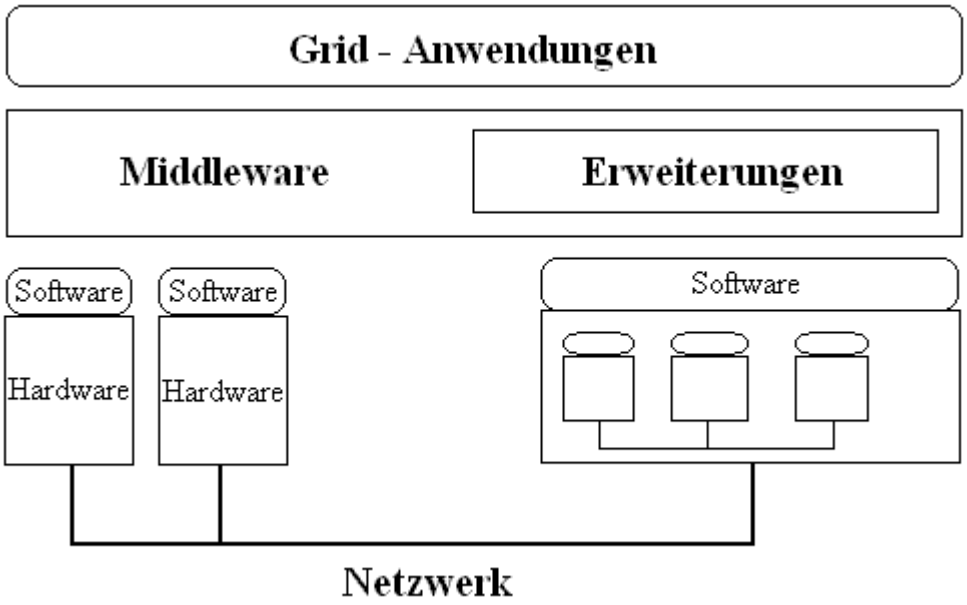


Abbildung: Ein verteiltes System als Grid

Middleware ist sozusagen ein Software-Aufsatz, der, obwohl auf jeder partizipierenden Komponente installiert, als Ganzes eine Schnittstelle des Systems nach außen zu entsprechenden Applikationen bildet. Es wird also eine virtuelle Schnittstelle geschaffen, die zur physikalischen Schicht der Hardware und natürlich auch der Software – wie z. B. dem jeweiligen Betriebssystem – übersetzt.
Eine Middleware, welche aus einem verteilten System ein Grid formt, heißt *Grid-Middleware*.
Durch die gerade bei Grids anzutreffende Vielfältigkeit der einzelnen Komponenten ist

ein hohes Maß an Interoperabilität der (häufig) sehr verschiedenen Systeme erforderlich, was die Entwicklung von Grid-Middleware offensichtlich erschwert. Dieser Umstand wiederum hat einige kommerzielle Anbieter dazu bewegt, nur für wenige Hard- und Software-Konfigurationen zu entwickeln (siehe dazu Abschnitt 1.5).
Auf die verschiedenen bisherigen Entwicklungen von Grid-Middleware soll hauptsächlich im nächsten Kapitel weiter eingegangen werden, wobei im Folgenden noch einige kommerzielle Produkte kurz vorgestellt werden.

1.2.1 OGSA – Open Grid Service Architecture

Wichtig für die Benutzung von Grid-Middleware ist, dass sie einheitlich von außen angesprochen werden kann. Daher besteht ein weiterer Schritt in der Standardisierung von Protokollen und Schnittstellen – wie von Ian Foster et al. im zweiten Punkt der "Three Point Checklist" gefordert – in der Formulierung der sogenannten *Open Grid Service Architecture*, oder kurz: OGSA [171].
Im Kernstück besteht diese Erweiterung bzw. Modernisierung der Grid-Protokolle aus der Implementierung von speziellen *Web Services* für Grids, den sogenannten *Grid Services*. Die durch das Global Grid Forum (GGF) [50] vorangetriebene Open-Source-Umsetzung als OGSI (siehe Abschnitt 2.3.2) hat die Zielsetzung, den Zusammenschluss aller Grid-Middleware – seien es Open-Source oder kommerzielle Produkte – zu schaffen. Dies soll unter anderem die Bezeichnung "das Grid" analog zu "das Internet" Realität werden lassen und so auch die Entwicklung von Applikationen für Grids unabhängig von den beteiligten Systemen und der Grid-Middleware ermöglichen. Allerdings gibt es auch andere Umsetzungen der OGSA, wie zum Beispiel vom Fraunhofer Institut (siehe Abschnitt 1.5.3.2).
Ressourcen werden hierbei als Dienst – also Grid Service – präsentiert, wobei dies nur eine virtuelle Sicht der Ressource bedeutet. Intern können diese Dienste wiederum auf mehrere andere Dienste zurückgreifen. Auf diese Weise wird eine völlige Entkopplung der physikalischen Ressourcen und ihrer Zuteilung erreicht. Dies ermöglicht Interoperabilität durch ein definiertes Serviceinterface und durch die dort zu den einzelnen Services definierten Protokolle.

Die erste Umsetzung von OGSA erfolgte in der Version 3 des "Globus Toolkit" als OGSI 1.0 [60], wobei aber weitere Revisionen folgten und noch folgen werden, da die GGF noch an weiteren notwendigen Definitionen von Diensten arbeitet und die API vieler bestehender Grid-Middleware nicht vollständig angesprochen werden kann (die Hauptmotivation der neuen Version 4 des Globus Toolkits). Unterstützung erfährt OGSA von vielen namhaften Firmen wie IBM, Microsoft, Platform, Sun, Avaki, Entropia und United Devices. In Zukunft wird es also hoffentlich nicht mehr notwendig sein, Grid-Applikationen Middleware-spezifisch zu entwickeln, was eine stärkere Belebung der Softwareentwicklung (vor allem auf dem kommerziellen Sektor) bedeuten könnte.

1.3 Aufgaben von Grids nach Anwendungsgebieten

Nachdem wir uns mit der Frage beschäftigt haben, was genau Grids sind und welche grundlegenden äußeren Merkmale sie aufweisen, gehen wir im Folgenden auf spezielle Anwendungsgebiete ein, die nicht nur die praktische Entwicklung von Grid Computing begleitet haben, sondern die auch sehr unterschiedliche Anforderungen an die Verfügbarkeit und an die Art der Ressourcen stellen.

1.3.1 Computational Grids

Ein sehr klassisches Gebiet sind die *Computational Grids*, denn schon früh in der Computerhistorie war es das Ziel von Computervernetzungen, eine höhere Rechenleistung zu erhalten. Häufig kamen zu diesem Zweck Cluster zum Einsatz. Praktisch werden dafür Grids als Infrastruktur zum massiven parallelen Rechnen benutzt. Die Benutzung von Grids kann besonders auf diesem Gebiet gegenüber dem klassischen parallelen Rechnen mit starrer Architektur vorteilhaft sein. Zum einen werden möglichst die am besten und geeignetsten am Grid partizipierenden Rechner benutzt, zum anderen kann so auf nicht ausgelastete Ressourcen an den benutzten Rechnern, z. B. Festplattenkapazität, immer noch in einem anderen Kontext effektiv zugegriffen werden [119].
Interessanterweise sind viele Computational Grids heutzutage – geographisch gesehen –

noch relativ klein. Meist sind alle teilnehmenden Rechner innerhalb eines Gebäudekomplexes oder sogar in einem einzigen Raum und damit innerhalb einer Organisation wie zum Beispiel eine Universität oder Firma. Dadurch will man die Kommunikationswege sehr kurz halten, um eine bessere Performance zu erreichen. Bei dieser Konfiguration des Grids ist klar, dass auf Dynamik kein großer Wert gelegt wird, denn das würde die Rechenzeitabschätzung sehr schwierig machen; schließlich ist es bei Aufgabenstellungen für Computational Grids wichtig, im Vorhinein zu wissen, ob das Ergebnis innerhalb einer Woche oder innerhalb eines Jahres zu erwarten ist [23]. Man kann diesen Aufbau des Grids mit einem Cluster vergleichen – und tatsächlich entsprechen einige Middleware-Lösungen der eines Clusters und scheinen nur zufällig den Aufdruck "Grid" zu tragen. Allerdings spricht nichts dagegen einen Cluster als eigenständige Ressource in ein Grid einzubinden.
Da Computational Grids einer der ersten realisierten Grid-Typen ist, gibt es für diesen Gebrauch mehrere etablierte Applikationen, wie zum Beispiel MPICH-G, eine MPI-Implementierung für Grids [120]. Leider gibt es noch keine ausgereiften bzw. verbreiteten Programmierumgebungen für Grids, welche die spezielle Dynamik der Grids zufriedenstellend unterstützen.

1.3.2 Data Grids

In zunehmendem Maße werden für wissenschaftliche Berechnungen sehr große Datenmengen benötigt. Dies resultiert unter anderem daraus, dass für genauere Ergebnisse mehr Daten berücksichtigt werden müssen und dass die Daten aus unterschiedlichen Quellen (z. B. Forschungszentren oder Messstationen) stammen können und auch von verschiedenen Teilnehmern ausgewertet werden sollen. Einerseits ergibt sich daraus das Problem, dass die Daten allen Teilnehmern gleichzeitig zur Verfügung stehen sollen und andererseits immer neue Daten – oder auch veränderte – als Ressourcen hinzukommen. Die Anforderungen an das Grid sind zum einen sicherheitstechnischer Natur, – z. B. wenn sensible Forschungsdaten betroffen sind – aber vor allem wird auf zeitkritischen Zugriff und Konsistenz der Daten Wert gelegt. Generell sind solchermaßen primär benutzte Grids heterogener Natur und geographisch sehr groß.

An einer allgemeinen Implementierung für solche Grids – sogenannte *Data Grids* – wird seit etwa 1999 gearbeitet [160], und diese findet ihren Einsatz unter anderem bei der Auswertung der Daten vom CERN, dem europäischen Kernforschungszentrum [48].
Das Kernstück der Data Grids besteht nach [160] aus dem *Storage System*, dem Teil, der auf logischer Ebene sämtliche Dateioperationen zur Verfügung stellt, und dem *Metadata Service*, der die Informationen – sogenannte *Metadaten* – über das Grid, also auch über die Dateien, verwaltet. Anwendungen können über den Metadata Service sowohl Informationen über Dateien erhalten als auch preisgeben. Damit können nun problemlos Daten als Ressourcen freigegeben und manipuliert werden, gemäß den dem Metadata Service mitgeteilten Bedingungen. Um nun eine optimale Ausnutzung der Daten zu gewährleisten, wie zum Beispiel um den weiter oben geforderten beschleunigten Zugriff zu realisieren, kann ein sogenannter *Replica Manager* real existierende Kopien an verschiedenen Stellen des Grids erzeugen bzw. löschen.
Dies ist natürlich nur ein kurzer Umriss der Arbeitsweise eines Data Grids, aber die Priorität nimmt durch die Nachfrage an effizienter Datenbearbeitung im großen Stil zu.

Insbesondere ist als Realisierung ein Grid doppelt interessant, da zum rechenintensiven Auswerten der Daten die Rechenleistung eines Computational Grids durchaus benötigt wird – man braucht also nur eine einzige Infrastruktur, die beides bereitstellt [172]. Dieses Konzept ist für aufwendige Forschungsprojekte, welche sich mit der Erfassung und Auswertung von kontinuierlichen Datenströmen (wie astronomische Daten oder die Messphalanx eines Teilchenbeschleunigers) befassen, schon früh erkannt worden. Es wurden Kooperativen gebildet, welche die Planung und Entwicklung von Infrastruktur zur Bewältigung von Datenmengen im Petabyte-Bereich als Ziel haben. Der grundlegende Lösungsansatz dafür ist, mit allen weltweit beteiligten Forschungseinrichtungen eines solchen Projektes ein Grid zu bilden und so deren jeweilige Ressourcen als Grundlage zur gemeinsamen Datenspeicherung und Auswertung zu nutzen [32,38,79,104,134,155,161].

1.3.3 Access Grids

Ein teilweise mit dem der Data Grids verwandtes Anwendungsgebiet haben die sogenannten *Access Grids*. Während man bei den Data Grids immer ein Szenario von mehreren Teilnehmern hat, die auf eine sehr große Datenmenge zugreifen, sind Access Grids etwas differenzierter. Im Allgemeinen steht hier die Möglichkeit und Optimierung des Zugriffs auf eine bestimmte Ressource im Vordergrund, insbesondere bei sehr vielen Teilnehmern.
Ein mögliches Anwendungsgebiet könnte ein Grid von hochverfügbaren Servern sein, die je nach Belastung Ressourcen von anderen Servern mitbenutzen, um Belastungsspitzen auszugleichen und Latenzzeiten zu minimieren. Ein anderes Beispiel wäre die "abgespeckte" Version eines Data Grids, wobei (meist im kleineren Rahmen) die Hochverfügbarkeit einer bestimmten Ressource – wie etwa Zeitschriftenartikel oder Log-Dateien – gefragt ist (analog zu den Seiten eines Web-Servers). Auch ist der dynamische Zugriff auf Softwarelizenzen ein Anwendungsgebiet, das in Zukunft mehr Bedeutung gewinnen könnte, denn eine Firma braucht eigentlich nur so viele Lizenzen wie Leute gleichzeitig mit der Software arbeiten, und nicht, wie viele Rechner insgesamt benutzt werden.
Sicherlich fällt dem Betrachter bei den oben genannten Beispielen schnell eine proprietäre Software ein, die ein derartiges Problem ohne Grid löst. Dabei sollte man aber beachten, dass bei einer proprietären Lösung meist auf Heterogenität verzichtet werden muss und ein Grid nur unter anderem als Access Grid genutzt werden kann. Der Vorzug zeigt sich insbesondere dann, wenn über das Grid verschiedene Dienste realisiert werden, wohingegen die Anwendung von mehreren proprietären Lösungen sehr umständlich sein kann (z. B. könnte jede eine eigene Benutzerverwaltung erfordern) oder aus technischen Gründen gar unvereinbar ist.

1.4 Grids in der Praxis

Auch wenn die Vision von Ian Foster et al. von einem praktisch weltumspannenden Grid noch in weiter Ferne liegt, so existieren doch schon Grids von beachtlicher Größe und

Leistung. Natürlich gibt es auch Grids, die zur “Forschung an Grids” dienen. Dort werden neue Applikationen, Protokolle oder neue Grid-Middleware getestet. Solche *Test-Grids* sind natürlich auch wichtig, um eine Aussage über die tatsächliche Qualität der Leistung zu machen, die sich in der Theorie nur sehr schwer abschätzen lässt. Dazu zählen auch quasi "öffentliche Grids", auf denen Entwickler und Privatpersonen eigene Programme zu Testzwecken ausführen lassen können [73,87]. Des Weiteren sind viele Test-Grids und insbesondere Data Grids per Standleitung oder Internet miteinander verbunden und können so bei exzessivem Bedarf an Ressourcen – z. B. bei bestimmten Berechnungen – zu einem einzigen Grid verschmelzen. Dies ist aber (noch) nicht zu verallgemeinern, denn das ist nur unter bestimmten Voraussetzungen möglich [25,34].

1.4.1 Gridähnliche Projekte

Bevor hier einige existierende Grids vorgestellt werden, soll noch ein Blick auf ähnliche Projekte geworfen werden. Dies ist vor allem deswegen ein interessantes und wichtiges Thema, da es hilft, eine deutliche Grenzline zu dem Begriff des Grid zu ziehen.
Wie vorher schon erwähnt, gehen wir von der in 1.1.2 vorgestellten "Three Point Checklist" von Ian Foster aus und legen uns damit auf eine bestimmte Sichtweise auf Grids fest. Dies ist aber in zweierlei Hinsicht gerechtfertigt. Zum einen haben Foster und seine Partner durch wegweisende theoretische Artikel über das Grid-Konzept die moderne Grid-Entwicklung begleitet und offensichtlich stark beeinflusst, zum anderen ist das von ihm mitgetragene "Globus Projekt" [162] durch seine hohe Verbreitung und kontinuierliche Weiterentwicklung praktisch zum Vorreiter und Quasi-Standard im Grid-Bereich geworden. Tatsächlich erfüllen nicht alle Produkte mit dem Wort "Grid" im Namen die in der Liste aufgeführten drei Kriterien.

1.4.1.1 Das SETI@home Projekt

Das von der NASA durchgeführte SETI (Search for ExtraTerrestrial Intelligence) Programm besteht im Wesentlichen aus vielen Antennen, die eine große Datenmenge an

aufgefangenen Radiosignalen liefern [149]. Die Aufgabe besteht darin, aus dem immer vorhandenen kosmischen Hintergrundrauschen abweichende Signale zu isolieren, die möglicherweise extraterrestrische Funksprüche sein könnten. Da die Datenmenge sehr groß ist, reicht die Kapazität von wenigen hundert Rechnern nicht aus, um sie zu bewältigen.
Gelöst wird das Problem, indem entsprechende Server bei der NASA an frei verfügbare Client-Programme Teile des Datenstroms schicken, welche diese dann auswerten und das ermittelte Ergebnis zurückschicken. Prinzipiell nehmen so weltweit viele hunderttausend Rechner teil, aber letztlich bleibt es bei einer einfachen Client-Server Verbindung. Allerdings übt der Zentralserver eine technisch zentralisierte Kontrolle aus, denn er hat ein genaues Wissen über den Zustand des Systems und der Benutzeranfragen sowie volle Kontrolle der einzelnen zur Verfügung gestellten Rechner. Daher erlaubt dies keine Einstufung als Grid nach der Liste von Foster. Zsolt Nemeth und Vaidy Sunderam hingegen sehen ihre formale Anforderungen als erfüllt an [21].

1.4.1.2 Filesharing

Ungeachtet der angespannten rechtlichen Lage sind sogenannte Filesharing-Programme wie Gnutella weltweit sehr beliebt. Dabei hat jeder Teilnehmer einen Client und macht über ihn lokale Dateien für andere Teilnehmer zugänglich. Jeder in dieser virtuellen Organisation kann nun auf derart freigegebene Dateien von anderen zugreifen, wobei eine von verschiedenen Personen mehrfach angebotene Datei von allen gleichzeitig heruntergeladen wird, welches die benötigte Zeit erheblich verkürzt und vor Datenverlust bei Ausfällen einzelner Clients schützt.
Bei dieser Anwendung gibt es offensichtlich keine zentrale Kontrollinstanz und für die Qualität der Leistung spricht vor allem die hohe Anzahl der Nutzer. Die Forderung nach standardisierten, offenen und vielfältigen Protokollen und Schnittstellen kann man aber selbst bei äußerster Dehnung der Begriffe nicht ganz als erfüllt erachten. Selbst Ian Foster fällt die Entscheidung für oder gegen ein Grid in seinem Artikel "What is the Grid?" [177] schwer, allerdings kann man sie als erste *gridähnliche* Systeme ansehen.

1.4.1.3 Das World Wide Web

Ohne Zweifel können dem Web vielseitige und standardisierte Protokolle sowie Schnittstellen zugesprochen werden. Dazu kommen die enorme angebotene Ressourcenvielfalt und -menge, die sich einer zentralen Kontrolle entziehen. Foster spricht dem Web – zumindest momentan – den Status als Grid ab, da seiner Meinung nach die zur Verfügung gestellten Ressourcen nicht entsprechend koordiniert werden können, um eine zufrieden stellende Qualität an Leistung zu erhalten [177] (beispielsweise findet man bestimmte Informationen nicht, obwohl sie verfügbar wären). Da das Web eigentlich nur für einen Informationsaustausch mit der dafür entsprechend geringen Bandbreite und sonstigen knappen Ressourcen konzipiert ist, verwundert das nicht weiter. Möglicherweise kann es in der Zukunft durch Web- und Grid-Services zu einer Verzahnung von Web und Grids kommen, aber das bleibt abzuwarten.

1.4.2 Frühe Pilotprojekte

Wie in vielen Bereichen gibt es auch beim Grid Computing verschiedene Ereignisse und Projekte im Vorfeld, die es erst ermöglicht haben, Grid-Infrastrukturen sowohl auf der Hardwareseite als auch im Bereich der Middleware zu entwickeln oder gar erst als praktisch umsetzbar anzusehen. In diesem Abschnitt sollen zwei wegweisende Projekte kurz vorgestellt werden.

1.4.2.1 CASA Gigabit Testbed

Im Zuge der Forschung an Hochgeschwindigkeitsnetzwerken wurde das von der National Science Foundation (NSF) und Advanced Research Projects Agency (ARPA) finanzierte CASA Gigabit Testbed 1993 initiiert [20]. Das Ziel dieses Projektes war es, sieben Hochleistungsrechner der Partizipienten – California Institute Of Technology (Caltech), Jet Propulsion Laboratory (JPL), Los Alamos National Laboratory (LANL), San Diego Supercomputer Center (SDSC) und die University Of California Los Angeles (UCLA) –

so zu vernetzten, dass sie als eine effektive Einheit fungieren konnten.
Die Herausforderungen bestanden zum einen in dem Zusammenschluss des heterogenen Systems und zum anderen im effektiven und schnellen Datenaustausch der weit voneinander entfernten Rechner, was mit einem Datendurchsatz von über einem Gigabit pro Sekunde erreicht werden sollte (daher die Namensgebung). Auf der Applikationsseite sollten Anwendungen in den Bereichen Global Climate Modeling (GCM), Interactive Real-Time Terrain Visualization (CALCRUST) und Quantum Chemistry Reaction Dynamics diese Infrastruktur nutzen.
Die Resultate dieses Pilotprojekts waren wegweisend für viele nachfolgende Unternehmungen [18]. Es wurde also nicht nur ein geographisch stark verteiltes Netzwerk für Hochleistungsrechnen geschaffen, das einen überlinearen Geschwindigkeitszuwachs aufwies, sondern auch hierarchische Programmiermodelle für heterogenes Arbeiten. Genau betrachtet kommt es nicht darauf an, wie es exakt ermöglicht wurde, sondern dass es überhaupt möglich ist. Somit sind dies die ersten Schritte in den modernen Bereich des Grid-Computing.

1.4.2.2 I-WAY

Eine dem heutigen Gridverständnis entsprechende Unternehmung war das I-WAY (Information Wide Area Year) Projekt [167]. Im Jahre 1995 durchgeführt, war es das erklärte Ziel, eine weitläufige Infrastruktur für neuartige, geographisch verteilte High-Performance-Anwendungen zu schaffen. Insgesamt wurden Hochleistungsrechner, Massenspeicher und AVDs (Advanced Visualization Devices) an 17 nordamerikanischen Einrichtungen mittels eines ATM (Asynchronous Transfer Mode) Netzwerks verbunden. Die ATM Netzwerke versprechen wegen ihren low-level Netzwerkfunktionen einen hohen Datendurchsatz durch den geringeren Overhead im Gegensatz zu Multi-Layer-Verbindungen wie beispielsweise beim Internet. Bei der Entwicklung der Middleware für dieses Projekt stand die Ausführung einer möglichst weiten Palette von Applikationen im Mittelpunkt. So verwundert es nicht, dass bis zu 70 Anwendungen ausgiebig getestet wurden, allein 60 davon wurden bei der Super Computing im Dezember 1995 in San Diego vorgeführt [148].

Obwohl das Projekt nur ein Jahr lief – daher auch der Name – waren die daraus gezogenen Erkenntnisse und die währenddessen entwickelte Software die Grundlage für nachfolgende Unternehmungen, wie beispielsweise das Globus Toolkit [166]. Es ist also nicht weiter verwunderlich, dass Ian Foster, der die Softwareentwicklung des I-WAY Projekts leitete, in der heutigen Grid-Computing-Community eine führende Rolle einnimmt.

1.4.3 Bestehende Grids

Dass die Visionen von Ian Foster und seinen Mitarbeitern (zumindest bis zu einem gewissen Grad) ihre Umsetzung gefunden haben, sollen die nachfolgenden Beispiele zeigen. Dabei ist natürlich klar, dass den Teilnehmern der meisten Grids in der Praxis mehr an den besonderen Eigenschaften, wie sie weiter oben diskutiert worden sind, gelegen ist, als für allgemeines Ressource-Sharing zur Verfügung zu stehen. Deshalb werden sie auch die für ihre Anwendung spezifische Bauweise und das entsprechende Systemdesign aufweisen.

1.4.3.1 Butterfly.net

Das relativ neue Unternehmen Butterfly.net Inc. (seit Mai 2005 umbenannt in Emergent Game Technologies) bietet in Zusammenarbeit mit IBM seit Ende 2002 ein Access Grid an, welches speziell für Online-Spiele (mit Schwerpunkt auf virtuelle Welten) für PCs, Spielekonsolen und auch PDAs über zwei Jahre lang entwickelt wurde [13].
In seiner untersuchten Konfiguration besteht das Grid aus zwei Clustern mit etwa 50 IBM e-Servern der xSeries, auf denen Linux läuft. Die als Spiel- und Datenbank-Server genutzten Rechner sind mittels optischer Hochgeschwindigkeitsleitungen miteinander vernetzt und pro Cluster sollen etwa eine Million Spieler gleichzeitig abgefertigt werden. Dieses soll in Zukunft noch ausgebaut werden, was also einen Zuwachs von Ressourcen während des laufenden Betriebs bedeutet. Die Tools und Bibliotheken zur Nutzung des Grids sind für Entwickler frei verfügbar und notwendig, da speziell entwickelte

Protokolle zum Einsatz kommen.
Diese kommerzielle Form ist natürlich ein Spezialfall eines Grids, denn alle Ressourcen sind innerhalb einer einzigen Organisation vorhanden. Allerdings ist die Wartung des Grids – zumindest auf der Seite der Anwendersoftware und ihrer speziellen Anbindung an die Middleware – auch von außen möglich.
Der Vorteil der Grid-Architektur besteht einerseits in der dynamischen Lastverteilung – so können Stoßzeiten aufgefangen werden und man kann auf Popularitätsveränderungen oder Beta-Test Projekte mit wenigen Teilnehmern problemlos reagieren. Insbesondere erlaubt die dynamische Verlegung während des Betriebs bei Ausfällen oder Wartungsarbeiten das Einspielen von Patches (was bei Spielen eine alltägliche Notwendigkeit ist). Das laufende Spiel muss dafür nicht unterbrochen werden.
Als Middleware kommt – wie bei IBM üblich – das Globus Toolkit zum Einsatz, wobei die OGSA explizit implementiert sind.

1.4.3.2 TeraGrid

Das durch die National Science Foundation (NSF) in Amerika geförderte TeraGrid [155] ist ein im Jahre 2001 gegründetes ehrgeiziges Projekt, um die weltgrößte und schnellste verteilte Infrastruktur für wissenschaftliche Forschung zu werden. Hervorgegangen aus dem vorangegangenen PACI Programm (Partnership for Advanced Computational Infrastructure) – welches die Funktion als grundlegendes Forschungs-Grid haben wollte – wurden zu dessen Teilnehmern immer mehr Institutionen eingebunden. Insgesamt sind zur Zeit fünf Rechenzentren beteiligt: das National Center for Supercomputing Applications (NCSA) an der Universität von Illinois in Urbana-Champaign, das San Diego Supercomputer Center (SDSC) an der Universität von California in San Diego, das Argonne National Laboratory in Argonne in Illinois, das Center for Advanced Computing Research (CACR) an dem California Institute of Technology in Pasadena und seit Oktober 2002 das Pittsburgh Supercomputer Center (PSC).
Erklärtes Ziel dieses Projekts sind über 21 TeraFlops an Rechenleistung und mehr als 1 PetaByte an Datenverarbeitung bei einer Kommunikationsgeschwindigkeit von 40 GigaBits pro Sekunde. Damit repräsentiert es problemlos alles, was hier bisher über Data

Grids und Computational Grids gesagt wurde. Realisiert wird dieses Mammutprojekt über das Globus Toolkit und unter Implementierung von Grid Services der OGSA. Da hierbei verständlicherweise die bisherigen Spezifikationen voll ausgereizt werden und unvorhergesehene Spezialfälle auftreten, arbeiten die Betreiber eng mit dem Global Grid Forum (GGF) zur Entwicklung neuer – bzw. erweiterter – Standards zusammen.

1.4.3.3 NASAs Information Power Grid (IPG)

Das Information Power Grid Projekt (IPG) wurde gegründet, um das Ziel zu erreichen, den Bedarf der NASA an Ressourcen in hohen Größenordnungen zu decken und daher die vorhandenen Ressourcen der verschiedenen NASA Forschungszentren zusammenzuführen [101]. Ein Grid erfüllt die Vorstellungen der NASA, eine gemeinsame Sicht der Ressourcen für Anwendungen zu haben, aber gleichzeitig die Administration und Kontrolle der Einzelressource lokal zu belassen. Ein ständiger Ausbau und Kooperativen mit anderen Rechenzentren aus staatlichen, wissenschaftlichen und auch kommerziellen Bereichen sollen das IPG für die wachsenden Anforderungen der Aufgabengebiete in den kommenden Jahren rüsten.
Die veröffentlichte (aber wahrscheinlich nicht mehr aktuelle) Konfiguration des Grids besteht aus den drei der NASA eigenen Ames-, Glenn- und Langley-Forschungszentren. In ihnen arbeiten sieben SGI Origin 2000 Systeme, ein mit Condor (siehe Abschnitt 2.2.1) betriebener Workstation-Pool von 280 Rechnern, ein Linux Cluster und vier Sun Sparcstations sowie vier Directory Service Server. In diesem Test-Grid laufen als Hauptanwendungen numerische Berechnungen mit NPSS und OVERFLOW. Einige Forschungsbereiche dazu werden in dem Abschnitt 1.5.3.1 genannt.
Realisiert wird das Grid im Kern durch das Globus Toolkit, wobei leider nicht zu erfahren war, welche Version benutzt wird und welche von der NASA entwickelten Grid-Werkzeuge sich im Einsatz befinden. Da es sich offenkundig um ein Test-Grid handelt, kann es in den Leistungsdaten mit dem TeraGrid und anderen nicht konkurrieren, aber ein weiterer Ausbau ist in jedem Fall geplant (bzw. schon umgesetzt).

1.5 Marktverbreitung und zukünftige Entwicklungen

Letztlich wird jede Technologie an ihrem Einsatz am sogenannten "Markt" gemessen, was sich auch das Grid Computing gefallen lassen muss. Zu der ganz am Anfang formulierten Vision eines alles versorgenden Netzes ist es bisher noch nicht gekommen, aber IBM bietet mit seinem e-Business "On Demand" mietbare Grid-Leistung an, was der Vision schon näher kommt, wenn man bedenkt, dass man heutzutage auch erst einmal mit einem Stromlieferanten einen Vertrag abschließen muss, bis man seine Leistung bekommt [89]. Aber auch mit der hier vorgestellten OGSA scheint ein großer Schritt in die entsprechende Richtung gemacht worden zu sein, denn wenn die darin enthaltenen InterGrid-Protokolle überall umgesetzt würden (wie in Form der OGSI beim Globus Toolkit geschehen), so könnte das die Technologie-Barriere der verschiedenen Entwickler überwinden und die vielen Grid-Realisierungen irgendwann zu einem einzigen Grid zusammenschweißen.

1.5.1 Kommerzielle Produkte

Obwohl Grids noch häufig als Zukunftsvisionen und "Near-Future" Projekte angesehen werden, werben doch viele namenhafte Firmen mit Grid-Lösungen und dem zusätzlichen Potential, das durch die neue Software aus den vorhandenen Rechnersystemen gewonnen werden kann. Dabei wird hauptsächlich das Argument ins Feld geführt, die vielen vorhandenen Desktop-Rechner in den Unternehmen seien praktisch überhaupt nicht ausgelastet und mit einer Einbindung in ein das Unternehmen umspannendes Grid könne man ungeahnte Leistungssteigerungen erzielen. Leider ist dieses Argument nur bei rechenintensiven, verteilten Anwendungen wie "Monte Carlo"-Simulationen nachzuvollziehen, aber der Ausdruck "Grid" wird anscheinend nur allzu oft mit "verteiltem Rechnen" gleichgesetzt.
Selbst IBM greift in seiner Einleitung zu Grids auf diese Argumentation zurück und spricht vom zu 95% ungenutzten Prozessor-Potential eines durchschnittlichen Desktop Rechners [88]. Entropia errechnet gar 28-fache Beschleunigung durch Nutzung von PCs statt spezialisierter Rechnerfarmen bei einem Kunden und legt optisch sehr

beeindruckende Statistiken vor [44]. Solche Zahlenspiele beziehen sich aber eher auf Extremfälle und es sollte daran erinnert werden, dass Grids mehr vermögen, als ungenutzte Prozessorleistungen zu aktivieren, wie man z. B. bei dem Avaki Data Grid unter 1.5.1.2 erfahren kann.

1.5.1.1 Entropia DCGrid

Die Idee des DCGrid von Entropia ähnelt der eines Computational Grids [43]. Das "Grid" in der Version 5.1 verwirklicht sich als Client/Server-Kombination. Der Client wird auf allen Windows-PCs der Kunden-Firma installiert und hängt sich in die Idle-Routine von Windows ein. Das ermöglicht die Benutzung von sonst ungenutzten Prozessor-Ressourcen. Der Server hingegen hat die Aufgaben der Katalogisierung und die eines Job-Schedulers. An den Server geschickte Jobs – genau genommen nur für verteilte Systeme entwickelte – werden dann an die Clients verteilt, die ihrerseits nach beendeter Ausführung das Ergebnis an den Server zurücksenden. Besonders stolz ist Entropia auf die zentrale Administration der Clients, welche bei Bedarf auch über ein Web-Interface möglich ist, und gut die hohen Sicherheitsmaßnahmen auf Seiten der Clients. Im Gegensatz zu beispielsweise SETI@home ist dies ein absolut homogenes verteiltes System, aber die Arbeitsweise ist analog.
Klar ist, dass die zentrale technische Kontrolle der Ressourcen, deren nicht vorhandene Vielfalt – nur Prozessoren und evtl. Speicher – nicht gerade für ein eigenständiges Grid nach der Vorstellung der Visionäre um Foster spricht. Es sieht eher nach einem kassischen Cluster mit dem vielversprechenden Aufdruck "Grid" aus.
Hingegen hat sich in den neueren Versionen seitens der Protokolle einiges getan. In einer Broschüre [42] wird die Unterstützung von OGSA und Globus Grid Standards beworben. Damit scheint Entropia ihre ursprünglich proprietäre Software in bewährte Grid-Standards als eigene Ressource eingliedern zu wollen.

1.5.1.2 Avaki Data Grid

Wie der Name schon impliziert, handelt es sich bei dem Avaki Data Grid um eine Grid-Lösung von Datenverfügbarkeit [6]. Da sie sich selbst als "Database-Middleware" versteht, verwundert es nicht, dass sie hauptsächlich auf den Zugriff und die Konvertierung von Datenbankzugriffen spezialisiert ist. Dabei kann die Middleware Zugriffe und Anfragen der verschiedensten Arten vornehmen, wie ODBC, JDBC, Web Services (SOAP), Dateizugriff und JSP/Tag Bibliotheken. Die Ergebnisse werden bei Bedarf per erstelltem XSL Stylesheet in die gewünschte Form gebracht.
Auf die Ressourcen kann man über einen standardisierten Daten-Katalog wie über eine lokale Ressource zugreifen; die Datentransparenz ist vollkommen gegeben. Dabei werden die Ressourcen lokal verwaltet und es wird auch nur vor Ort definiert, ob und wie darauf zugegriffen werden darf. Da für ein kommerzielles Produkt insbesondere die Qualität der Leistung wichtig ist um erfolgreich zu sein, scheint es sich im engeren Sinne um ein Grid nach Foster zu handeln, wobei die Art der Ressourcen letztlich auf Daten beschränkt ist. In der untersuchten Version 4.0 wurde noch keine der OGSA entsprechende Schnittstelle realisiert [4].

1.5.1.3 Sun ONE Grid Engine (Enterprise Edition)

Sun vertreibt seine ONE Grid Engine 5.3 für SPARC und x86 Prozessoren unter Solaris und für Linux auf x86 kompatiblen Prozessoren, wobei nur die Enterprise Edition den uneingeschränkten Funktionsumfang bietet [152]. Bei diesem Konzept hat jeder partizipierende Rechner einen *Agenten* im Hintergrund laufen, der regelmäßig mit anderen Agenten in Kontakt steht, um die momentane Auslastung der Ressourcen zu übermitteln. Das interne Verwalten der Ressourcen geschieht im gesamten System verteilt, das Job-Scheduling wird anhand der von den Agenten gesammelten Informationen erledigt. Insgesamt werden mehrere Ressourcen (wie Prozessoren, Speicher und Plattenkapazität) und deren Verfügbarkeit berücksichtigt (wie zulässige Last, zulässige Jobanzahl, Verfügbarkeitszeitraum und weitere). Leider ist nicht zu erfahren gewesen, wie die Freigabe der Ressourcen erfolgt. Letztlich bildet die Grid Engine aber einen Cluster,

welcher einer zentralen Kontrolle unterliegt. Damit ist sie kein eigenständiges Grid nach der Liste von Foster. Auch Zsolt Nemeth und Vaidy Sunderam sehen aufgrund ihrer Definition die Grid Engine nicht als Grid an [21].
Ebenso ist nicht sicher, welche Protokolle verwendet werden, aber es wurde noch nicht die Open Grid Service Architecture implementiert.
Allerdings ist die Version 5.3 die letzte dieser Produktlinie und wird nur noch bis 2010 unterstützt. Bei den neueren Ansätzen der Grid Engine scheint Sun sich mehr in Richtung Grids zu entwickeln [153].

1.5.1.4 Platform Globus Toolkit

Analog zu den bekannten Linux-Distributionen – wie SuSE oder RedHat – besteht das Platform Globus Toolkit aus dem bewährten Open-Source Globus Toolkit mit teilweise selbst entwickelten ergänzenden Tools und Applikationen [139]. Das Toolkit kann zum Beispiel Cluster, welche mittels anderer Software wie Platforms eigene Load Sharing Facility, den Condor Workload Manager oder der Sun Grid Engine gebildet wurden, als eigenständige Ressourcen einbinden. Es gibt vorkompilierte Versionen für beinahe jeden Prozessortyp und ausführliche Anleitungen zur Installation und Einrichtung.
Das Globus Toolkit wird im nächsten Kapitel genauer besprochen, aber aus der Website von Platform [138] ging nicht hervor, welche Version des Toolkits zur Zeit benutzt wird.

1.5.1.5 IBM Grid Computing

Eine Ausnahme zu den vorangegangenen reinen Softwarelösungen (wenn man davon absieht, dass die auf Windows basierenden Anwendungen wie DCGrid nur auf x86-Prozessoren laufen und damit das benutzte System praktisch feststeht) bildet hier IBM, die hauptsächlich Komplettsysteme vertreiben. Als Grid-Middleware kommt hier das Globus Toolkit in der hauseigenen Distribution Grid Toolbox (aktuell in der Version V3) mit unterschiedlichen Erweiterungen zum Einsatz (unter Umständen kombiniert mit LoadLeveler zur Cluster-Bildung), welches auf den verschiedensten Hardwarelösungen

von IBM aufsetzt [92]. Das sind hauptsächlich x86- und Power-Prozessoren mit AIX-, Linux- oder Windows-Systemen.
Unabhängig von dem Vertrieb der Komplett-Systeme bietet IBM e-Business "On Demand" an, es vermietet also Rechenleistung bzw. Computerinfrastruktur für bestimmte Aufgabengebiete [95]. Nach Aussagen von IBM ist es nur dank der Flexibilität ihrer Grids möglich, solchen Kunden adäquate Leistung zu liefern. Dies bedeutet einen weiteren Schritt zur Verwirklichung der Vision der allgemein verfügbaren Ressourcen durch ein Grid.

1.5.2 Projekte in der Entwicklung

Wie schon anfangs erwähnt, ist das Gebiet des Grid Computings ein relativ neues und es gibt viele Bereiche und Ideen, die sich noch in der Entwicklung befinden. Zwei davon sollen im Folgenden genauer vorgestellt werden.

1.5.3.1 NASA Agency of Science (NAS)

Die Forschungsabteilung der NASA beschäftigt sich mit ihrer IPG-Gruppe mit der direkten Forschung an Grids, wie z. B. Werkzeuge zur Programmentwicklung für verteilte Systeme, ein allgemeiner Ansatz zum gemeinsamen Verwalten von Ressourcenanforderungen, System Management, Benutzeridentifikation und Sicherheit. Dazu kommen unter anderem ein komplettes Paket an Werkzeugen, Diensten und einer Infrastruktur zum Verwalten und Zuteilen von dynamisch gebündelten Ressourcen – wie Prozessoren, Daten-/Informationsträger, Kommunikationssystemen, "Real-Time" Datenquellen und Instrumenten bis hin zu Mitarbeitern [123]. Dieser Ansatz geht also etwas weiter, als nur die klassischen Ressourcen wie Rechnleistung und Speicherplatz zu beachten.

1.5.3.2 Fraunhofer Resource Grid

Das Fraunhofer Resource Grid (FhRG) [49] ist ein Zusammenschluss der Kapazitäten von fünf Instituten der "Fraunhofer Gruppe Informations- und Kommunikationstechnik". Dieses seit 2001 bestehende Grid basiert inzwischen auf dem Globus Toolkit 2.4 (zum Ende des Jahres 2004) mit einer eigenen OGSA-Implementierung. Um die problemlose und effiziente Kommunikation zwischen den einzelnen Soft- und Hardwarekomponenten zu gewährleisten, wurde eine XML-basierende Beschreibungssprache entwickelt - die Grid Application Defintion Language (GADL).
Diese besteht aus den folgenden spezialisierten Untersprachen:

- GResourceDL, zuständig für die Beschreibung der Ressourcen im FhRG
- GJobDL, zuständig für die genaue Beschreibung eines Jobs im Grid und beinhaltet daher Informationen über die genutze Menge der Ressourcen, ihrer Abhängigkeiten und logischen Verflechtungen
- GInterfaceDL, zuständig für die Beschreibung der Softwareschnittstellen der interagierenden Komponenten
- GDataDL, zuständig für die Beschreibung der Daten und des Datendurchsatzes

Damit ist das System in der Lage, von klassischen Globus-Jobs (in GRAM RSL) über Grid Services bis hin zu nach Parametern automatisch generierten Jobs auf die vorhandene Grid-Implementierung abzubilden (also hier das Globus Toolkit).
Diese Grid Architektur ist auch als Open-Source Projekt unter dem Namen eXeGrid [46] veröffentlicht.
Auf dieser Infrastruktur basiert das von dem Bundesministerium für Bildung und Forschung unterstützte I-Lab Forschungsprojekt [84].
Neben der Weiterentwicklung der obigen Middleware-Komponenten ist die Zielsetzung eine Problemlösungsumgebung zu schaffen. Dies ist Umsetzung der Idee, dass zur Problemlösung bedarfsweise Ressurcen angefordert (bzw. angemietet) werden können. Darüber hinaus soll das Problem bedienungsfreundlich über eine Internetseite formuliert werden können (mit entsprechender Benutzerauthentfikation) und in der Problemlösungsumgebung die Aufgabe mittels einer Komponentendatenbank in einen Grid-Job umgesetzt werden. Dieser wird dann an das Fraunhofer Resource Grid weitergegeben und die Ergebnisse über die Internetseite dem Benutzer (bzw. Kunden)

zugänglich gemacht.

Diese Infrastruktur der on-demand Nutzung von Ressourcen könnte in naher Zukunft zu einer Etablierung von speziellen Ressourcen-Providern führen und ist daher sehr interessant.

2. Grid-Middleware

2.1 Grid-Middleware im Allgemeinen

Wie im Abschnitt 1.2 einleitend besprochen wurde, ist ein Grid ein verteiltes System, welches durch die Zusammenbindung vernetzter Komponenten mit einer speziellen Middleware das Grid formt. Ergänzend sollen hier noch die wichtigsten Aufgaben der Grid-Middleware vorgestellt werden

Tatsächlich steht bei Grids die Anpassung an das jeweilige Betriebssystem und dessen Komponenten im Vordergrund, denn Hardwareabhängigkeit besteht im Normalfall nur beim Maschinencode und eventuell bei der Erfassung von Hardware-Informationen – durchgeführt beispielsweise von den Information Services beim Globus Toolkit (siehe Abschnitt 2.3.1.3). Um das Grid selbst zu bilden, muss die Grid-Middleware natürlich die gesamten verfügbaren Ressourcen des verteilten Systems verwalten. Dabei ist eine Nutzerverwaltung samt entsprechender Zugriffskontrolle für die Ressourcen unerlässlich, insbesondere im Hinblick auf den Themenkomplex der virtuellen Organisationen. Wenn die entsprechenden Jobs über die Middleware im Grid gestartet werden, müssen diese bis zu ihrer Terminierung überwacht sowie für das Scheduling und die Zuteilung der Ressourcen gesorgt werden. Letzteres stellt insbesondere bei der Dynamik des Grids hohe Anforderungen an die Kommunikations- und Synchronisationsmechanismen der Middleware. Neben einer möglichen Kostenabrechnung des Ressourcenverbrauchs muss auch die Infrastruktur sowohl für die Sicherheit als auch für die Überwachung der Performance und der auftretenden Fehler geschaffen werden.

Eine wesentliche Aufgabe der Grid-Middleware ist es also, den Zugriff auf das Grid für die Applikationen möglichst transparent zu gestalten.

2.2 Beispiele für Grid-Middleware

2.2.1 Condor

Das Condor Projekt [158] wird an der Universität von Wisconsin-Madison seit über fünfzehn Jahren vorangetrieben und ist erst seit Anfang 2003 als Open-Source freigegeben. Der Grundgedanke dieses Projektes ist es, auf verteilten Ressourcen einen hohen Verarbeitungsdurchsatz zu erreichen, also sogenanntes *High Throughput Computing* (HTC).
Condor ist im Kern ein klassisches Batch-System, das neben den erwarteten Aufgaben wie Auftragsverkettung, gesteuertes Scheduling, Prioritätsschemata, Ressourceüberwachung und Ressourcemanagement einige Erweiterungen und Eigenheiten bietet.
Zum Beispiel kann Condor einerseits einen ganzen virtuellen Cluster von dedizierten Computerknoten verwalten (ein sogenannter *Condor-Pool*), aber andererseits auch eine nicht ausgelastete Workstation zeitweise okkupieren, welche bei Belastung wieder – transparent für die Anwendung – geräumt wird.
Eine Besonderheit der Ressourcenvergabe ist die Möglichkeit, dass Jobs minimale und empfohlene Ressourcenanforderungen angeben können, aber auch Maschinen entsprechende Anforderungsprofile an die Jobs stellen können. Dies ist wichtig, da so keine zentrale Administration der Ressourcen vorliegt.
Letzteres ist bekanntlich eine Voraussetzung für ein Grid und mittels Condor-G [159], dem spezialisierten entkoppelten Jobmanagement von Condor, ist es möglich, den Condor-Pool um die Ressourcen eines Globus-Grids zu erweitern, aber nicht umgekehrt. Man sieht der Condor-Middleware noch an einigen Stellen an, dass sie ursprünglich zur reinen Cluster-Bildung entwickelt wurde, aber durch die manigfaltigen Erweiterungen ist sie eine gute Grundlage für Computational Grids.

2.2.2 Harness

HARNESS (Heterogenous Adaptable Reconfigurable NEtworked SystemS) stellt – vereinfacht ausgedrückt – eine dynamisch anpassbare Umgebung für parallele Programme dar [83]. Dies wird erreicht, indem sämtliche eingebundenen Komponenten (wie verschiedene Computer) als ein einziger virtueller Computer mit großem verteilten Speicher dargestellt werden. Als Programmierer fokussiert man also nur auf diese sogenannte *Distributed Virtual Machine* (DVM) und nicht auf eigentlich existierende Strukturen. Deshalb nennt man HARNESS auch ein *Metacomputing System*. Es können mehrere Benutzer die gleiche DVM benutzen, oder auch mehrere DVM auf der gleichen Hardware laufen haben. Daher beinhaltet HARNESS auch die Möglichkeit DVMs zu vereinigen bzw. zu teilen.
Die eigentliche Stärke der Virtualisierung besteht darin, dass man selbst in der Laufzeit real existierende Komponenten austauschen oder erweitern kann, da solche Vorgänge von der Middleware durch das Konzept problemlos transparent gehalten werden können. Auch ist die Interoperabilität leicht zu gewährleisten, da nur für die virtuelle Schnittstelle programmiert wird (analog zur Java Virtual Machine).
Die Besonderheit von HARNESS liegt in der Umsetzung. Die Virtual Machine wird durch sogenannte *Deamons* gebildet, welche für entsprechende Funktionalitäten oder Komponenten verantwortlich sind, also die eigentliche Schnittstelle zum realen System bilden. Durch Hinzufügen oder auch Austauschen dieser Deamons erlangt das System seine hohe Dynamik. Mit diesem Design ergibt sich auch die Möglichkeit Deamons, welche nicht gebraucht – oder vorerst nicht benutzt – werden, gar nicht erst zu laden und so einen minimalen Verwaltungsaufwand zu erreichen ohne den möglichen Funktionsumfang einzuschränken.
Die Programmierung der DVM geschieht im Allgemeinen durch entsprechend angepasste PVM- und MPI-Umgebungen (siehe Kapitel 3). Letztlich qualifiziert dies die Middleware insbesondere für Computational Grids.

2.2.3 Legion

Legion ist ein Softwareprojekt an der Universität von Virginia [114]. Das Projekt läuft seit Ende 1993 und ist als offenes Projekt geplant, um Drittentwickler zu ermutigen, Applikationen und einzelne Komponenten selbst zu entwickeln oder bestehende zu verfeinern. Das erste lauffähige System wurde 1997 vorgestellt.
Das Charakteristische an Legion ist seine objektbasierende Struktur. Praktisch alles wird durch Objekte repräsentiert, welche miteinander kommunizieren können:

- Host-Objekte, die Prozessoren repräsentieren
- Vault-Objekte, welche Speicher darstellen
- Context-Objekte, die Kontext-Namen – also menschenlesbare Bezeichner – in sogenannte Legion Object Identifier (LOID) umsetzen, welche die intern benutzten Bezeichner darstellen
- Binding-Agenten, welche wiederum die nur lokal gültigen LOIDs in eine Legion Object Address (LOA) umwandeln, um z.B. zwischen Objekten global zu kommunizieren
- Implementation-Objekte, die den auszuführenden Binärcode enthalten

Ein Programm-Objekt würde also sowohl ein Context- als auch ein Binding-Objekt haben und die ihm über seine Klasse zugeordneten Vault- (Speicher), Implementation- (Binärcode) und Host-Objekte (Prozessoren).
Die Programmierung erfolgt naturgemäß nur in objektorientierten Sprachen, da nur diese die objektbasierende Architektur von Legion adäquat umsetzen können. Dabei setzt Legion die Objekte immer für die jeweilige Sprache entsprechend um.
Das Legion-Projekt spezifiziert die Funktionalität und die Schnittstellen für das jeweilige Objekt, nicht aber das Kommunikationsprotokoll oder die Programmiersprache. Legion liefert Implementierungen aller Standard-Objekte, aber der Benutzer kann auch eigene Objekte implementieren, wenn er andere Performance-, Funktionalitäts- oder Sicherheitsbedürfnisse hat.
Der vielseitige Ansatz von Legion wurde hauptsächlich für paralleles Rechnen mit bekannten Bibliotheken (PVM und MPI – siehe Kapitel 3) und parallelen Programmiersprachen wie MPL (Mentat Programming Language – C++ basierend) verwendet. Daher ergibt sich auch hier das primäre Einsatzgebiet als Computational Grid.

Seit Mitte 2001 ist es um das Legion-Projekt sehr ruhig geworden und es wird daher wohl kaum noch weiterentwickelt [168].

2.2.4 Unicore

Das UNICORE (UNiform Interface to COmputing REsources) Projekt [171,172], welches im Jahre 2000 erstmals zur vollständigen Einsatzfähigkeit gereift ist, sieht sich als Standardisierung von Batch-Zugriffen über das Internet auf Computer und assoziierte Ressourcen.
Unterteilt wird die UNICORE Architektur in drei Aufgabengebiete:

- Mit einem UNICORE Client, welcher dank Java-2 auf praktisch jedem Rechner im Internet lauffähig ist, kann ein Benutzer einzelne oder verkettete Aufträge erstellen, abschicken und kontrollieren.
- Der Client nimmt Kontakt zu einem UNICORE Gateway auf, welcher sowohl den Client als auch den Benutzer authentifiziert. Normalerweise steht hinter einem Gateway und dessen Firewall ein entsprechender Pool an Ressourcen. Sie übernehmen auch das Routing zu anderen Gateways, falls nötig.
- Die Aufträge erreichen dann einen für den angesprochenen Pool zuständigen UNICORE Server, welcher die abstrakten Aufgaben des Clients auf das lokale native System abbildet. Dabei kümmern sie sich um alle nötigen Datenübertragungen sowie deren Synchronisation und schicken Informationen wie Statusmeldungen und die Ausgabe der Jobs an den Client zurück.

Ein Auftrag in UNICORE besteht aus einer Anzahl von Tasks, welche in vordefinierten Beziehungen zueinander stehen. Als mögliche Tasks stehen Verarbeitung von Skripten, Übersetzen, Linken und Ausführung sowie Datentransfer zur Verfügung. Tasks können so gruppiert werden, dass eine hierarchische Auftragsstruktur entsteht – sie ermöglichen damit verschiedene Stufen der Ausführung auf verschiedenen Systemen im UNICORE Grid. Für jeden Auftrag spezifiziert der Benutzer das Zielsystem und die Ressourcenanforderung der Task. Wenn der Client diese mit den Ressourcen des Zielsystems vergleicht und zu einem positiven Ergebnis kommt, wird der Auftrag ausgeführt.

UNICORE wird kontinuierlich weiterentwickelt und beständig an die neuesten Grid-Entwicklungen angepasst.

2.3 Das Globus Toolkit

Das *Globus Projekt* [162] ist eine Gemeinschaftsarbeit des Argonne National Laboratory, dem Information Sciences Institute der Universität von Süd-Kalifornien und der Universität von Chicago. Im Rahmen dieses Projektes wird Forschung und Entwicklung für grundlegende Techniken betrieben um ein Grid zu erschaffen, welches gemeinsame Nutzung von Rechenkapazität, Datenbanken und andere Online-Werkzeuge sicher über Firmen-, Instituts- und geographische Grenzen ermöglicht und weiterhin die Autonomie der Teilnehmer gewährleistet.
Ein fundamentaler Grundsatz des Projektes ist es, sämtliche Software als Open-Source zu veröffentlichen, um die Forschung und Entwicklung anderer Organisationen und Firmen zu ermutigen. Daher beruhen einige kommerzielle Produkte von IT-Firmen auf Globus-Technologie.
Das Kernstück des Projektes bildet das Globus Toolkit, welches praktisch die Bausteine liefert, aus denen ein Grid aufgebaut ist. Als solches beinhaltet es natürlich Softwaredienste und Bibliotheken zur Überwachung, Entdeckung und Verwaltung von Ressourcen sowie Sicherheits- und Datenmanagement.
Erstmals wurde das Globus Toolkit mit der Version 2 weltweit erfolgreich eingesetzt und avancierte damit quasi zum Standard der Grid-Middleware. Diese auch heute noch bei länger bestehenden Grids anzutreffende Version wird ausführlich im Abschnitt 2.3.1 vorgestellt. Eine hauptsächlich um die Implementierung der OGSA (Abschnitt 1.2.1) erweiterte Version 3 des Toolkits (Abschnitt 2.3.2) fand durch die Abwärtskompatibilität fast denselben Anklang wie der Vorgänger. Die erst seit kurzem erhältliche Version 4 ist in der Standardinstallation nicht mehr abwärtskompatibel, was auf eine umfassende Änderung der Komponenten zurückzuführen ist. Der Einsatz dieser aktuellen Version konnte aufgrund der erst wenige Wochen alten Veröffentlichung nicht beobachtet werden.

2.3.1 Der Aufbau des Globus Toolkit 2

Wir befassen uns hier mit der Toolkit Version 2.4, was der letzten Version des Globus Toolkits 2 entspricht (IBM benutzte in ihrer Grid Toolbox die Version 2.2).
Die hier vorgestellten Komponenten sind natürlich in ihrer erweiterten Form auch in den Versionen 3.0 und 3.2 des Toolkits enthalten, so dass sich bei diesen vom Konzept her nicht viel ändert.

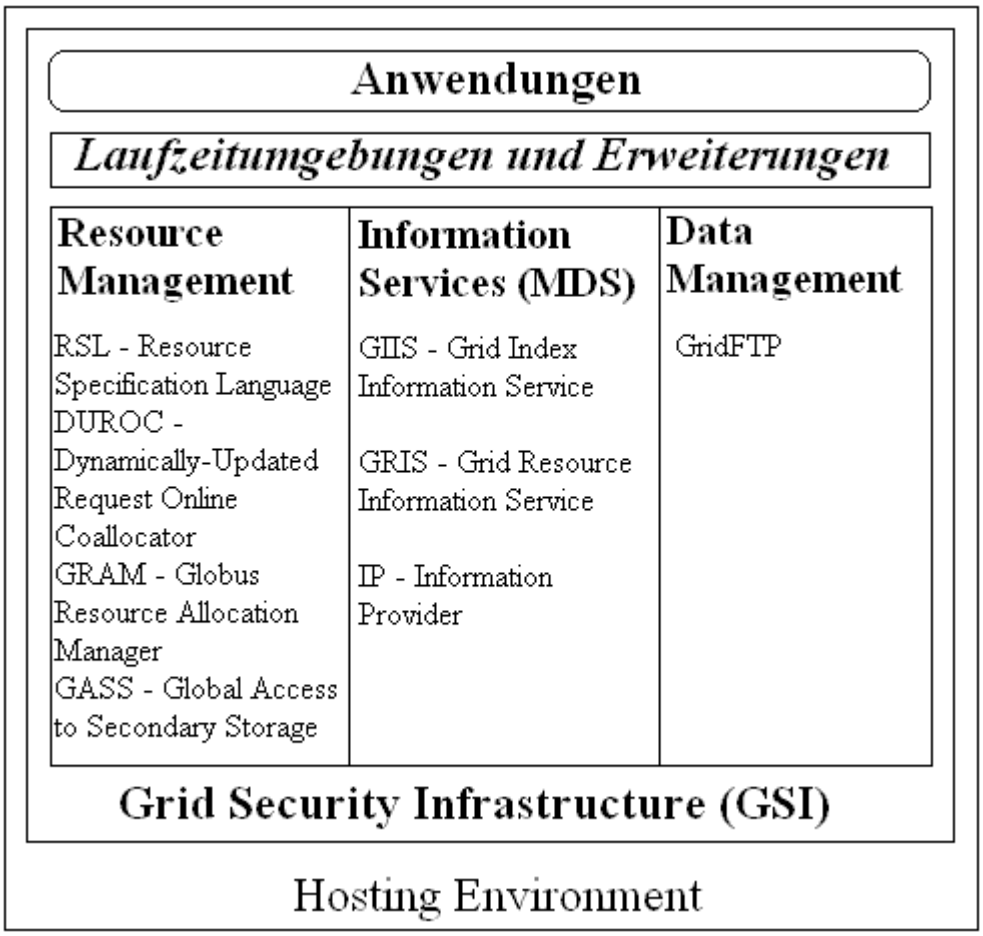


Abbildung: Das Globus Toolkit 2

Das Globus Toolkit ist in drei logische Aufgabenbereiche aufgeteilt, welche die grundlegenden Bestandteile zum Betrieb eines Grids zur Verfügung stellen. Als Erstes ist das *Resource Management* zu nennen, welches die Belegung der vom Grid zur Verfügung gestellten Ressourcen übernimmt. Als Voraussetzung dazu stellen die *Information Services* die genauen Informationen über die vorhandenen (bzw. freigegebenen) Ressourcen des Grids zusammen. Unterstützend übernimmt das *Data Management* die

Aufgabe der Verwaltung und Zugriffssteuerung aller Daten in der Gridumgebung. Eingebettet sind diese Komponenten in der so genannten Grid Security Infrastructure, welche eine sichere Kommunikationsumgebung zur Verfügung stellt.

2.3.1.1 Grid Security Infrastructure (GSI)

Die *Grid Security Infrastructure* [57] wird im Globus Toolkit benutzt, um in einem prinzipiell offenen Netzwerk (wie z. B. dem Internet) Authentifizierung und sichere Kommunikation zwischen den einzelnen Grid-Komponenten zu gewährleisten. Dazu kommt noch die Umsetzung der gridbedingten Anforderung, dass es kein zentral gesteuertes Sicherheitssystem sein darf. Dies wird erreicht, indem zwar auf Grid-Ebene jeder User nach den Regeln einer Public Key Infrastructure (PKI) eine eigene Identität aufweist, jedoch diese auf allen Rechnern, die der User nutzen will, einem lokalen Benutzer zugewiesen sein muss. Er agiert dann auf dem konkreten Rechner mit den Rechten des entsprechenden Benutzers. Normalerweise wird auf den beteiligten Computern ein generischer Benutzer angelegt (z.B. `globus`), auf den dann alle erwünschten Grid-Benutzer gemappt werden.
Zu den wichtigsten Merkmalen zählen die Möglichkeit zur gemeinsamen und gleichzeitigen Authentifizierung im Grid sowie das so genannte *Single Sign-On* (englisch für einmaliges Anmelden). Letzteres wird dadurch erreicht, dass ein User einmalig einen Proxy startet und sich ihm gegenüber authentifiziert. Alle aufkommenden Aufforderungen zur Identitätsverifikation beantwortet dann der Proxy für den User.

2.3.1.2 Das Resource Management

Das Resource Management wird aus folgenden Einzelkomponenten gebildet:

- Resource Specification Language (RSL)
  Die Kommunikation der beteiligten Komponenten im Resource Management untereinander wird mittels beschreibender Skripten in der standardisierten Resource

Specification Language durchgeführt [59]. Insbesondere ist es als Benutzer möglich, jede einzelne Komponente des Ressourcemanagement über diese Schnittstelle direkt anzusprechen. Mit den Weiterentwicklungen und Erweiterungen des Managements (Globus Toolkit 3 und 4) wächst diese Sprache natürlich auch mit, aber die grundlegende Syntax ändert sich nicht. Ein Beispiel dazu findet sich im Kapitel 4 beim Starten des Testprogramms `hello`.

- Globus Resource Allocation Manager (GRAM)
  Der Globus Resource Allocation Manager [56] bildet die unterste Schicht in der Ressourcenverwaltung des Globus Toolkits. Er nimmt mittels eines Gatekeepers auf einem konkreten System Ausführungsanfragen von Benutzern entgegen. Diese werden dann unter Berücksichtigung der im RSL-Skript gemachten Angaben an die entsprechenden Jobmanager zur Ausführung weitergeleitet. Darin sind insbesondere die Auswahl der Ressourcen, die Prozessgenerierung sowie die Art der Kommunikation des Auftrags festgelegt. Daher werden auch die Möglichkeiten zur Überwachung und Verwaltung des Auftrags zur Verfügung gestellt.
- Dynamically-Updated Request Online Coallocator (DUROC)
  Eine Schicht höher in der Verwaltungsebene liegt der Dynamically-Updated Request Online Coallocator [54]. Über ihn können Anfragen und Aufträge an mehrere Resource Manager gleichzeitig erledigt werden. Diese Schnittstelle übernimmt also die Aufgabe eines Stellwerkes, welches den Kommunikationsfluss der Benutzer zu den entsprechenden Managern regelt.
- Global Access to Secondary Storage (GASS)
  Ergänzend zum Resource Manager hat das Global Access to Secondary Storage [55] die Aufgabe ausführungsrelevante Dateien vom Client anzufordern, bevor der Auftrag ausgeführt wird, also ein konsistentes Auftragspaket zu gewährleisten. Außerdem wird über ihn die Ausgabe der ausgeführten Aufträge an den Client abgewickelt.

2.3.1.3 Die Information Services

Die Ausdrücke "Information Services" und "MDS" werden häufig synonym verwendet, obwohl dies vom theoretischen Konzept her nicht ganz korrekt ist, denn es könnten noch

weitere Komponenten zu den Information Services hinzukommen.
Unter dem Begriff Monitoring and Discovery Service [58] versteht sich ein Satz von Informationsdiensten, welche die Aufgabe haben, in einer Grid-Umgebung Informationen von lokalen Diensten über Ressourcen zusammenzufassen und diese im gesamten Grid zugänglich zu machen. Letzteres geschieht mittels des freien LDAP-Standards, so dass dazu auf vielerlei vorhandene Programme und Schnittstellen-Bibliotheken zurückgegriffen werden kann.
Dem MDS sind folgende Komponenten angegliedert [52]:

- Information Provider (IP)
  Die Information Provider liefern spezielle Daten einer Ressource, wie z. B. die Auslastung von Prozessoren oder die Daten eines Messgerätes – aber auch der Status eines separaten Clusters ist möglich. Eine Ressource hat also einen oder mehrere Information Provider.
- Grid Resource Information Service (GRIS)
  Der Grid Resource Information Service, manchmal auch treffend "lokaler Server" genannt, sammelt die gesamten Informationen der Information Provider einer (virtuellen) Ressource. Dabei werden die Information Provider sequentiell abgefragt. Normalerweise werden die gesammelten Daten an einen (oder mehrere) GIIS (nachfolgende Komponente) mittels dem Grid Resource Registration Protocol (GRRP) weitergeleitet.
- Grid Index Information Service (GIIS)
  Der Grid Index Information Service, auch "Sammelserver" genannt, ist die nächst höhere Hierarchieebene im Hinblick auf den Grid Resource Information Service. Auch wenn ein hierarchisch aufgebautes System von mehreren GIIS möglich ist (für virtuelle Organisationen praktisch unverzichtbar), so gibt es doch nur ein als LDAP-Verzeichnis realisiertes GIIS, welches die Informationen von allen im Grid vorhandenen Ressourcen enthält. Um über den Zustand des Grids informiert zu werden, muss man dann nur noch die Einträge im obersten GIIS durchgehen.

2.3.1.4 Das Data Management

Das Data Management ist im Globus Toolkit 2 praktisch identisch mit GridFTP.
Um Dateien im Grid zu verschieben, wird eine auf dem bekannten File Transfer Protokoll (FTP) basierende Entwicklung verwendet – GridFTP [53]. Zur Gewährleistung der Sicherheit wird die Grid Security Infrastructure (GSI) bei den Kontroll- und Datenkanälen benutzt.
Eigenschaften wie die Nutzung mehrerer Datenkanäle zur parallelen Übertragung, die Entgegennahme mehrerer Befehle zur sukzessiven Abarbeitung und die von außen gesteuerte Server-zu-Server Übertragung ermöglichen einen reibungslosen integrierten Datenverkehr im Globus Toolkit.
Darüber hinaus gibt es noch spezielle Möglichkeiten zum Aufbau eines Data Grids, allerdings sind diese Komponenten nicht Teil der Standardpakete [163].

2.3.2 Implementierung der OGSA im Globus Toolkit 3

Im vorangegangenem Kapitel wurde einführend auf die grundlegenden Ideen der Open Grid Service Architecture (OGSA) eingegangen, welche sich auf die Veröffentlichung [171] stützen. In diesem Abschnitt werden wir uns mit ihrer Umsetzung im Globus Toolkit befassen, genauer gesagt mit der Implementierung in der Toolkit Version 3.0 bzw. 3.2 [61].
Ein grundlegender Unterschied zu den älteren Versionen des Globus Toolkits sind die Voraussetzungen für die Software-Infrastruktur des Servers. Wurde vorher praktisch nur die Existenz eines Perl-Interpreters zum Ausführen von Skripten gefordert (siehe Kapitel 4), so wird nun die Installation eines Webservers ebenso vorausgesetzt wie die Erweiterung um eine *Web Service Engine*, welche Nachrichtenaustausch auf XML-Basis ermöglicht.
Die Änderungen an der klassischen Globus-API zu der Version 2.4 sind nur minimal (unter anderem ist als Ergänzung zu GridFTP der RLS – Replica Location Service – für Data Grids hinzugekommen); das Augenmerk wurde hauptsächlich auf eine neue Schnittstelle gelegt: die Open Grid Service Architecture. Um diese umzusetzen wurde die

vom Global Grid Forum speziell entwickelte Open Grid Services Infrastructure (OGSI) [125] in der Version 1.0 implementiert. Die OGSI stellt sozusagen die praktische Umsetzung der OGSA dar, indem konkrete Schnittstellen bzw. elementare Web Services formuliert werden, auf welchen *Grid Services* [80] (spezialisierte Web Services) basieren. Die Besonderheit ist, dass die elementaren, in der OGSI spezifizierten Web Services als Grundlage zum Komponieren neuer Grid Services benutzt und dann selbst wieder ein Bestandteil anderer Grid Services werden können. Durch die Standardisierung der notwendigsten Services können erstellte Grid Services auf allen Grids benutzt werden, welche die Open Grid Services Architecture implementieren.
Grid Services werden in zwei Arten unterteilt: in *Grid Service Descriptions* und *Grid Service Implementations*. Erstere beschreiben das Interface und die Protokolle einer bestimmten Ressource (z. B. um Instanzen mit bestimmten Eigenschaften zu suchen), während letztere die virtuelle Realisierung der Ressource sind. Einer Grid Service Description können also mehrere Grid Service Instances entsprechen.
Ein *Grid Service Container*, also eine abgeschlossene Grid Service Umgebung eines Servers, besteht aus der *Web Service Engine* und der darauf aufsetzenden *OGSI Specification Implementation*. Die OGSI Specification Implementation übernimmt die technische Umsetzung der in der OGSI formulierten Schnittstellen. Ergänzend sichert die *Security Infrastructure* die Nachrichtenebene ab, überprüft die Zugriffsberechtigung und führt die Authentifikation durch.
Weiterhin gibt es sehr spezielle OGSI-kompatible Grid Services, welche sich mit der untersten System-Ebene befassen, den *System Level Services*. Diese bestehen momentan aus drei Services:

- dem *Ping Service*, welcher zur Überprüfung der Erreichbarkeit anderer Platformen dient
- dem *Management Service* zur Überwachung der Auslastung des Grid Service Containers und dessen Status im Allgemeinen sowie des Herunterfahrens desselben.
- dem *Logging Management Service* für die Konfiguration sämtlicher Protokollierer während der Laufzeit.

In einem solchen Grid Service Container können nun benutzerdefinierte OGSI-basierende Grid Services existieren. Es gibt allerdings noch die *Base Services*, welche selbst normale Grid Services sind (ab Toolkit Version 3.2) und Teile der Middleware-API umsetzen:

- *WS GRAM* ist ein Service zur Programmausführung und damit letztlich eine Grid Service Schnittstelle für GRAM.
- *Reliable File Transfer* (RFT) ist – wie der Name schon sagt – eine Grid Service Schnittstelle für GridFTP.

Abschließend ist zu sagen, dass durch die Verwendung von OGSI im Globus Toolkit ein allgemeiner Standard offen gelegt wurde (die Implementierung ist Open-Source), was somit die Möglichkeit einer Umsetzung der Open Grid Services Architecture beweist. Natürlich ist diese erst von realer Bedeutung, wenn andere Grids die OGSA ebenfalls adaptieren.

2.3.3 Das Globus Toolkit 4 im Überblick

Da kurz vor der Fertigstellung dieser Arbeit das Globus Toolkit Version 4.0 endgültig veröffentlicht wurde, soll hier ein Überblick des Aufbaus und eine kurze Vorstellung der einzelnen Komponenten (analog zum Abschnitt 2.3.1) gegeben werden. Leider war die Online-Dokumentation noch nicht vollständig und fehlerfrei, aber die Konzepte können entsprechend wiedergegeben werden [62].
Generell kann man sagen, dass – stärker noch als in der Version 3 – fast jede Funktion sowohl eine Umsetzung als Grid Service als auch eine proprietäre API (häufig die gleiche wie in der Version 2) hat. Die entsprechenden Grid Services bieten dabei im Allgemeinen einen größeren Funktionsumfang – ein Umstand der vornehmlich deren Aktualität zuzuschreiben ist.

2.3.3.1 Grid Security Infrastructure

Die Aufgabe der GSI hat sich im Toolkit 4 nicht verändert [63].
Sicherheitsmechanismen für Web Services umfassen Vertraulichkeit für Nachrichten und Datentransport zwischen Web Services.
Es werden zusätzlich Wekzeuge wie eine Certification Authority (CA) zur Erstellung eigener X.509 Zertifikate und eine GSI-basierende ssh-Shell mitgeliefert.

2.3.3.2 Execution Management

Der Bereich des Resource Management der älteren Versionen des Toolkits wurde treffender in Execution Management umbenannt [64].
Neben dem schon bekannten Resource Management aus dem Globus Toolkit 2 ist auch eine neuere Version des WS-GRAM (bekannt aus der Version 3.2) als Abbildung der API auf Grid Services integriert. Letzteres greift zur benötigten Dateiübertragung vor (und nach) der Job-Ausführung selbstständig auf den Reliable File Transfer (RFT) Dienst zurück und gewinnt durch diese Integration an Effektivität.

2.3.3.3 Information Services (Monitoring and Discovery System)

Der größte Unterschied zu den Vorgängerversionen liegt im *Monitoring and Discovery System* (MDS4) [65]. Dieses ist nicht abwärtskompatibel, allerdings soll es möglich sein, dass MDS2 des Globus Toolkit 2 nachträglich zu installieren. Der Grund für diese Inkompatibilität ist die Abkehr von den LDAP-Verzeichnissen zu dem *Xpath*-Protokoll. Dabei hat MDS4 nicht nur Web Service- und Kommandozeilen-Schnittstellen, sondern bietet auch ein komfortables Browser-Interface (WebMDS).
Die nötigen Daten werden von sogenannten *Information Sources* (wie beispielsweise Dateien, Programme oder Web Services) an drei verschiedene Sammel-Dienste weitergeleitet, welche abgefragt werden können:

- *MDS-Index*, welcher Xpath-Abfragen der zuletzt von den Information Sources gelesenen Daten erlaubt
- *MDS-Trigger*, der bei benutzerspezifizierten Zuständen der Information Sources vordefinierbare Aktionen auslöst (wie das Senden einer Email oder das Verfassen eines Eintrages in eine Log-Datei)
- *MDS-Archiver*, über den mittels Xpath-Abfragen auf historische Daten der Information Services zugegriffen werden kann

Dabei sind die bei den Sammel-Diensten registrierten dynamischen Information Sources mit einem Verfallsdatum versehen, was diese nach Ablauf desselben aus der Überwachung entläßt. Damit wird sichergestellt, dass nur Daten über möglichst aktuelle

Ressourcen verfügbar sind – also keine Dateileichen existieren. Die Information Sources können sich jederzeit bei den Sammel-Diensten registrieren, was eine Erneuerung des Verfallsdatums bedeutet.
Jeder Web Service Container beinhaltet standardmäßig einen MDS-Index, bei dem jeder Grid Service des Containers automatisch registriert wird. Darüber hinaus können beliebige MDS-Indices als Grid-weite MDS-Index definiert werden. Dann werden zusätzlich die Daten aller MDS-Indices an diese weitergeleitet.

2.3.3.4 Data Management

Die Aufgabe des Data Management hat sich seit dem Toolkit 2 nicht geändert und so sind nicht nur die Schnittstellen des alten Toolkits erhalten geblieben (GridFTP) [66], sondern auch die Erweiterungen von Version 3.0 (RLS) und 3.2 (RFT – die Grid Service Schnittstelle von GridFTP).
Als experimentelle Versionen sind im Toolkit Release 4.0 noch der Data Replication Service (DRS) [67] – zum Suchen und lokalen Replizieren von Dateien im Grid – und OGSA-DAI [68], eine Grid-Erweiterung für den Zugriff und die Integration von Dateien sowie relationalen und XML-Datenbanken. Diese sind aber nur bedingt nutzbar, da sie kaum ausgetestet sind und in weiteren Versionen des Toolkits noch stärkere Veränderungen erfahren werden.

3. Parallele Programmierung für Grids

Als paralleles Rechnen bezeichnet man die Vorgehensweise ein Problem gleichzeitig auf mehreren Prozessoren (parallel) zu bearbeiten. Dabei wird das Ursprungsproblem in Teilprobleme zerlegt und mit diesen von den einzelnen Prozessoren in gemeinsamer Kooperation das Urprungsproblem gelöst.
In diesem Kapitel wird die später in der Arbeit verwendete MPI-Bibliothek vorgestellt, welche insbesondere das Aufteilen der Teilaufgaben auf die Prozessoren und deren Kommunikation untereinander unterstützt.

3.1 Das Message-Passing Programmiermodell

Beim Message-Passing Programmiermodell (vgl. Kapitel 4 von [130]) wird ein Parallelrechner mit verteiltem Speicher zugrunde gelegt, wobei im Allgemeinen keine Besonderheiten der Topologie des Rechners genutzt werden, was für die Portabilität der Programme wichtig ist. Ein Programm besteht nach diesem Modell aus mehreren Prozessen und den ihnen zugeordneten lokalen Daten. Die Prozesse können auf ihre Daten uneingeschränkt zugreifen und diese mittels Nachrichten mit anderen Prozessen austauschen. Nach dem Modell kann jeder Prozess ein eigenes Programm ausführen, in der Praxis wird jedem Prozess allerdings meist das gleiche Programm zum Ausführen übermittelt, wobei einige Bereiche des Programmtextes nur von einigen Prozessen bearbeitet werden.
Die einzige Möglichkeit für die Prozesse, eine gemeinsame Aufgabe auszuführen, ist es, miteinander in Kontakt zu treten, was, wie der Name schon vermuten lässt, mittels Nachrichten geschieht. Dabei muss jede Kommunikationsoperation im Programmtext explizit angegeben sein. Möglich ist dabei nicht nur Prozess-zu-Prozess Kommunikation, sondern auch Kommunikation zu einer beliebigen Untermenge der Prozesse.
Eine klassische Programmiermethode ist es, einen Prozess auszuzeichnen und diesen alle relevanten Daten an die restlichen Prozesse verteilen zu lassen. Nach Beendigung der Rechnungen sammelt dieser Prozess dann die Teilergebnisse und fügt sie zu einem Gesamtergebnis zusammen.

Bekannte Vertreter des Message-Passing Modells sind PVM (Parallel Virtual Machine) [128] und MPI (Message Passing Interface) [120]. Auf Letzteres werden wir im Folgenden näher eingehen.

3.2 Message Passing Interface (MPI)

MPI ist ein Standard für Message-Passing Bibliotheken, welcher Programmen genormte Schnittstellen für Kommunikationsoperationen – wie die Prozess-zu-Prozess Kommunikation – zur Verfügung stellt. Die Standardisierung wird von Firmen, Forschungseinrichtungen und Universitäten getragen, diee als allgemeine Anlaufstelle das MPI-Forum gegründet haben.
Die erste Version von MPI wurde 1994 veröffentlicht. Etwas später folgte 1997 mit MPI-2 eine enorm umfangreiche Erweiterung. In FORTRAN-77 und C (bzw. C++) werden die MPI-Bibliotheken bisher am häufigsten genutzt. Darüber hinaus gibt es Implementierungen von MPI-2 auch für Java.
Der MPI-Standard legt die Syntax der Anweisungen und die Semantik der einzelnen Operationen fest. Dadurch ist gewährleistet, dass ein und derselbe Programmtext auf verschiedenen Systemen ohne Änderung lauffähig ist. Allerdings ist es nicht möglich, gleich bleibende Performance zu garantieren, da die Implementierungen der Bibliothek für verschiedene Hardwareplattformen intern unterschiedlich realisiert sein können. Auch ist zu beachten, dass MPI keinerlei Regelungen über beispielsweise den Start von Prozessen oder die Zuordnung von Prozessen zu Prozessoren definiert.
Bekannte frei verfügbare MPI-Implementierungen sind unter anderem MPICH [121] vom Argonne National Lab und der Mississipi State Universität, LAM (Local Area Multicomputer) [116] vom Ohio Supercomputing Center und CHIMP (Common High-Level Interface to Message Passing) [22] vom Edinburgh Parallel Computing Center.

3.3 MPI für das Globus Toolkit

Da zum Ausführen von MPI-Programmen diese zu den entsprechenden Computern bzw. Prozessoren transferiert werden müssen, wird auf bestehende Infrastrukturen zurückgegriffen. Auf Mehrprozessorrechnern geschieht dies normalerweise mittels Betriebssystemfunktionen und bei einem Linux-Cluster wird im Allgemeinen ein MPI-Programm durch Remote-Shell gestartet. Dabei müssen natürlich vor allem bestehende Sicherheitsstrukturen – wie vorhandene Benutzerkonten und die Authentifizierung mit Passwort – beachtet werden. In einem Grid, das eigenständige Mechanismen zur Authentifizierung und Datenübermittlung in der Middleware mit sich bringt, ist zumindest eine Integration der genutzten Benutzerauthentifikation und der ferngesteuerten Programmausführung nötig. Weitergehende Anpassungen, wie die Berücksichtigung der Topologie, erhöhen natürlich die Performance und Integration, sind aber nicht unbedingt notwendig.

Für das Globus Toolkit 2 existiert eine MPI-Anpassung mit dem Namen MPICH-G2 [169], welche im September 2000 erschienen ist und kontinuierlich weiterentwickelt wird. Wie der Name schon vermuten lässt, basiert MPICH-G2 auf den weiter oben erwähnten MPICH-Bibliotheken. Die hier verwendete Version von MPICH-G2 unterstützt den kompletten MPI 1.2 Standard mit einigen wenigen Erweiterungen von MPI-2. Im Laufe des Jahres 2005 ist die neue Version MPICH-G3 erschienen, welche insbesondere den vollen MPI-2 Standard implementiert und an das Globus Toolkit 3 angepasst ist.

Im Folgenden sollen mehrere spezielle Eigenschaften von MPICH-G2 erläutert werden, die nicht direkt mit der MPI-Funktionalität zu tun haben, sondern vornehmlich Anpassungen an das Globus Toolkit 2 sind:

- Die Aufgabe von MPICH-G2 in einem Grid ist es natürlich, die Möglichkeit zu schaffen, mehrere – potentiell mit verschiedener Architektur ausgestattete – Rechner zwecks Ausführung von MPI-Programmen zuammenzuschalten. Dafür werden Daten automatisch in Nachrichten konvertiert, um sie zwischen Rechnern mit verschiedenen Architekturen auszutauschen zu können. Durch die Möglichkeit, verschiedene Protokolle zur Kommunikation zu verwenden, kann TCP zur Übertragung zwischen verschiedenen Rechnern und – wenn verfügbar – zwischen herstellerspezifischen MPI-Routinen, welche mit MPICH kompatibel sein müssen, für die innere Kommunikation

in einem Rechner benutzt werden. Klassische Beispiele für herstellerspezifische MPI-Implementierungen sind Sillicon Graphics MPT oder IBMs PSSP.

- Wenn große Datenpakete zwischen MPI Prozessen mittels des TCP-Protokolls ausgetauscht werden, gibt es die Möglichkeit, die Daten zu splitten und auf mehreren Kanälen parallel zu verschicken. Dies geschieht unter Ausnutzung der GridFTP Bibliotheken und soll eine signifikante Übertragungsbeschleunigung bewirken.
- Um die MPI Prozesse und die von ihnen aufgerufenen Funktionen zu synchronisieren (zum Beispiel um nichtblockierende Kommunikation zu verwirklichen) ist verständlicherweise eine umfangreiche Ereignisbehandlung (vom englischen *Event-Handling*) erforderlich. Um diese optimal in das Globus Toolkit einzupassen werden für die Ereignisbehandlung die Globus-eigenen Mechanismen genutzt. Dadurch war es früher nicht möglich, bestimmte Globus-Bibliotheken zu verwenden, welche teilweise dieselben Ereignisnummern benutzten und so Ereignisse auftreten ließen, deren Empfänger nicht eindeutig aufgelöst werden konnte. So verursachten MPI-Funktionen Ereignisse für die Globus-Bibliotheken aus und umgekehrt. Durch konsequente Anpassung an die weiterentwickelten Möglichkeiten des Globus Toolkits konnten verschiedene Ereignisräume für die MPI-Funktionen und andere Bibliotheken geschaffen werden. Damit geschieht nun die Ereignisbehandlung von MPI völlig transparent für den Programmierer, und Nebeneffekte mit anderen Bibliotheken sind in dieser Hinsicht ausgeschlossen.
- Wie schon anfangs erwähnt, beinhaltet das Message Passing Modell nicht nur die Kommunikation von Prozess-zu-Prozess, sondern auch die Übertragung zu mehreren Prozessen, wie zum Beispiel einen Broadcast – also eine Nachrichtenübermittlung an alle Prozesse. Um den Kommunikationsaufwand bzw. die Übertragungsdauer zu minimieren werden zur Optimierung bestimmte Algorithmen verwendet. MPICH benutzt hierbei beispielsweise einen auf Binärbäumen basierenden Algorithmus. Da in einem – durchaus weltweiten – Grid die beteiligten Computer und damit auch die ihnen zugeteilten Prozesse sehr weit auseinander liegen können, macht es verständlicherweise Sinn, die Topologie mit zu berücksichtigen und die Kommunikation zwischen weit entfernten Prozessen zu minimieren. MPICH-G2 verwendet bei fast allen sogenannten *kollektiven Operationen* einen entfernungsbasierten Algorithmus.

Dazu klassifiziert MPICH-G2 die Entfernungen zwischen Prozessen in vier Kategorien bezüglich der angewendeten Kommunikationsmethode. Diese sind nachfolgend von der langsamsten zur schnellsten aufgeführt:

- TCP über Wide Area Network (WAN)
- TCP über Local Area Network (LAN)
- TCP innerhalb eines Rechners
- herstellerspezifische MPI-Bibliotheken innerhalb eines Rechners

Beim Beispiel des Broadcasts würde ein Prozess im ersten Schritt die Nachricht an je einen Prozess pro über das WAN erreichbarem Netzwerk weitergeben. Im nächsten Schritt würden dann alle diese Prozesse an je einen Prozess pro Rechner im LAN die Nachricht gleichzeitig weitergeben. Als Nächstes würden diese dann – falls möglich – die Nachricht an andere Prozesse im Rechner, die sie mittels TCP erreichen, parallel übermittelt. Zuletzt werden dann die Prozesse informiert, welche über herstellerspezifische MPI-Bibliotheken miteinander kommunizieren. Die Broadcasts innerhalb einer Kategorie (wie zum Beispiel die Weiterleitung der Nachricht von einem Rechner im LAN zu den restlichen) wird mit den Standardalgorithmen der benutzten MPI-Bibliotheken vorgenommen, also entweder aus den MPICH-Bibliotheken oder – für die letzte Kategorie – aus den vom Hardwarehersteller vorgesehenen Algorithmen.

Dieses Nachichtenübermittlungsverfahren wird bei allen kollektiven Operationen bis auf `MPI_Gatherv()`, `MPI_Scatterv()`, und `MPI_Alltoallv()` angewendet.

· Wie wir oben gesehen haben, besitzt MPICH-G2 einige Informationen über die Topologie des Grids. Diese Information ist natürlich für MPI-Programmierer sehr interessant, denn es macht zum Beispiel in bestimmten Fällen Sinn mittels MPI-Kommunikatoren bestimmte Operationen in LANs zu bündeln anstatt willkürlich Prozesse zusammenzufassen. Natürlich kann man – wie MPICH-G2 auch – dazu entsprechende Globus Bibliotheken verwenden, jedoch würde dabei die Komplexität des Programmes zunehmen. Vorausschauenderweise gibt uns MPICH-G2 diese Informationen schon bei jedem Kommunikator mit dazu. Die beiden Attribute `MPICHX_TOPOLOGY_DEPTHS` und `MPICHX_TOPOLOGY_COLORS` werden mit jedem Kommunikator gesetzt und enthalten folgende Informationen:

  - `MPICHX_TOPOLOGY_DEPTHS` ist ein Vektor, der für jeden Prozess im

Kommunikator einen Integer-Eintrag enthält. Dieser Eintrag ist entweder "vier", falls der Prozess über eine herstellerspezifische MPI-Anbindung verfügt, oder der Eintrag lautet "drei", wenn dem nicht so ist und somit der Prozess mittels klassischem TCP-Protokoll angebunden ist.
- `MPICHX_TOPOLOGY_COLORS` ist ein Vektor, der für jeden Prozess im Kommunikator wiederum einen Vektor auf einen Integer enthält, dessen Länge der `MPICHX_TOPOLOGY_DEPTHS` des Prozesses entspricht (also entweder drei oder vier). Jeder dieser drei bzw. vier Einträge entspricht einer der Kommunikationsebenen, wie sie im vorangegangenen Abschnitt vorgestellt worden sind:
(1) erster Eintrag für Kommunikation über WAN-TCP
(2) zweiter Eintrag für Kommunikation über LAN-TCP
(3) dritter Eintrag für Kommunikation über TCP innerhalb eines Rechners
(4) vierter Eintrag für Kommunikation über herstellerspezifische MPI-Bibliotheken
Klar ist, dass der vierte Eintrag nur für Prozesse existiert, die auch über herstellerspezifische MPI-Bibliotheken zum Kommunizieren verfügen.
Alle Prozesse, die nun zum Beispiel mittels LAN-TCP miteinander kommunizieren können, haben im zweiten Eintrag unter ihrer jeweiligen Prozessnummer die gleiche "Farbe", das heißt den gleichen Integer-Wert.

3.4 Grundlegende MPI-Funktionen

In dem nun folgenden Abschnitt sollen alle in zukünftigen Programmbeispielen verwendeten MPI-Funktionen kurz vorgestellt und erläutert werden.
Da als Programmiersprache nur C benutzt wird, werden auch alle Funktionen im C-Stil dargestellt. Mit Grundkenntnissen in C bzw. auch Java, wie sie zum Beispiel im Grundstudium der Informatik erworben werden, sollte das Verständnis der MPI-Funktionsbibliothek und der Programme keine Probleme bereiten.
Die hier aufgeführten Funktionen sind alle in der Bibliothek `mpi.h` enthalten. Die Syntax orientiert sich an dem Buch "Parallele und verteilte Programmierung" [130].

3.4.1 MPI Datentypen, welche im Folgenden benutzt werden

`MPI_Comm`

Eine Variable vom Typ `MPI_Comm` wird *Kommunikator* genannt und dient als Identifikator einer Menge von Prozessen, die untereinander Nachrichten austauschen wollen. Sie wird normalerweise nicht direkt vom Programm verändert, sondern von MPI Funktionen als Rückgabeparameter erhalten oder ihnen übergeben. Ein wichtiger vordefinierter Kommunikator ist `MPI_COMM_WORLD`, welcher alle in der MPI Umgebung gestarteten Prozesse umfasst.

`MPI_Datatype`

Variablen vom Typ `MPI_Datatype` enthalten in der Regel vordefinierte Konstanten wie `MPI_INT`, `MPI_UNSIGNED_LONG` oder `MPI_CHAR`. Normalerweise werden sie bei der Kommunikation verwendet, damit der Empfänger weiß, als was er die empfangenen Daten zu interpretieren hat. Generell kann man sagen, dass es zu jedem Datentyp in C einen korrespondierenden Typ in MPI gibt. In MPICH-G2 existiert sogar der Datentyp `MPI_LONG_LONG`, welcher in der Regel mit 64 Bit implementiert ist.

`MPI_Status`

Dieser Datentyp bezeichnet eine Struktur, die Statusinformationen empfangener Nachrichten enthält. Eine Variable von einem solchen Typ wird also analog zu `MPI_Comm` vom Programm nicht selbst verändert, sondern von den entsprechenden MPI Funktionen gesetzt bzw. ihnen übergeben.
Die Variable `status` vom Typ `MPI_Status` würde folgende Informationen enthalten:

- `status.MPI_SOURCE` enthält die Nummer des Absenders der empfangenen Nachricht als Integer
- `status.MPI_TAG` enthält die (frei wählbare) Kennziffer der empfangenen Nachricht als Integer
- `status.MPI_ERROR` liefert einen Fehlercode als Integer

3.4.2 MPI Funktionen, auf die im Folgenden zurückgegriffen wird

In den nachfolgenden Kapiteln wird bei Programmbeispielen und Programmentwicklung auf die MPI-Bibliotheken zurückgegriffen. Da dies normalerweise nicht Teil des Informatikstudiums ist, aber zum tieferen Verständis der nachfolgenden Kapitel 4 und 6 als notwendig angesehen werden kann, sind hier die wichtigsten Funktionen aufgeführt. Mit den zugehörigen Erklärungen sollte es keine Probleme bereiten, die anstehenden Programme nachzuvollziehen.

```
int MPI_Init (int *argc, char **argv)
int MPI_Finalize (void)
```

Diese beiden Funktionen sind für ein MPI Programm essentiell, obwohl sie für den Inhalt des Programms selbst nicht von Bedeutung sind. In übertragenem Sinne bilden sie den Rahmen für den Teil des Programms, welcher MPI Funktionen benutzt. Genau genommen muss also `MPI_Init()` als erste MPI Funktion im Programm aufgerufen werden, bevor irgendeine andere benutzt wird. Erst jetzt wird das Programm ein Teil der MPI Umgebung. Analog dazu bewirkt `MPI_Finalize()` eine Art Abmeldung aus der MPI Umgebung und ist somit die letzte MPI Funktion im Programm, welche aufgerufen wird.

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

Mit dieser Funktion kann man die Anzahl der Prozesse erfahren, welche im angegebenen Kommunikator erreichbar sind. Speziell würde man also mit `MPI_COMM_WORLD` als Kommunikator die Anzahl aller in der MPI Umgebung erreichbaren Prozesse erfahren.

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Als Rückgabewert liefert diese MPI Funktion die Prozessnummer des eigenen Prozesses in den angegebenen Kommunikator zurück. Dabei ist zu beachten, dass die Prozesse von 0 an aufwärts durchnummeriert sind.

Wenn zum Beispiel obige Funktion `MPI_Comm_size()` für den Kommunikator `MPI_COMM_WORLD` den Wert p zurückgegeben hat, würden wir hier bei `MPI_Comm_rank()` für den gleichen Kommunikator einen Wert von 0 bis p-1 erhalten.
Somit ist es leicht, mit demselben Programmtext durch Abfrage der eigenen Prozessnummer für jeden Prozess unterschiedliche Programmteile auszuführen. Wie anfangs erwähnt, erspart dies das umständliche Arbeiten mit einem Programmtext pro Prozess und es ist problemlos möglich, eine variable Anzahl von Prozessen im Programm vorzusehen.

```
double MPI_Wtime(void)
```

Diese Funktion liefert einen Zeitstempel in Sekunden als Double zurück, welcher beim Benchmarken bestimmter Programmabschnitte benutzt werden kann. Durch das Aufrufen vor und nach dem interessierenden Bereich kann man die Differenz der Rückgabewerte bilden, die dann die verbrauchte Zeit angibt.

```
int MPI_Send(void *smessage,
             int count,
             MPI_Datatype datatype,
             int dest,
             int tag,
             MPI_Comm comm)
```

Mit dieser MPI Funktion kann ein Prozess an einen anderen eine Nachricht übermitteln. Die Nachricht ist dabei in `smessage` gespeichert und besteht aus `count` vielen Elementen des Typs `datatype`. Der Zielprozess im Kommunikator `comm` hat die Nummer `dest`. Es ist noch möglich, der Nachricht eine beliebige Nummer `tag` mitzugeben, um dem Empfänger die inhaltliche Interpretation der Nachricht zu erleichtern, insbesondere wenn mehrere Nachrichten von ein und demselben Sender empfangen werden.
Dabei gehört `MPI_Send()` im Allgemeinen zu der Klasse der so genannten *blockierenden Kommunikationsanweisungen*. Im Gegensatz zu den anderen (nichtblockierenden) Funktionen wird das Programm erst dann fortgesetzt, wenn die

übergebenen Variablen nicht mehr benötigt werden. Dies ist entweder dann der Fall, wenn der Empfänger die Nachricht aus `smessage` ausgelesen hat oder wenn – wie bei einigen MPI Implementierungen – die Nachricht in einem Systempuffer zwischengespeichert wird. Dieses kann natürlich einen effektiven Geschwindigkeitsgewinn bedeuten, aber bei hohem Kommunikationsaufwand steigt auch der Speicherverbrauch rapide an.

```
int MPI_Recv(void *rmessage,
             int count,
             MPI_Datatype datatype,
             int source,
             int tag,
             MPI_Comm comm,
             MPI_Status *status)
```

Mittels der `MPI_Recv()` Funktion können Nachrichten von Prozessen empfangen werden, welche per `MPI_Send()` verschickt wurden. Die empfangene Nachricht wird in `rmessage` gespeichert und wird als maximal `count` viele Elemente des Datentyps `datatype` interpretiert. Da mehrere Nachrichten zu einem Prozess geschickt werden können, kann man die mögliche Nachricht einerseits einschränken, indem mit `source` ein spezieller Prozess im Kommunikator `comm` angegeben wird, von dem Nachrichten angenommen werden und andererseits mit `tag` nur bestimmte Nachrichtentypen ausgewählt werden. Wenn keinerlei Einschränkungen gewünscht werden, so ist es möglich, anstatt einen bestimmten Prozess mit `source` anzugeben, die Konstante `MPI_ANY_SOURCE` zu verwenden oder – statt eines bestimmten Wertes für `tag` – die Konstante `MPI_ANY_TAG` anzugeben. Genauere Angaben einer so empfangenen Nachricht kann man dann – wie weiter oben ausgeführt – der Variable `status` entnehmen.
Auch diese Funktion blockiert das Programm so lange, bis eine Nachricht eingetroffen ist, welche in das Anforderungsschema bezüglich `comm`, `source` und `tag` passt. Das ist normalerweise nicht weiter gravierend, wenn ohne die zu empfangenden Daten das Programm ohnehin nicht weiterarbeiten könnte.

4. Installationsbeispiel des Globus Toolkits 2 und MPICH-G2

Im Folgenden soll die Installation des verwendeten Globus Toolkits 2.4 mit der MPI-Erweiterung MPICH-G2 besprochen werden, welche für den praktischen Teil der Arbeit als Referenzsystem dient. Zur Zeit der Programmentwicklung im 6. Kapitel war MPICH-G3 für das Globus Toolkit 3 noch nicht erschienen. Letzteres wäre auch allein durch die nötige Web Service Engine für die von mir verwendete Hardware zu überdimensioniert. Für das neue Globus Toolkit 4 von Mai 2005 gibt es noch keine MPI-Implementierung. Allerdings wird die Installation der neueren Versionen auch mit dem unten vorgestellten GPT durchgeführt – nur die Einrichtung der zusätzlichen Komponenten des Toolkits kann hier natürlich nicht beschrieben werden.
Als System nehme ich im Weiteren einen PC auf Pentium-Basis an, welcher SuSE Linux in der Version 8.2 installiert hat, wobei natürlich zu beachten ist, dass auf abweichenden Systemen andere Befehle bzw. Ausdrücke erforderlich sein könnten. Die SuSE-Distribution enthält alle benötigten Programme und Werkzeuge um das Globus Toolkit problemlos zu installieren. Außer den Dateien des Toolkits selbst werden also keine externen Ressourcen mehr benötigt.

4.1 Installation des Grid Packaging Toolkits und des Globus Toolkits

Eine so komplexe Open-Source Entwicklung wie das Globus Toolkit (GT) besteht aus mehreren Komponenten, welche nicht nur in Verwendungszwecke – wie eine volle Serverinstallation oder einen Clientzugang zum Grid – aufgeteilt sind, sondern auch spezielle Werkzeuge und Bibliotheken für die Entwicklung am Globus Toolkit selbst zur Verfügung stellen können. Darüber hinaus gibt es Forschungsprojekte für Ergänzungen und Erweiterungen, die insbesondere nachträglich installiert bzw. deinstalliert werden können bzw. müssen. Um bei Installationen, Deinstallationen oder natürlich auch Updates einzelner Komponenten die Integrität und Versionsstruktur zu wahren, wurde als Installationsprogramm von der NCSA das sogenannte Grid Packaging Toolkit – oder kurz GPT – entwickelt [74].

Die Funktionsweise des GPT entspricht der anderer Installationsprogramme – wie zum Beispiel rpm. Die kleinsten Installationseinheiten sind die sogenannten *Packages*, welche allerdings normalerweise zu themenbezogenen *Bundles* zusammengefasst werden. Typische kleinere Bundles wären die Module des Globus Toolkits wie GSI, GRAM, MDS oder GridFTP. Damit können diese Module einzeln weiterentwickelt werden und es ist bei Bedarf nur ein Teilupdate des Toolkits nötig [51].
Darüber hinaus sind natürlich Makro-Bundles wie das gesamte Globus Toolkit als Server- bzw. Client-Installation möglich, was insbesondere bei der Erstinstallation einfacher zu handhaben ist. Grundsätzlich sind alle Bundles wegen des Open-Source Gedankens natürlich sowohl in Source- als auch in diversen Binary-Versionen verfügbar. Die schon übersetzten Bundles sind für alle gängigen UNIX und Linux Derivate für sämtliche aktuellen Prozessortypen erhältlich. In letzter Zeit werden auch verstärkt Umsetzungen für MacOS X – das Betriebssystem von Apple mit UNIX-Kernel – angeboten.
Ein wichtiger Unterschied zwischen fertig übersetzten Bundles und deren Source-Varianten sind sogenannte *Flavors*. Diese Flavor-Namen beinhalten die zum Übersetzen gewählten Optionen.

Folgende Beispiele für Flavor-Namen veranschaulichen das Prinzip:

- "gcc32dbgpthr" bedeutet, dass der GNU Compiler für 32-Bit Architekturen mit aktivierter Debug-Option und pthread-Bibliothek benutzt wurde
- "vendorcc64dbgmpi" bedeutet, dass der hardwareherstellerspezifische Compiler für 64-Bit Architekturen benutzt wurde, die Debug-Option aktiviert und auch auf die MPI Funktionen des Herstellers für die MPI-Kommunikation zurückgegriffen werden kann

Die schon vorgefertigten Binary-Bundles benutzen normalerweise die Flavors "gcc32dbgpthr" und "gcc32dbg". Damit ist klar, dass für die Ausnutzung einer 64-Bit Architektur bzw. der Nutzung von nativen MPI Funktionen ein Source-Bundle benutzt werden muss.

4.1.1 Voraussetzungen für die Installation

Um diese Source-Bundles bzw. einzelne Source-Packages zu entpacken und zu übersetzen, muss GPT Zugriff auf folgende vorher installierte Programme haben:

- Perl (mindestens in der Version 5.005), da sämtliche Scripts von GPT in Perl geschrieben sind
- gcc, der GNU Compiler zum Übersetzen der Source-Packages
- GNU tar, zum Entpacken (bzw. Expandieren) der Packages bzw. Bundles
- GNU make, da zu jedem Source-Package ein `Makefile` zum Übersetzen beiliegt

Zur Einrichtung des Servers des Globus Toolkits ist natürlich ein Internet-Deamon unumgänglich:

- `xinetd`
- oder der ältere `inetd` (die hier verwendete Syntax kann aber abweichen)

4.1.2 Durchführung der Beispielinstallation

Zur Installation wird von einem speziell dafür angelegten Benutzer mit dem Namen `globus` ausgegangen. Dies ist nicht unbedingt notwendig, hat aber den Vorteil, dass das Rechteschema dieses Benutzers gleich dem Rechteschema des Toolkits gesetzt werden kann, insbesondere was den Zugriff auf das Dateisystem betrifft. Das Verzeichnis `$HOME` liegt dann bei `/home/globus` (SuSE 8.2).
Des Weiteren wird für alle Eingaben in der Kommandozeile der Syntax die bash-Shell benutzt. Diese erkennt man an dem vorangestellten ">"-Zeichen.

4.1.2.1 Die benutzten Archive

Hier wird die Source-Version des GPT 3.0.1 verwendet, welche alle Globus Toolkits bis zur Version 3 unterstützt:

`gpt-3.0.1-src.tar.gz`[1]

Zur Vereinfachung wird auf die letzte Version des Binary-Bundles des Globus Toolkits 2.4 zurückgegriffen, und zwar inklusive aller Module:

`globus-all-2.4.3-i686-pc-linux-gnu-bin.tar.gz`[2]

Für die Authentifikation benötigen wir noch die entsprechenden Zertifikate zum Verifizieren:

`globus_gcs_b38b4d8c_setup-0.1.tar.gz`[3]

4.1.2.2 Anlegen der Verzeichnisse und Setzen der Umgebungsvariablen

```
>mkdir $HOME/gpt
>mkdir $HOME/globus

>export GPT_LOCATION=$HOME/gpt
>export GLOBUS_LOCATION=$HOME/globus
```

4.1.2.3 Entpacken und Installation des GPT

Es wird ein Verzeichnis `gpt-3.0.1` erstellt, welches nach der Installation gelöscht werden kann.

```
>gzip -dc gpt-3.0.1-src.tar.gz | tar xf -
>cd gpt-3.0.1
>./build_gpt
```

4.1.2.4 Installation des Globus Toolkits

Das Binary-Bundle wird zur Installation dem GPT übergeben. Das ausgewählte Bundle beinhaltet die Client- und Server-Bundles sowie alle Bibliotheken des Software Development Kits (SDK):

1 verfügbar unter "www-unix.globus.org/ftppub/gt2/2.4/2.4-latest/gpt/gpt-3.0.1-src.tar.gz"
2 verfügbar unter "www-unix.globus.org/ftppub/gt2/2.4/2.4-latest/globus-all-sdk-2.4.3-i686-pc-linux-gnu-bin.tar.gz"
3 verfügbar unter "www-unix.globus.org/security/gcs/globus_gcs_b38b4d8c_setup-0.1.tar.gz"

```
>$GPT_LOCATION/sbin/gpt-install \
globus-all-2.4.3-i686-pc-linux-gnu-bin.tar.gz
```

Die Umgebungsvariablen werden mittels eines vorgefertigten Shell-Scripts gesetzt:

```
>source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

Am besten sollte dieses Skript mit den obigen Variablen `GPT_LOCATION` und `GLOBUS_LOCATION` in die rc-Datei der verwendeten Shell integriert werden, damit dies nach dem Einloggen des Benutzers nicht immer von Hand ausgeführt werden muss, um die Programme und Werkzeuge des Toolkits zu nutzen.

4.1.2.5 Abschließen der Installation

GPT überprüft die Integrität darauf, ob alle Packages richtig intalliert wurden, bzw. führt abschließende Skripte der Packages aus:

```
>$GPT_LOCATION/sbin/gpt-postinstall
```

Der Vollständigkeit halber sollte noch der Header-File für das Software Development Kit (SDK) erzeugt werden. Als Flavor wird der Standard-Flavor der Binary-Bundles verwendet (gcc32dbg).

```
>$GPT_LOCATION/sbin/gpt-build gcc32dbg -nosrc
```

4.1.2.6 Einrichten der Grid Security Infrastructure (GSI)

Die Grid Security Infrastructure (GSI) muss natürlich mit Supervisor-Rechten eingerichtet werden, deshalb ist es notwendig zum Benutzer `root` zu wechseln:

```
>su
>$GLOBUS_LOCATION/setup/globus/setup-gsi
```

Dabei ist zu bedenken, dass dafür auch als `root` die Pfade bzw. Umgebungsvariablen gesetzt werden müssen. Dies muss allerdings nur einmal bei der Installation geschehen.

4.1.2.7 Überprüfen der Installation

Das folgende Kommando sollte ohne Fehlermeldung ausgeführt werden, ansonsten wurden offensichtlich einige Packages gar nicht oder nicht richtig installiert:

```
>$GPT_LOCATION/sbin/gpt-verify
```

4.1.2.8 Installation der Globus-Zertifikate zur Authentifikation

Um eine sichere Authentifikation von Benutzern und Computern zu gewährleisten, müssen X.509 Zertifikate ausgestellt werden und die dazu verwendeten Schlüssel der ausstellenden Stelle verfügbar sein. Wenn keine eigene Infrastruktur dafür besteht, gibt es die Möglichkeit, die Zertifizierung direkt über eine Internetseite bei www.globus.org vorzunehmen. Damit der lokale Rechner diese Zertifikate anerkennt, braucht er natürlich eine Kopie der erzeugten Schlüsselpaare der Zertifizierungsstelle.
Dieses kleine Package wird natürlich auch mittels GPT installiert:

```
>$GPT_LOCATION/sbin/gpt-build \
globus_gcs_b38b4d8c_setup-0.1.tar.gz
>$GPT_LOCATION/sbin/gpt-postinstall
>su
>$GLOBUS_LOCATION/setup/globus_gcs_b38b4d8c_setup/setup-gsi
```

4.1.3 Konfiguration des Globus Toolkits

Nun ist das Globus Toolkit 2 zwar auf einem bzw. mehreren Rechnern installiert, allerdings muss noch einiges an Arbeit in die Konfiguration des Systems und des Toolkits gesteckt werden, bevor es vollständig benutzt werden kann.

4.1.3.1 Anforderung des Benutzerzertifikats

Jeder Benutzer muss sich dem Grid gegenüber mit einem Zertifikat ausweisen. Dies sei hier am Beispiel des Benutzers `globus` gezeigt. Natürlich kann jede beliebige

sogenannte *Certificate Authority* (CA) zum Ausstellen der entsprechenden Zertifikate benutzt werden, allerdings würde eine derartige Installation den Rahmen hier sprengen. Statt dessen kann auf eine CA über die Seite des Globus Projects zurückgegriffen werden[4].

```
>grid-cert-request
```

Dieser Aufruf erzeugt eine Datei (`/home/globus/.globus/usercert_request.pem`), welche auf der entsprechenden Seite[5] eingefügt werden sollte. Man erhält so ein Zertifikat, das man unter `/home/globus/.globus/usercert.pem` speichert.
Sämtliche ausgestellten Zertifikate im Grid, welche je einen Grid-Benutzer darstellen, müssen auf jedem Computer einem dort existierenden Benutzer-Account entsprechen, um den lokalen Zugriffsbestimmungen unterworfen zu sein. Die einfachste Möglichkeit ist es, das Globus Toolkit unter einem eigenen Account zu installieren – hier `globus` – und die Zugriffsrechte auf die lokalen Ressourcen für das Grid stellvertretend für diesen Account zu vergeben.

```
>su
>echo \
"\C=US/O=Globus\ Alliance/OU=User/CN=fa5d0b5ead.a1ef3585 \
globus" > \
/etc/grid-security/grid-mapfile
>chmod 644 /etc/grid-security/grid-mapfile
```

4.1.3.2 Anforderung des Rechnerzertifikats

Jeder physikalische Computer, also hier jeder PC, auf dem das Globus Toolkit installiert wurde, benötigt ein eigenes Zertifikat. Dies wird zur Authentifikation bei Verbindungen zwischen den Computern benötigt, wie z.B. Datentransfer. Diese sind nur auf die Grid-Funktionalität bezogen und haben keine Verbindung zu ähnlichen Verfahren, wie z.B. bei der Security Shell (ssh).

4 Globus online Certificate Authority erreichbar unter "gcs.globus.org:8080/gcs/index.html"
5 Benutzerzertifikat beantragen unter "gcs.globus.org:8080/gcs/usercert.html"

Wichtig ist hierbei, dass nach dem Parameter `host` der volle Rechnername angegeben wird:

```
>su
>grid-cert-request -service host -host \
fontane.informatik.uni-giessen.de
```

Es wid die Datei `/etc/grid-security/hostcert_request.pem` erzeugt, welche auf der entsprechenden Seite[6] eingefügt wird. Die dort erhaltene Datei wird unter `/etc/grid-security/hostcert.pem` gespeichert.

4.1.3.3 Anforderung des LDAP-Zertifikats

Die Information Services des Globus Toolkits benutzen eine spezialisierte Datenbank, in der sämtliche Informationen aller Ressourcen und Zustände des Grids verfügbar sind. Im Detail wird der LDAP-Dienst weiter unten im Abschnitt 4.4 vorgestellt.
Zur Authentifikation untereinander erhält jeder Rechner ein LDAP-Zertifikat. Allerdings ist es auch möglich, anonym auf die Daten zuzugreifen, ohne LDAP-Zertifikate auszustellen. Die Standardeinstellung erlaubt dies.

Analog zu den anderen Zertifikaten wird wieder `grid-cert-request` mit den entsprechenden Parametern aufgerufen. Auch hier ist darauf zu achten, dass nach dem Parameter `host` der volle Rechnername angegeben wird:

```
>su
>grid-cert-request -service ldap -host \
fontane.informatik.uni-giessen.de
```

Die generierte Datei `/etc/grid-security/ldap/ldapcert_request.pem` wird auf der dafür vorgesehenen Seite[7] eingefügt, dann wird wie zuvor die erhaltene Datei als `/etc/grid-security/ldapcert.pem` gespeichert.

6 Rechnerzertifikat beantragen unter "gcs.globus.org:8080/gcs/servicecert.html"
7 LDAP-Zertifikat beantragen unter "gcs.globus.org:8080/gcs/servicecert.html" mit Option "LDAP"

4.1.4 Testlauf von GRAM

Um diesen Test durchführen zu können, ist es notwendig, dass – wie oben beschrieben – die Zertifikatsdateien `usercert.pem` und `hostcert.pem` vorliegen. Damit haben wir alle restlichen Voraussetzungen für einen ersten Testlauf des Globus Toolkits erfüllt.
Als Erstes starten wir einen Proxy, welcher die Aufgabe übernimmt bei allen Authentifikationsanfragen im Grid unser Zertifikat stellvertretend vorzuzeigen:

```
>grid-proxy-init -debug -verify
```

Jetzt wird nach der *Passphrase* des Zertifikats des momentanen Benutzers gefragt, welche bei der Ausführung von `grid-cert-request` eingegeben wurde.
Da wir noch keinen Gatekeeper als Dienst im Hintergrund eingerichtet haben, also eine Authentifikations-Instanz, um auf die lokalen Grid-Ressourcen zugreifen zu können, müssen wir ihn selbst starten:

```
>globus-personal-gatekeeper -start
```

Er gibt eine Zeichenkette aus, welche mit einer Kontaktadresse vergleichbar ist. Diese nennt man *Contact* oder auch *Contact-String* und ist eine eindeutige Bezeichnung für diesen speziellen Gatekeeper.
Anschließend können wir dem Jobmanager (von GRAM) ein Programm mitsamt dem Contact-String übergeben, um eine Zuteilung der nötigen Ressourcen wie Speicher und Prozessor zu erreichen:

```
>globus-job-run "<CONTACT>" /bin/date
```

Nun sollte das aktuelle Datum mit Uhrzeit erscheinen.

Abschließend sollten sowohl der Gatekeeper als auch der Proxy beendet werden:

```
>globus-personal-gatekeeper -killall
>grid-proxy-destroy
```

4.1.5 Testlauf von MDS

Hierbei ist vorerst das LDAP-Zertifikat `ldapcert.pem` nicht notwendig, da die Anfragen in der Standardeinstellung anonym durchgeführt werden.

Als Erstes werden die Information Services des Globus Toolkits gestartet:

```
>$GLOBUS_LOCATION/sbin/globus-mds start
```

Der folgende Befehl gibt die Daten des lokalen LDAP Verzeichnisses aus, allerdings ist die Abfrage nicht sehr komfortabel und hauptsächlich für solche Testzwecke geeignet:

```
>$GLOBUS_LOCATION/bin/grid-info-search -anonymous -L
```

4.1.6 Testlauf von GridFTP

Als vielseitiger Vertreter des Data Managements des Globus Toolkits ist die Funktionalität von GridFTP besonders wichtig,

Zur Steuerung der Zugriffsrechte muss natürlich ein Bezug zwischen lokalen Benutzerkonten des Computers und den zertifizierten Grid-Benutzern gewährleistet sein. Deshalb wird dieser Bezug in einer Konfigurationsdatei hergestellt. Normalerweise geschieht dies in der Datei `/etc/grid-security/grid-mapfile`, wird diese aber nicht gefunden (zum Beispiel bei einer reinen Client-Installation ohne GSI), so kann auf die Benutzerdatei `$HOME/.gridmap` zurückgegriffen werden. Generell ist es sinnvoll, alle Zertifikate auf den Account abzubilden, unter dem das Globus Toolkit installiert wurde – in diesem Fall also der Benutzer `globus`:

```
>echo \
"/C=US/O=Globus\ Alliance/OU=User/CN=fa5d0b5ead.a1ef3585 \
globus" > ~/.gridmap
```

Wie üblich wird vor dem Zugriff auf das Grid der Proxy gestartet:

```
>grid-proxy-init -verify -debug
```

Anschließend wird der GridFTP-Deamon gestartet, hier auf Port 5678:

```
>$GLOBUS_LOCATION/sbin/in.ftpd -S -p 5678
```

Jetzt wird noch eine Testdatei `/tmp/file1` angelegt und anschließend mittels GridFTP als Datei `/tmp/file2` kopiert:

```
>echo "testfile" > /tmp/file1
>globus-url-copy -s "`grid-cert-info -subject`" \
gsiftp://localhost:5678/tmp/file1 file:///tmp/file2
```

Wenn die neue Datei `/tmp/file2` mit entsprechendem Inhalt erstellt worden ist, so ist der Testlauf gelungen.
Wen alle Testläufe gelungen sind, arbeiten alle Komponenten des Globus Toolkits einwandfrei.

4.1.7 Einrichten eines Servers mit dem Globus Toolkit

In den vorangegangenen Testläufen haben wir mit teilweise sehr spezialisierten Programmen einzelne Funktionalitäten des Globus Toolkits benutzt. Um aber ein Grid aus mehreren Rechnern aufzubauen, müssen die Dienste benutzerunabhängig gestartet werden und ohne äußeres Zutun funktionieren. Hierfür ist der Server des Globus Toolkits zuständig, dessen Komponenten wir mit dem Bundle weiter oben installiert haben. Nun muss die Integration der Komponenten in das bestehende System vorgenommen werden. Dies ist natürlich besonders stark vom verwendeten Betriebssystem und dessen Hilfsprogrammen abhängig, aber dank der weitreichenden Standardisierung von UNIX/Linux-Systemen dürften die Abweichungen nur geringfügig sein. Die nachfolgenden Änderungen werden selbstverständlich mit Superuser-Rechten durchgeführt.

Um die netzweite Erreichbarkeit der lokalen Globus-Dienste zu ermöglichen, müssen diesen entsprechende Ports zugewiesen werden.
Als Erstes müssen dazu der Datei `/etc/services` folgende Zeilen hinzugefügt werden:

```
gsi             1850/tcp        # GSI
gsi             1850/udp        # GSI
gsigatekeeper   2119/tcp        # GSIGATEKEEPER
gsigatekeeper   2119/udp        # GSIGATEKEEPER
gsiftp          2811/tcp        # GSI FTP
gsiftp          2811/udp        # GSI FTP
```

Als Nächstes müssen die Dienste bei dem verwendeten Internet-Deamon registriert werden. Hier wird die Syntax des immer häufiger benutzten `xinetd` verwendet.

In dem Ordner `/etc/xinetd.d` muss der folgende Text als Datei `globus-gatekeeper` hinzugefügt werden:

```
service gsigatekeeper
{
   socket_type  = stream
   protocol     = tcp
   wait         = no
   user         = root
   env          = LD_LIBRARY_PATH=/home/globus/globus/lib
   server       = /home/globus/globus/sbin/globus-gatekeeper
   server_args  = -conf /home/globus/globus/etc/globus-gatekeeper.conf
   disable      = no
}
```

Ebenso wie oben wird im Ordner `/etc/xinetd.d` auch dieser Text als Datei `grid-ftp` hinzufügt:

```
service gsiftp
{
   instances        = 1000
   socket_type      = stream
   wait             = no
   user             = root
   env              = LD_LIBRARY_PATH=/home/globus/globus/lib
   server           = /home/globus/globus/sbin/in.ftpd
   server_args      = -l -a -G /home/globus/globus
   log_on_success  += DURATION USERID
   log_on_failure  += USERID
   nice             = 10
   disable          = no
}
```

Die Monitoring and Discovery Services (MDS) werden als ein normaler Prozess gestartet. Um dies gleich beim Rechnerstart zu ermöglichen, sollte der Befehl

`/home/globus/globus/sbin/globus-mds start`

in die Start-Up Dateien des Sytems aufgenommen werden.

Bei dem hier verwendeten Basissystem genügt je ein Eintrag in die Ordner `/etc/rc.d/rc3.d` und `/etc/rc.d/rc5.d`, wo auch `xinetd` gestartet wird.

Einfach und effektiv – wenn auch ein wenig unsauber – wäre folgende Anweisung:

```
>echo "/home/globus/globus/sbin/globus-mds start" >> \
/etc/rc.d/xinetd
```

Es sei hierbei noch angemerkt, dass bei neueren SuSE Versionen der Ordner `rc.d` in `init.d` umbenannt wurde.

Ein Neustart des Systems sollte nun alle relevanten Globus-Dienste starten.

4.1.8 Erstellen und Starten eines Globus-Testprogramms

In diesem Abschnitt soll kurz die Entwicklung eines Programms für die Grid-Umgebung vorgeführt werden. Um den Programmtext nicht unnötig zu verkomplizieren, wird hierzu das bekannte "Hello World"-Beispiel verwendet. Es wird empfohlen, alle beteiligten Dateien in einem eigens dafür angelegten Ordner unterzubringen.

Der Programmtext von `hello.c`:

```
#include <globus_duroc_runtime.h>

int main(int argc, char **argv)
{
#if defined(GLOBUS_CALLBACK_GLOBAL_SPACE)
    globus_module_set_args(&argc, &argv);
#endif

    globus_module_activate(GLOBUS_DUROC_RUNTIME_MODULE);
    globus_duroc_runtime_barrier();
    globus_module_deactivate(GLOBUS_DUROC_RUNTIME_MODULE);

    printf("hello, world\n");
}
```

Der Makefile `makefile` zum Übersetzen des Quellcodes:

```
include makefile_header
hello:
      $(GLOBUS_CC) $(GLOBUS_CFLAGS) $(GLOBUS_INCLUDES) -c hello.c
      $(GLOBUS_LD) -o hello hello.o \
      $(GLOBUS_LDFLAGS) \
      $(GLOBUS_PKG_LIBS) \
      $(GLOBUS_LIBS)
clean:
      /bin/rm -rf *.o hello
```

Die Datei `makefile_header` muss noch erzeugt werden. Dafür existiert ein entsprechendes Werkzeug des Globus Toolkits: `globus-makefile-header`. Ihm werden sowohl der Flavor der Installation als auch die im Programm benutzten Packages übergeben.

```
>$GLOBUS_LOCATION/bin/globus-makefile-header \
-flavor=gcc32dbg globus_common globus_gram_client \
globus_io globus_data_conversion globus_duroc_runtime \
globus_duroc_bootstrap > makefile_header
```

Jetzt sind alle Vorbereitungen erledigt, um das Programm zu übersetzen:

```
>make hello
```

Wenn das Programm ohne Fehlermeldung übersetzt wurde, steht nun die Binärdatei `hello` zur Verfügung.

Da bereits der Server des Globus Toolkits eingerichtet wurde, muss nur noch der Proxy vom Benutzer gestartet werden. Dieses muss nicht immer wieder neu erfolgen, denn ohne weitere Angaben bleibt der Proxy zwölf Stunden aktiv. Mit der Option `-valid <Stunden>:<Minuten>` kann die Dauer in Stunden und Minuten angegeben werden:

```
>grid-proxy-init -debug -verify -valid 1:0
```

Um ein Programm in der Grid Umgebung auszuführen, ist es von Vorteil, alle dazu notwendigen Parameter dem Resource Management zu übergeben. Dies geschieht mit der sogenannten Resource Specification Language (RSL) [59] in einer Art Beschreibung des auszuführenden *Jobs*.

So könnte eine einfache Beschreibung als RSL-Datei für das Programm `hello` aussehen (`hello.rsl`):

```
+
( &(resourceManagerContact="fontane.informatik.uni-giessen.de")
   (count=2)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
                (LD_LIBRARY_PATH /home/globus/globus/lib/)
   )
   (directory=/home/globus)
   (executable=/home/globus/hello)
)
```

Hierbei bezeichnet `resourceManagerContact` den Zielrechner des Auftrages (bzw. einen dort gestarteten Gatekeeper über dessen Contact-String), `count` steht für die Anzahl der zu startenden Instanzen (das Programm würde hier also zweimal gestartet), `label` ist einfach nur eine frei zu vergebene Beschreibung, mittels `environment` lassen sich Umgebungsvariablen setzen, `directory` bezeichnet das zu setzende Arbeitsverzeichnis und `executable` gibt natürlich die Binär-Datei an.
Wenn man verschiedene Binär-Dateien gleichzeitig übergeben oder mehrere Gatekeeper ansprechen will, so kann man einfach die obige Struktur – entsprechend ausgefüllt – mehrmals in eine RSL-Datei schreiben. So lassen sich teilweise sehr umfangreiche und komplexe Jobs mit vielen *Subjobs* erstellen.

Das Übergeben des soeben erstellten Jobs an das Grid ist relativ einfach:
`>$GLOBUS_LOCATION/bin/globusrun -w -f hello.rsl`
Hierbei bewirkt der Parameter `-w`, dass unnötige Statusmeldungen unterdrückt werden. Die RSL-Beschreibung des auszuführenden Jobs wird mittels `-f hello.rsl` aus der eben vorbereiteten Datei eingelesen.
Wenn alles richtig konfiguriert wurde, sollte anschließend zweimal `Hello World` erscheinen, da mit `count=2` zwei Instanzen von `hello` gestartet wurden.

4.2 Installation von MPICH-G2:

Um MPI-Programme in einem mittels des Globus-Toolkit erstellten Grid auszuführen, ist eine spezielle Variante von MPICH entwickelt worden: MPICH-G2 (siehe 3. Kapitel). Die hier zur Veranschaulichung verwendete Version ist die 1.2.5.2, welche als Quellcode vorliegt:

`mpich.tar.gz`[8]

Es sollte darauf geachtet werden, dass die Umgebungsvariablen des Globus Toolkits wie oben angegeben gesetzt sind. Dies sollte kein Problem darstellen, wenn alle das Globus Toolkit betreffenden Programme unter einem gemeinsamen Account installiert werden, was hier unter der Benutzerkennung `globus` der Fall ist.

Zuerst wird die Datei entpackt. Es entsteht das Verzeichnis `mpich-1.2.5.2`:

```
>gunzip -c mpich.tar.gz | tar xvf -
```

Zum Installieren wird der `makefile` an das Globus Toolkit angepasst, indem auch hier der Flavor der bestehenden Installation übergeben wird:

```
>cd mpich-1.2.5.2
>./configure -device=globus2:-flavor=gcc32dbg
>make
```

4.2.1 Starten eines MPI-Testprogramms

Es soll hier kurz auf die Ausführung von MPI-Programmen über das Globus Toolkit eingegangen werden. Zur Anwendung kommt ein ursprünglich von der MPICH-G2 Website [168] stammendes Programm `ring.c` (der Sourcecode ist auch im Anhang enthalten). Dieses Programm führt zwischen allen gestarteten Instanzen reihum eine Prozess-zu-Prozess-Kommunikation durch. Dies geschieht, indem ein Prozess damit beginnt, eine Nachricht an den Nächsten zu schicken, welcher diese dann wieder an den Nächsten schickt usw., bis der letzte Prozess wieder an den ersten Prozess zurücksendet.

8 verfügbar unter "www-unix.mcs.anl.gov/mpi/mpich/download.html"

Damit lässt sich die Verständigung der verschiedenen MPI-Installationen auf den beteiligten Rechnern gut untersuchen: wenn bei der Kommunikation weniger Prozesse beteiligt sind als gestartet wurden, so sind einige Rechner vermutlich noch nicht richtig konfiguriert.

Das Programm wird nun übersetzt – die erzeugte Binärdatei heißt `ring`:

```
>/home/globus/mpich-1.2.5.2/bin/mpicc -o ring ring.c
```

Vor dem Testlauf sollte man sich vergewissern, ob ein eventuell schon gestarteter Proxy noch läuft. Bei Bedarf muss dann – wie oben besprochen – ein neuer mit ausreichender Restzeit gestartet werden.
Das fertige MPI-Programm lässt sich nun auf zwei Arten ausführen. Einerseits ist es möglich – wie auch bei nicht Grid-spezifischen Versionen von MPI – einfach die Anzahl der zu startenden Instanzen anzugeben, welche dann auf den in den Konfigurationsdateien von MPI angegebenen Maschinen mittels eines dort definierten Schlüssels verteilt werden:

```
>/home/globus/mpich-1.2.5.2/bin/mpirun -np 4 \
/home/globus/ring -v
```

Andererseits kann man auch hier individuelle Jobs in RSL umschreiben. Um die RSL-Dateien nicht komplett neu schreiben zu müssen, ist es möglich, dass MPICH-G2 schon passende Prototypen erstellt, die man dann den aktuellen Bedürfnissen anpassen kann – hier die Datei `ring.rsl`:

```
>/home/globus/mpich-1.2.5.2/bin/mpirun -dumprsl -np 4 \
/home/globus/ring > /home/globus/ring.rsl
```

Wie schon weiter oben beschrieben, können somit Instanzen auf verschiedenen Rechnern gestartet werden, insbesondere kann so für verschiedene Architekturen auch eine andere Binärdatei übermittelt werden. Die in einem so beschriebenen Job gestarteten MPI-Programme befinden sich alle in derselben virtuellen MPI-Umgebung, d. h. sie sind alle in dem Kommunikator `MPI_COMM_WORLD` vertreten.

Eine fertige RSL-Datei lässt sich dann sehr einfach an die MPI-Implementierung übergeben:

```
>/home/globus/mpich-1.2.5.2/bin/mpirun -globusrsl \
/home/globus/ring.rsl
```

Es sollte nun die beschriebene Ring-Kommunikation zwischen allen gestarteten Instanzen stattfinden und abschließend folgende Ausgabe erscheinen:

```
Master: end of trip 1 of 1: after receiving passed_num=4
(should be =trip*numprocs=4) from source=3
```

4.3 Konfiguration des GRIS und GIIS

Sobald die Monitoring and Discovery Services (MDS) auf einem Computer gestartet werden, wird nach der Standardkonfiguration auf diesem automatisch ein GRIS bzw. GIIS eingerichtet, in welchem sowohl Statusinformationen des Globus Toolkits als auch die Hardwarespezifikationen des Computers bis hin zum Betriebssystem verfügbar sind.
Wie schon bei der Installation erwähnt, ist mit der Voreinstellung die Authentifikation deaktiviert. Um dies zu ändern muss in der Datei `$GLOBUS_LOCATION/etc/grid-info-slapd.conf` die Option "anonymousbind" von "yes" auf "no" geändert werden. Um nun die benötigte Zuordnung von Zertifikat-Inhabern zu Benutzerkonten zu erhalten, ist die Datei `$GLOBUS_LOCATION/etc/grid-mapfile` vonnöten, welche genau so aufgebaut ist wie die weiter oben erwähnte `/etc/grid-security/grid-mapfile`.
Hätte man nur einen Computer, wäre das automatisch eingerichtete GIIS (eine Kopie des lokalen GRIS) gleichzeitig das vollständige GIIS des sehr kleinen Grids. Wenn nun mehrere Computer zu einem Grid verbunden werden, ist es sinnvoll, dass alle Rechner ihre GRIS-Einträge an einen zentral erreichbaren Rechner weitergeben, der somit ein GIIS für das ganze Grid darstellt. In sämtlichen Programmen und Anwendungen braucht somit nur noch dieses eine Verzeichnis durchsucht zu werden, um Informationen zu allen im Grid erreichbaren Ressourcen zu erlangen. Dieses Szenario ist für nicht geographisch verteilte Grids durchaus sinnvoll und soll nun weiter untersucht werden.
Jeder Rechner muss dazu die Einträge seines GRIS an das GIIS des Zielrechners

regelmäßig übertragen. Die Datei `$GLOBUS_LOCATION/etc/grid-info-resource-register.conf` beinhaltet die Liste der Verzeichnisdienste, an die das GRIS seine Daten weiterleitet. Der bestehende Eintrag muss nur minimal verändert werden – es reicht, wenn unter der Option "reghn" der Name des neuen GIIS-Rechners eingetragen wird.
Ist der obige Vorgang für jeden Computer im Grid durchgeführt, muss noch das GIIS-Verzeichnis des Zielrechners so konfiguriert werden, dass es die Daten der anderen Rechner annimmt. Dort muss in der Datei `$GLOBUS_LOCATION/etc/grid-info-site-policy.conf` die Option "policydata" angepasst werden. Wenn zum Beispiel alle beteiligten Rechner aus dem Subnetz "informatik.uni-giessen.de" stammen, so könnte der Eintrag folgendermaßen lauten:

```
(&(Mds-Service-hn=*.informatik.uni-giessen.de)(Mds-Service-port=2135))
```

4.4 Grundlagen des LDAP

Wie im 2. Kapitel ausgeführt, werden die Ressourcen und ihre Statusinformationen beim Globus Toolkit 2 in spezielle LDAP-Verzeichnisse eingetragen (GRIS und GIIS). Da dieses Konzept einigen Lesern unbekannt sein dürfte, wird in diesem Abschnitt ein kurzer Überblick gegeben und eine Beispielabfrage des GIIS in der Programmiersprache C vorgestellt.

4.4.1 Das Prinzip der Verzeichnisdienste (Directory Services)

Ein *Verzeichnis* ist eine sehr spezialisierte Datenbank, die für Lesen, Durchsuchen und Suchen optimiert ist. Verzeichnisse beinhalten normalerweise auf Attributen basierende Beschreibungen und bieten anspruchsvolle Möglichkeiten der Datenfilterung. Im Allgemeinen ermöglichen solche Verzeichnisse auch keine komplexen Transaktionen oder Roll-Back-Möglichkeiten, wie man sie zur Eintragsunterstützung in herkömmlichen Datenbanken findet. Neue Eintragungen erfolgen typischerweise nach dem Alles-oder-

Nichts-Prinzip, falls sie überhaupt vorkommen. Man kann also sagen, dass Verzeichnisse hauptsächlich dazu dienen, schnelle Antworten zu sehr umfangreichen Nachschlage- oder Suchoperationen zu erhalten. Um die Reaktionszeit zu verringern bzw. die Verlässlichkeit zu steigern, können sie möglicherweise sogar Daten weiträumig replizieren. Dabei mögen durchaus Inkonsistenzen auftreten, sofern sichergestellt ist, dass zeitweise ein Abgleich durchgeführt wird.
Es gibt viele verschiedene Wege, um einen Verzeichnisdienst zu realisieren. Dies führt dazu, dass die verschiedenen benutzten Methoden auch unterschiedliche Möglichkeiten eröffnen. Zum Beispiel wird eingeschränkt, welche Arten von Daten gespeichert werden können oder ob und wie Sicherheitsmechanismen zum Tragen kommen. So gibt es auch Verzeichnisdienste, die nur auf einem Rechner lokal verfügbar sind (wie zum Beispiel der `finger`-Dienst auf UNIX-Rechnern). Auf der anderen Seite existieren natürlich globale Dienste, die zum Beispiel für das ganze Internet verfügbar sind. Letztere sind häufig sogenannte *verteilte Dienste* – ihre Daten sind also auf mehrere Rechner verteilt, welche in kooperativer Gemeinschaftsarbeit diesen einen Dienst repräsentieren. Bei solchen Diensten wird normalerweise ein einheitlicher so genannter *Namespace* festgelegt, um immer eine gleiche Sicht auf die Daten zu haben – unabhängig vom physikalischen Ort der Daten bzw. dem eigenen Standpunkt. Ein bekanntes Beispiel für einen global verteilten Verzeichnisdienst ist das Internet Domain Name System (DNS).

4.4.2 Das Lightweight Directory Access Protocol (LDAP)

LDAP ist die Abkürzung von Lightweight Directory Access Protocol [170], also zu deutsch etwa ein "Schlankes Verzeichnisdienst-Zugriffs-Protokoll". Es dient dazu, ohne großen Aufwand auf Verzeichnisdienste zuzugreifen. Genaugenommen ist es für auf X.500 beruhende Verzeichnisdienste gedacht und enthält insbesondere eine Untermenge der X.500 Funktionen. Zum Transport wird primär TCP/IP verwendet, aber auch jeder anderer transportorientierter Transferdienst ist möglich.
Das Informationsmodell von LDAP basiert auf sogenannten *Einträgen*. Ein Eintrag ist eine Menge von *Attributen* mit einem global einzigartigen sogenannten *Distinguished Name* (DN). Letzterer kann dazu benutzt werden, den Eintrag direkt zu addressieren –

man könnte ihn also als eine *absolute Pfadangabe* bezeichnen, wenn man einen Vergleich zu einem Dateisystem ziehen will. Jedes Attribut des Eintrages hat einen *Typ* und einen oder mehrere *Werte*. Die Typen sind assoziative Bezeichnungen wie `cn` für `common name` oder `mail` für eine Email-Adresse oder gar `mp3` für eine entsprechende Audiodatei. Das Format der zugehörigen Werte hängt dann natürlich vom Typ ab. Bei `mail` würde man eine Email-Adresse in klassischer Schreibweise erwarten (`ingram.huemmer@math.uni-giessen.de`) und bei `mp3` entsprechend kodierte binäre Daten.
Als Kontrollmöglichkeit dafür, welche Attribute vorhanden sein müssen und welche optional sind, existiert ein Attribut mit dem Namen `objectClass`. Die Werte dieses Attributs legen ein Schema fest, dem jeder Eintrag entsprechen muss.
Die Verzeichniseinträge sind in einer hierarchischen Baumstruktur abgelegt. Ein traditioneller Aufbau einer solchen Struktur wäre eine Abgrenzung anhand von geographischen bzw. organisatorischen Unterschieden. Die Baumstruktur würde dann einem "Hineinzoomen" entsprechen – makroskopische Gebilde sind weit oben im Baum und ganz unten wären die Einträge von Personen oder Rechnern.
Inzwischen wird auch häufig eine Baumstruktur anhand von Internet Domain Names erstellt. Dies erlaubt eine Lokalisierung der Verzeichnisdienste unter Zuhilfenahme des DNS.
Jeder Eintrag hat einen eindeutigen Namen, über den er erreichbar ist. Dieser setzt sich aus dem *Relative Distinguished Name* (RDN) und der Aneinanderreihung aller Vorfahren in der Baumstruktur zusammen. Ein Eintrag meiner universitären Mailadresse könnte also unter der RDN `uid=huemmer` beziehungsweise der DN `uid=huemmer, ou=people, dc=uni-giessen, dc=de` gespeichert sein [115].
Das LDAP definiert Operationen zur Abfrage und zum Ergänzen des Verzeichnisses. Letztere bieten die Möglichkeiten zum Hinzufügen und Löschen von Einträgen aus dem Verzeichnis sowie dem Verändern bzw. Umbenennen von bereits existierenden Einträgen. Hauptsächlich wird das LDAP allerdings benutzt, um Informationen aus dem Verzeichnis zu erfragen. Dabei können Teile des Verzeichnisses nach Einträgen durchsucht werden, welche den speziellen Kriterien eines Filters entsprechen. Die Informationen von jedem Eintrag, auf den der Filter zutrifft, sind somit verfügbar.
Da der LDAP-Verzeichnisdienst auf dem Client-Server Modell beruht, könnte jeder

beliebige Benutzer Informationen aus dem Server abfragen. Um den unauthorisierten Zugriff zu verhindern stellt LDAP die Möglichkeiten zur Verfügung, mit denen sich ein Client gegenüber einem Verzeichnisserver ausweisen und dann auf das gesamte Verzeichnis zugreifen kann. Typischerweise wird dafür eine Authentifizierung mittels X.509 Zertifikaten verwendet.

4.4.3 Testprogramm zur Abfrage des GIIS-Servers

Um die grundlegenden Funktionen eines LDAP Verzeichnisdienstes wie das GIIS oder GRIS eines Globus Servers zu demonstrieren, soll hier kurz ein Beispielprogramm vorgestellt werden. Es lässt sich leicht modifizieren, um auf beliebige Daten zugreifen zu können, aber hier fokussieren wir auf eine Abfrage, welche einem Job-Start vorausgehen könnte. Es werden sämtliche verfügbaren Computer gesucht und elementare Daten – wie Anzahl der Prozessoren und verfügbarer Hauptspeicher sowie freier Festplattenplatz – ermittelt.
Der Sourcecode des Programms `ldaptest.c` befindet sich im Anhang.

Es könnte folgende Ausgabe liefern:

```
Connecting to GIIS (mds-vo-name=site, o=grid) at kafka.informatik.uni-
giessen.de:2135 ...

Entries found for Mds-Host-hn=*: 2
Query found DN (1): Mds-Host-hn=kafka.informatik.uni-giessen.de
Mds-Host-hn for kafka.informatik.uni-giessen.de is kafka.informatik.uni-
giessen.de
Mds-Cpu-Total-count for kafka.informatik.uni-giessen.de is 1
Mds-Cpu-Total-Free-15minX100 for kafka.informatik.uni-giessen.de is 50
Mds-Memory-Ram-Total-freeMB for kafka.informatik.uni-giessen.de is 10
Mds-Memory-Vm-Total-freeMB for kafka.informatik.uni-giessen.de is 400
Mds-Fs-Total-freeMB for kafka.informatik.uni-giessen.de is 1846

Query found DN (2): Mds-Host-hn=fontane.informatik.uni-giessen.de
Mds-Host-hn for fontane.informatik.uni-giessen.de is
fontane.informatik.uni-giessen.de
Mds-Cpu-Total-count for fontane.informatik.uni-giessen.de is 1
Mds-Cpu-Total-Free-15minX100 for fontane.informatik.uni-giessen.de is 59
Mds-Memory-Ram-Total-freeMB for fontane.informatik.uni-giessen.de is 12
Mds-Memory-Vm-Total-freeMB for fontane.informatik.uni-giessen.de is 219
Mds-Fs-Total-freeMB for fontane.informatik.uni-giessen.de is 213
```

Natürlich wurden in dem Programm Beispieldaten benutzt. Das GIIS (DN: `mds-vo-name=site, o=grid`) befindet sich hier auf dem Rechner "kafka.informatik.uni-giessen.de". Bis auf den Rechnernamen wurden die Standardwerte der Globus-Installation verwendet – das Programm kann also als Testprogramm für das installierte GRIS bzw. GIIS verwendet werden, sofern der anonyme Zugang nicht deaktiviert wurde.
Ergänzend soll hier aber noch angemerkt werden, dass sehr viele LDAP-Browser auf unterschiedlichen Basen, wie zum Beispiel PHP-Websites oder Java Applikationen, existieren. Selbst das bekannte UNIX-Mailprogramm "Pine" kann auf Verzeichnisdienste zugreifen, allerdings ist dies in unserem Kontext nur sinnvoll, wenn auch die Mailadressen der Grid-Benutzer im GIIS gespeichert wären.

5. Theoretische Grundlagen zur Primzahlberechnung

5.1 Mathematische Grundlagen

Da das nachfolgende Kapitel die Entwicklung und Beurteilung eines MPI-Programms zur parallelen Berechnung von Primzahlen thematisiert, müssen hier im Vorfeld die theoretischen Grundlagen erarbeitet werden. Während im mathematischen Grundstudium die Primzahlen als Teilgebiet der Zahlentheorie bestenfalls flüchtig behandelt werden, so sind die tiefergehenden Konzepte der Primzahltheorie nur in speziellen weiterführenden Vorlesungen zu erlernen. Schritt für Schritt sollen hier deshalb die Hintergründe von der Definition einer Primzahl bis hin zu Möglichkeiten der Berechnung einer solchen lückenlos zusammengefasst werden. Wie in der Mathematik üblich wird jede Behauptung mit einem Beweis untermauert.

5.1.1 Definition der Teilbarkeit

Auf der Menge der ganzen Zahlen lassen sich die elementaren Operationen wie Addition, Subtraktion und Multiplikation beliebig kombiniert und beliebig oft ausführen, so dass die Ergebnisse selbst wieder ganze Zahlen sind. Bei der Division ist dies nicht mehr ohne weiteres der Fall:
Gibt es für die natürliche Zahl *n* die Produktdarstellung $n = d \cdot q$, wobei *q* und *d* auch natürliche Zahlen sind, so heißt *d* ein *Teiler* von *n*. Man schreibt dafür $d|n$. Als den *komplementären Teiler* bezeichnet man *q*. Klar ist, dass wegen $n = 1 \cdot n$ jedes *n* die trivialen Teiler 1 und *n* besitzt.
Um festzustellen, ob $d|n$ gilt, wird die Division $n \div d$ mit Rest ausgeführt, wobei dadurch die Darstellung $n = q \cdot d + r$ erhalten wird, mit $q \neq 0$ als Quotient und $0 \leq r \leq d - 1$ als Rest. Ist $r = 0$, so gilt offensichtlich $d|n$. Der Quotient kann auch als $q = [n \div d]$ geschrieben werden, wobei dieser Ausdruck $q = [r]$ die größte ganze Zahl $q \leq r$ bedeutet (für positive reelle Zahlen *r*).

5.1.2 Definition von Primzahlen

Nachdem die begrifflichen Grundlagen gelegt wurden, wird nun formal definiert, was eine Primzahl genau ist:
Eine natürliche Zahl $p \neq 1$ heißt *Primzahl*, wenn sie keine nichttrivialen Teiler hat.
Man rechnet die 1 nicht zu den Primzahlen, da jede Zahl diese in beliebiger Potenz enthält und so die folgende Primfaktorzerlegung nicht viel Sinn machen würde. Es ist zu beachten, dass die 2 die einzige gerade Primzahl ist.

5.1.3 Zerlegbarkeit in Primzahlen

Jetzt zeigen wir, dass wir jede natürliche Zahl auch durch ihre Primfaktoren ausdrücken können. Somit wird eine beliebige Zahl durch Primzahlen charakterisiert.

Satz:
Jede natürliche Zahl $n > 0$ ist, ungeachtet der Reihenfolge, eindeutig als Produkt von Primzahlen darstellbar:

$$n = p_1 \cdot p_2 \cdot p_3 \cdots p_r \quad mit\, r > 0$$

Beweis:
Zuerst zeigen wir die Existenz dieser Zerlegung:
Wir spalten den kleinsten Teiler $p_1 \neq 1$ von n ab. Dieser ist eine Primzahl, denn wenn er keine wäre, hätte er selbst einen nicht trivialen Teiler, welcher ja auch wiederum ein Teiler von n ist. Also wäre p_1 nicht minimal – ein Widerspruch zur Annahme.
Im übrig gebliebenen komplementären Teiler lässt sich ebenso wieder eine Primzahl p_2 abspalten $(p_1 \leq p_2)$ und so weiter. Schließlich bricht dieses Verfahren bei einer Primzahl p_r ab.
Somit entsteht folgende Zerlegung:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k} \quad mit\ p_1 < p_2 < p_3 < \ldots < p_k \ \ und\ e_i \geq 1 \ \ für\ i = 1, \ldots, k$$

Nun wird die Eindeutigkeit der Zerlegung gezeigt:
Entweder ist jedes *n* eindeutig zerlegbar oder es existiert eine kleinste Zahl *m* mit zwei verschiedenen Zerlegungen in Primzahlen:

$$m = p_1 \cdot p_2 \cdots p_s = q_1 \cdot q_2 \cdots q_t$$

Es gilt, dass $p_i \neq q_j \ (\text{für } i=1,\ldots,s \text{ und } j=1,\ldots,t)$ ist – denn wäre oBdA $q_1 = p_1$, so hätte $m \div p_1$ zwei Zerlegungen und *m* wäre nicht minimal.
Wir nehmen weiter oBdA an, dass q_1 die kleinste Primzahl ist, welche *m* teilt, und erhalten durch Division folgende Zerlegungen:

$$p_1 = q_1 \cdot Q_1 + r_1 \ , \ p_2 = q_1 \cdot Q_2 + r_2 \ , \ldots , \ p_s = q_1 \cdot Q_s + r_s$$

Durch Multiplikation ergibt sich folgender Ausdruck für *m*:

$$m = p_1 \cdot p_2 \cdots p_s = q_1 \cdot Q_{Restterm} + r_1 \cdot r_2 \ldots r_s = q_1 \cdot q_2 \cdots q_t$$

Das Restglied $R = r_1 \cdot r_2 \cdots r_s$ ist also auch durch q_1 teilbar. Offensichtlich gilt $0 < R < m$. Nach Voraussetzung sind sowohl *R* als auch alle $r_i \ (i=1,\ldots,s)$ eindeutig zerlegbar. Da aber $r_i < q_1 \ (i=1,\ldots,s)$ nach obiger Zerlegung gilt, so kann q_1 als Primfaktor nicht in einer Zerlegung von *R* auftreten.
Dies ist offensichtlich ein Widerspruch, also gibt es nur eine Zerlegung von *n*. Damit ist auch die Eindeutigkeit gezeigt.

5.1.4 Die Anzahl der Primzahlen

Wir wissen noch nichts über die konkrete Anzahl der Primzahlen. Wenn wir zukünftig immer neue Primzahlen suchen wollen, ist folgender Satz von Bedeutung:

Satz:
Die Anzahl der Primzahlen ist unendlich.
Beweis:
Angenommen, es gibt nur endlich viele Primzahlen – sei die größte also *p*.
Bilde nun $P = 2 \cdot 3 \cdot 4 \cdot 5 \cdot 7 \cdots p$ als das Produkt aller Primzahlen.
Die Zahl $P + 1$ ist offensichtlich nicht durch die Primzahlen teilbar, welche kleiner als *p* sind, denn es würde immer der Rest 1 entstehen. Da aber nach dem vorherigen Satz jede

Zahl als Produkt von Primzahlen geschrieben werden kann, ist $P+1$ entweder selbst eine Primzahl oder das Produkt von Primzahlen größer als *p*. Damit ist die Annahme zum Widerspruch geführt und die Behauptung bewiesen.

5.1.5 Das Siebprinzip

Nachdem wir nun wissen, dass es unendlich viele Primzahlen zu entdecken gibt, kann man sich darüber Gedanken machen, wie man sie berechnet. Eine sehr alte und einfache Methode ist das sogenannte *Sieb des Eratosthenes*, womit die Primzahlen bis zu einer natürlichen Zahl *N* berechnet werden:
Man schreibt alle Zahlen von 2 bis *N* hintereinander. Die Primzahl 2 bleibt als erstes stehen und alle Vielfachen der 2 werden als Nicht-Primzahlen gestrichen. Die nächste nicht gestrichene Zahl – eine Primzahl – ist die 3. Jetzt werden wiederum alle Vielfachen von 3 gestrichen. Die nächste übrig gebliebene Zahl ist wieder eine Primzahl und deren Vielfache werden wiederum gestrichen und so weiter. Es wird auffallen, dass die Primzahlen ab der Wurzel von *N* keine Vielfachen mehr in der Liste besitzen und somit das Verfahren beendet ist – die restlichen nicht gestrichenen Zahlen sind Primzahlen.
Ist also eine Primzahl $p>\sqrt{N}$ gefunden, so sind alle Vielfachen $k\cdot p\le N$ schon gestrichen worden, da wegen $k<\sqrt{N}$ diese Vielfachen von *k* schon früher als Vielfache von Primzahlen kleiner *p* gestrichen wurden.

Formal wird in der Zahlentheorie das *allgemeine Siebprinzip* folgendermaßen definiert:

Es seien $S_1,\ldots,S_n$ endliche Mengen. Man definiert die Summe der Mächtigkeiten der Schnitte:

$$\alpha_i = \sum_{1<j_1<\ldots<j_i<n} \left|\cap_{k=1}^{i} S_{j_k}\right|$$

Es werden also für jedes α_i alle möglichen Schnitte mit *i* Mengen der $S_1,\ldots,S_n$ gebildet und deren Mächtigkeiten aufsummiert.

Dann gilt:

$$\left|\cap_{i=1}^{n} S_i\right| = \sum_{i=1}^{n} (-1)^{i-1} \alpha_i$$

Beweis:

Es sei *s* ein beliebiges Element der Vereinigung.

Man zeigt: *s* wird durch die Summe der α_i genau einmal gezählt.

Angenommen, *s* ist in genau *r* der Mengen $S_1, \ldots, S_n$ enthalten (oBdA seien das $S_1, \ldots, S_r$).

Das bedeutet, dass für $i>r$ kein α_i mehr *s* mitzählt.

In α_1 wird *s* genau *r*-mal erfaßt, in α_2 genau $\binom{r}{2}$ -mal, bis hin zu α_r, wo *s* genau $\binom{r}{r}$ -mal erfaßt wird. Es tauchen also alle Binomialkoeffizienten bis auf $\binom{r}{0}$ auf. Da es gleich viele gerade wie ungerade Koeffizienten gibt und diese sich durch das Vorzeichen in der Summe aufheben, gilt:

$$\sum_{i=1}^{r} (-1)^{i-1} \cdot \binom{r}{i} = 0$$

Allerdings wird in der obigen Summe $\binom{r}{0}$ =1 nicht mitgezählt bzw. abgezogen, also beträgt das Gesamtergebnis 1 und nicht 0, was die ursprüngliche Behauptung beweist.

Lemma:

Um eine Zahl *n* auszusieben, die keine Primzahl ist, reicht es aus, die Primzahlen bis zu ihrer Wurzel zu kennen.

Beweis:

Sei $p \geq \sqrt{n}$ ein Primteiler von n. Es gibt mindestens mit $p' \leq p$ einen weiteren Primteiler, denn $p^2 \geq n$, da $p \geq \sqrt{n}$, und n ist eindeutig zerlegbar in $n = p \cdot p' \cdots$ (es kann gelten $p = p'$).

Bleibt noch zu zeigen, dass $p' \leq \sqrt{n}$, und damit wird n als Vielfaches von p' bereits ausgesiebt:

Wegen der Zerlegbarkeit gilt $p \cdot p' \leq n$ und damit auch $p^2 \cdot p'^2 \leq n^2$ und weiter $p^2 \cdot p'^2 \leq n \cdot p^2$, da nach oben $p^2 \leq n$, und so durch Kürzen $p'^2 \leq n$ bzw. $p' \leq \sqrt{n}$ ist. Damit ist die Behauptung gezeigt.

5.1.6 Die Primzahlfunktion

Um die Anzahl der Primzahlen bis zu einer bestimmten Grenze anzugeben, definiert man die *Primzahlfunktion*:
Man bezeichnet mit $\pi(x)$ die Anzahl der Primzahlen bis x.

Diese Funktion ist sehr interessant, aber leider gibt es keinen einfachen Weg, sie zu berechnen, da man bisher auf sehr aufwendige Formeln angewiesen ist. Am einfachsten ist es, in einer Primzahltafel nachzusehen und die Primzahlen zu zählen. Zum Erstellen der Tafel ist es natürlich notwendig, alle Primzahlen bis x zu kennen bzw. zu berechnen. Für große Werte von x gibt es zumindest folgende Schätzung (Primzahlensatz):

$$\lim_{x\to\infty} \frac{\pi(x)\cdot\ln(x)}{x} = 1 \quad \text{also} \quad \pi(x) \approx \frac{x}{\ln(x)}$$

Für diese asymptotische Abschätzung gibt es zwar seit 1948 einen elementaren Beweis durch P. Erdös und A. Selberg, allerdings würde selbst dieser mehrere Dutzend Seiten einnehmen [146].

5.1.7 Beispiel für das Siebprinzip

Es soll nun $\pi(x)$ mittels der Siebmethode berechnet werden:
Setzen wir x=20, dann ist $[\sqrt{x}]=4$. Damit müssen wir mit den beiden Primzahlen 2 und 3 sieben (n=2):

$$\pi(x) = x-\sum_{i=1}^{n}(-1)^{i-1}\cdot\alpha_i+n-1$$

Von den 20 Zahlen ziehen wir also sowohl jedes Vielfache der Primzahlen ab, außer den Primzahlen selbst und der 1, welche ja nicht ausgesiebt wird.
S_1={alle Vielfachen von 2 bis 20}, $|S_1|$ = [20/2] =10
S_2={alle Vielfachen von 3 bis 20}, $|S_2|$ = [20/3] = 6
Und damit $|S_1 \cap S_2|$ = [20/(*kgv*(2,3))]= [20/6] = 3
Es folgt:
$\alpha_1 = |S_1| + |S_2| = 10 + 6 = 16$

$\alpha_2 = |S_1 \cap S_2| = 6$

Und somit:

$\pi(20) = 20 - (16 - 3) + 2 - 1 = 8$

Es gibt also acht Primzahlen bis 20 (2,3,5,7,11,13,17,19). Man beachte, dass mit dieser Methode die Primzahlen nicht explizit berechnet werden, sondern nur über deren Anzahl etwas ausgesagt wird.

Vergleichsweise sagt der Primzahlensatz aus, dass es etwa $20/\ln(20) \approx 7$ Primzahlen bis 20 gibt. Die Abweichung ist also selbst bei einer so geringen Zahl wie 20 nicht sehr gravierend.

5.1.8 Ein alternatives Primzahlsieb

Ergänzend soll hier noch eine alternative Methode vorgestellt werden, Primzahlen bis zu eine Schranke N zu erhalten [147]. Allerdings kann man mit diesem Verfahren nur Primzahlen einer bestimmten Form berechnen:

Man wähle eine Funktion $f : \mathbb{N}_0 \to \mathbb{Z}$, für welche die Bedingung gilt:

$f(x+n) \equiv f(x)\ MOD\, n$ für alle $n \in \mathbb{N}$ und alle $x \in \mathbb{N}_0$

Folgerung:

Sei *i* eine Nullstelle von *f*, also $f(i) = 0$.

Dann gilt auch $j | f(i+j)$ also $f(x) = (x-i) \cdot g(x)$ für ein entsprechendes *g*.

Solche Funktionen mit möglichen Nullstellen sind für die Primzahlsuche offensichtlich uninteressant und sollten nicht verwendet werden.

Bemerkung:

Eine Zahl j kann höchstens einen Primteiler t haben mit $t > \sqrt{j}$.

Die Funktion $f(x) = x^2 + x + 1$ beispielsweise erfüllt offensichtlich die obige Bedingung und hat für $x \geq 0$ keine Nullstellen.

Der Algorithmus verläuft wie folgt:

(1) Analog zum Sieb des Eratosthenes werden alle Werte der Liste

$f(0), f(1), \ldots, f(N)$ im Voraus berechnet.

(2) Jetzt werden nacheinander die Teiler der Einträge wie folgt ermittelt:

Ist *t* ein Teiler von *f*(*i*), dann ist, nach Wahl von *f*, *t* auch ein Teiler von $f(i+k\cdot t)$ für $k \geq 0$.

(3) Berechne $f(i+k\cdot t) \div t^l$ für alle $k \geq 0$ und maximal möglichem *l*.

(4) Gilt nun $|f(j)| < (j+1)^2$, dann enthält *f*(*j*) keinen Primteiler mehr, also entweder ist *f*(*j*)=1 oder *f*(*j*) ist eine Primzahl.

Denn es gilt:

Wegen obiger Bemerkung muss für einen möglichen Primteiler *t* gelten:

$t < \sqrt{f(j)} < (k+1)$. Wegen der Wahl von *f* sind alle Primteiler bis (*k*-1) schon abgespalten worden.

Würde $k|f(k)$ gelten, so auch $k|f(0)$, was schon geprüft wurde.

(5) Im Falle von $|f(j)| \geq (j+1)^2$ muss direkt nach dem Teiler *t* gesucht werden mit $t \geq (j+1)$.

(6)Weiter bei (2).

Der Vorteil dieses Primzahltests ist, dass nicht alle Primzahlen bis zur Wurzel der Zahl *f*(*i*) berechnet werden müssen, um deren Teiler zu finden – es werden die schon ermittelten Teiler der Zahlen *f*(1) bis *f*(*i*-1) mitbenutzt. Allerdings können hier nur eine bestimmte (aber unbeschränkte) Menge von Primzahlen gefunden werden.

Beispiel für ein quadatisches Primzahlsieb:

x	***f*(*x*)=*x*²+*x*+1**	**Ausgesiebte Funktionswerte**
1	3	1 + 3k (hier: 4, 7, 10)
2	7	2 + 7k (hier: 9)
3	13	3 + 13k
4	7 = 21/3	4 + 7k
5	31	5 + 31k
6	43	6 + 43k
7	19 = 57/3	7 + 19k
8	73	8 + 73k
9	13 = 91/7	9 + 13k
10	37 = 111/3	10 + 37k

5.1.9 Der Miller-Rabin-Test

Eine in der Praxis häufig benutzte Methode um (Pseudo-)Primzahlen einer bestimmten Größe zu erhalten ist der Miller-Rabin-Test [35,118]. Genau genommen kann man nur mit einem beliebig kleinen Fehler sagen, dass die gefundene Zahl eine Primzahl ist – deshalb nennt man sie auch *Pseudoprimzahl*. Da dies aber nicht so aufwendig ist, wie echte Primzahlen zu berechnen oder Primzahltabellen bereit zu halten, findet dieser Test gerade bei sehr großen gesuchten Primzahlen seine Verwendung.

Dazu wird der Satz von Fermat benötigt:

Es gilt

$a^{n-1} \equiv 1 \; MOD \; n$ mit $a \in [1,\ldots,n-1]$

genau dann, wenn *n* eine Primzahl ist.

Beweis:

Betrachte $(1 \cdot a) \cdot (2 \cdot a) \cdot (3 \cdot a) \cdots [(n-1) \cdot a] = (n-1)! \cdot a^{n-1}$.

Daraus folgt $(n-1)! \cdot a^{n-1} \equiv (n-1)! \; MOD \; n$.

Durch Teilen von $(n-1)!$ erhält man $a^{n-1} \equiv 1 \; MOD \; n$.

Wenn für eine zufällige Zahl *n* ein *a* aus obigem Intervall gefunden wird, so dass die Äquivalenz nicht gilt, wurde erkannt, dass es sich bei *n* nicht um eine Primzahl handelt.

Der Miller-Rabin-Test verfeinert den Fermat-Test, indem mehrstufig vorgegangen wird:

- Sei oBdA $n>2$ ungerade die zu testende Zahl.
- Da *n* ungerade bilde $n = d \cdot 2^s + 1$ bzw. $(n-1) = d \cdot 2^s$ mit *d* minimal.
- Wähle ein zufälliges $a \in [1,\ldots,n-1]$.
- Wenn $a^{n-1} \equiv 1 \; MOD \; n$ und
- für alle $0 \le r \le s-1$ gilt $a^{d \cdot 2^r} \neq 1 \; MOD \; n$, so ist *n* vermutlich eine Primzahl.

Um den Algorithmus zu verstehen, muss noch erwähnt werden, dass der Restklassen-Ring $\mathbb{Z}_n$ für eine Primzahl *n* ein Körper ist. Demzufolge hat dort jede quadratische Gleichung (für $n>2$) genau zwei verschiedene Lösungen. Für $x^2 \equiv 1 \; MOD \; n$ gibt es

bereits die Lösungen $x=1$ und $x=-1=n-1$. Wenn aber im letzten Schritt des Algorithmus beim sukzessiven Quadrieren von a^d eine weitere Lösung berechnet wird, so kann $\mathbb{Z}_n$ kein Körper sein, also ist *n* keine Primzahl.
Genau genommen erhält man bei einer zufälligen Wahl von *a* mit einer maximalen Wahrscheinlichkeit von 0,5 ein Element, welches bei einer nicht-Primzahl *n* beim obigen Test dieses auch aufdeckt (etwa die Hälfte der Elemente).
Deshalb wird der Miller-Rabin-Test mehrfach durchgeführt (mit jeweils zufälliger Wahl von a). Bei *t* Durchläufen entspricht dies einer Fehleinschätzung für eine Primzahl mit einer immer geriger werdenden Wahrscheinlichkeit von $(0,5)^t$.

5.2 Anwendungsgebiete von Primzahlberechnungen

Seit frühester Zeit bis heute sind in der Mathematik Primzahlen ein ergiebiges Feld der Forschung. Natürlich gibt es in der Mathematik selbst viele Stellen, an denen Primzahlen wichtig sind, wie zum Beispiel bei den Sylow-Sätzen der Algebra. Aber auch darüber hinaus finden sie in immer mehr Gebieten der Mathematik praktische Anwendung. Insbesondere soll hier auf das relativ neue Gebiet der Kryptographie eingegangen werden.

5.2.1 Die Goldbachsche Vermutung

Als einführendes Beispiel für die Berechnung beliebig großer Primzahlen soll die vielzitierte Goldbachsche Vermutung dienen.
1742 teilte der preußische Mathematiker Christian Goldbach seinem auch heute noch sehr berühmten Kollegen Leonhard Euler folgende Vermutung in einem Brief mit:
Jede gerade Zahl, welche größer als zwei ist, läßt sich als Summe von zwei Primzahlen ausdrücken.
In den nachfolgenden Jahrhunderten wurde häufig versucht diese Aussage zu beweisen, denn für alle durchgerechneten Beispiele schien sie zu stimmen.
Im Zeitalter der vernetzten Computer wurde dieses natürlich fortgeführt, denn auch wenn

noch so viele Beispiele stimmen mögen, so ist die Aussage nicht bewiesen. Aber wenn auch nur ein einziges Gegenbeispiel gefunden würde, wäre die Aussage widerlegt.
So hielt 1998 der Mathematiker Jörg Richstein vom Institut für Informatik an der Universität Gießen den weltweiten Rekord [36], indem er die geraden Zahlen bis 400 Billionen untersucht hatte [175]. Diese Berechnungen erfordern natürlich auch entsprechende Primzahlen derselben Größenordnung.
Jenes Beispiel zeigt insbesondere, dass der Umgang mit großen Zahlen nur noch mit dem Computer möglich ist. So wurde etwa bis 1953 die Goldbachsche Vermutung nur bis zur Zahl 100000 untersucht [146].

5.2.2 Anwendung in der Kryptographie

Die Zahlentheorie ist seit Beginn der vierziger Jahre des zwanzigsten Jahrhunderts aus der angewandten Mathematik nicht mehr wegzudenken. Damals wurde die Entwicklung von Verschlüsselungsverfahren zu militärischen Zwecken und insbesondere deren Entschlüsselung auf der Gegenseite mittels rein mathematischer Hilfsmittel erstmals durchgeführt. War es vorher noch ein Gebiet der Linguistik, so wurde es zunehmend von Mathematikern übernommen. Seitdem werden Verfahren entwickelt, welche auch gegen rein mathematische Angriffe robust sind.
Generell teilt man Verschlüsselungsverfahren in zwei Kategorien ein: die s*ymmetrischen* und die a*symmetrischen Verfahren*. Die symmetrischen Verfahren sind die bekannteste und älteste Klasse, denn sie setzen voraus, dass sowohl der Verschlüsseler als auch der Entschlüsseler den gleichen geheimen Schlüssel benutzen, also beide ein gemeinsames Geheimnis besitzen. Der essentielle Nachteil dieser Methode ist relativ klar: Die beteiligten Parteien müssen zu einem Zeitpunkt dieses Geheimnis festlegen, also sich persönlich treffen, denn das Geheimnis ließe sich nicht über eine verschlüsselte Botschaft übertragen, weil man dafür wiederum einen gemeinsamen Schlüssel bräuchte usw. Dieses Problem beheben die asymmetrischen Verfahren. Bei diesen hat jeder Teilnehmer einen eigenen geheimen und einen öffentlichen Schlüssel. Natürlich hängen diese mathematisch zusammen, aber es ist praktisch unmöglich, aus dem öffentlichen Schlüssel den privaten Schlüssel zu berechnen. Da jeder seinen öffentlichen Schlüssel an andere Teilnehmer

ungeschützt weitergeben kann bzw. muss, erlaubt dieses System eine höhere Flexibilität. Tatsächlich wurde die Existenz eines solchen Verfahrens lange Zeit für unmöglich gehalten, bis 1978 der erste Algorithmus dazu veröffentlicht wurde – der RSA [1].

5.2.2.1 Definition der Eulerschen Phi-Funktion

Für die genauere Betrachtung des nachfolgenden Beispiels zur Relevanz der Primzahlen benötigen wir noch diese Definition:

Für $n \in \mathbb{N}$ definiert man die Eulersche Phi-Funktion folgendermaßen:

$$\varphi(n) = |\{k \in \mathbb{N} | 1 \le k \le n,\ ggT(n,k) = 1\}|$$

Beispiel:

$$\varphi(15) = |\{1,2,4,7,8,11,13,14\}| = 8$$

Insbesondere gilt für ein Produkt von zwei Primzahlen $n = p \cdot q$ natürlich:

$$\varphi(n) = (p-1) \cdot (q-1)$$

In diesem Beispiel gilt also wegen $15 = 3 \cdot 5$ die Berechnung

$$\varphi(15) = (3-1) \cdot (5-1) = 8 \quad .$$

5.2.2.2 Die Funktionsweise des RSA

- Die Schlüsselerzeugung:
  Man wählt zwei verschiedene Primzahlen *p*, *q* und bildet das Produkt $n = p \cdot q$ sowie die Eulersche Phi-Funktion $\varphi(n) = (p-1) \cdot (q-1)$.
  Dann sucht man eine natürliche Zahl *e* mit der Eigenschaft $ggT(e, \varphi(n)) = 1$.
  Schließlich bestimmt man die natürliche Zahl *d*, so dass $e \cdot d \equiv 1 (mod\ \varphi(n))$ gilt.
  Der öffentliche Schlüssel ist nun das Paar (*e*,*n*) und der private Schlüssel die Zahl *d*.
- Die Verschlüsselung:
  Sei *m* eine Nachricht und (*e*,*n*) der öffentliche Schlüssel des Empfängers,
  dann wird die verschlüsselte Nachricht *c* mittels der Potenzierung $c = m \cdot e (mod\ n)$
  gebildet.

- Die Entschlüsselung:
  Sei *d* der eigene private Schlüssel (also passend zu *e*), dann wird die Nachricht durch die Potenzierung $m = c^d \,(mod\ n)$ wiedergewonnen.

Um zu beweisen, dass durch die Entschlüsselung auch tatsächlich die ursprüngliche Nachricht wiederhergestellt wird, benötigen wir noch den Satz von Euler.

5.2.2.3 Der Satz von Euler

Für die natürlichen Zahlen *m* und *n* sowie für alle natürlichen $k \geq 1$ mit $m < n = p \cdot q$ und *p* sowie *q* Primzahlen gilt:

- Sind *m* und *n* teilerfremd, so gilt:
  $m^{\varphi(n)} \,(mod\ n) \equiv 1$ bzw. $m^{k \cdot \varphi(n)} \,(mod\ n) \equiv 1$
- Durch Multiplikation mit *m* gilt allgemeiner:
  $m^{\varphi(n)+1} \,(mod\ n) \equiv m$ bzw. $m^{k \cdot \varphi(n)+1} \,(mod\ n) \equiv m$

Beweis:

Wir beweisen zunächst für jede Primzahl *p* mit Induktion nach *a*:

$a^p \,(mod\ p) \equiv a \;\; für\ alle\ a < p$

Induktionsverankerung: $a = 1$

$1^p \,(mod\ p) \equiv 1$ ist klar.

Induktionsschritt: $a \to a+1$

Anwendung des binomischen Lehrsatzes führt zu:

$$(a+1)^p = a^p + \binom{p}{1} \cdot a^{p-1} + \binom{p}{2} \cdot a^{p-2} + \ldots + \binom{p}{p-1} \cdot a + 1$$

Da $\binom{p}{i}$ für $1 \leq i \leq p-1$ durch *p* teilbar ist, sind diese Terme modulo *p* gleich 0.

Also bleibt:

$(a+1)^p \,(mod\ p) \equiv a^p + 1 \,(mod\ p) \equiv a+1$ (nach Induktion)

Durch Division durch *a* folgt:

$a^{p-1} \,(mod\ p) \equiv 1$ bzw. $p \mid a^{p-1} - 1$

Sei nun $m<n$ eine zu *n* teilerfremde natürliche Zahl.

Zeige $m^{\varphi(n)}(mod\ n)\equiv 1$ bzw. $n|m^{(p-1)\cdot(q-1)}-1$:

Nun gilt nach obiger Induktion: $p|m^{p-1}-1$ und damit insbesondere auch:

$p|m^{(p-1)\cdot(q-1)}-1$, da *m* teilerfremd zu *n*, also ist insbesondere $n\neq p$.

Analog gilt natürlich auch Entsprechendes für eine Primzahl *q*.

Zusammen folgt also:

$n = p\cdot q|m^{(p-1)\cdot(q-1)}-1$

5.2.2.4 Beweis der korrekten Entschlüsselung beim RSA

Mit dem Satz von Euler gilt folgende Äquivalenz:

$m=m^1\equiv m^{k\cdot\varphi(n)+1}(mod\ n)$

Durch die Konstruktion der Schlüssel gilt wegen $e\cdot d\equiv 1(mod\ \varphi(n))$ weiter:

$m^{k\cdot\varphi(n)+1}(mod\ n)\equiv m^{e\cdot d}(mod\ n)\equiv(m^e(mod\ n))^d(mod\ n)$

$\equiv c^d(mod\ n)$

5.2.2.5 Ein Beispiel des RSA

Sei $p=3$ und $q=5$ und damit $n=15$.

Damit ergibt sich $\varphi(15)=8$, wie weiter oben berechnet, und wir wählen passend

$e=3$ und $d=3$.

Somit sind die Anforderungen $ggT(3,8)=1$ und $3\cdot 3\equiv 1(mod\ 8)$ erfüllt.

Jetzt haben wir den öffentlichen Schlüssel (7,15) und den privaten Schlüssel 3.

- Verschlüsselung:

  Wir verschlüsseln die Nachricht $m=13$ zu $13^3(mod\ 15)=2197(mod\ 15)=7=c$.
- Entschlüsselung:

  Jetzt entschlüsseln wir wieder $c=7$ zu $7^3(mod\ 15)=343(mod\ 15)=13=m$.

Um sinnvolle Nachrichten verschlüsseln zu können, bietet es sich an, jedes Zeichen der Nachricht als Element des 16-bittigen Unicode anzusehen und so zu verschlüsseln. Damit

ergibt sich natürlich die Anforderung, dass n größer als 2^{16} sein muss, also sind die Primzahlen p und q entsprechend zu wählen.

5.2.2.6 Die Sicherheit des RSA und die Primzahlen

Um die Nachricht zu entschlüsseln muss man den privaten Schlüssel d berechnen. Dieser steht in Beziehung zu den bekannten Größen e und n durch $e \cdot d \equiv 1 \,(mod\, \varphi\,(n))$.
Es gelten bekanntlich die zwei Gleichungen:

- $n = p \cdot q$
- $\varphi\,(n) = (p-1) \cdot (q-1)$

Wenn man also n faktorisieren kann (man erhält p und q), erhält man ebenso $\varphi(n)$ und kann d berechnen.
Also muss p und q groß genug gewählt werden, damit n praktisch nicht zu faktorisieren ist.
Ergänzend soll hier aber gesagt werden, dass es (noch) nicht bewiesen ist, dass die Sicherheit des RSA nur von dem Problem der Faktorisierung abhängt – in der nicht-mathematischen Welt wird allerdings davon ausgegangen. Somit liegt die Sicherheit des RSA bei der Wahl von großen Primzahlen. Allerdings ist es in der Praxis nicht unbedingt von Vorteil, sehr große Zahlen zu wählen, denn das Potenzieren bei der Verschlüsselung bzw. Entschlüsselung ist sehr zeitaufwendig. Man muss also abwägen, welche minimale Größenordnung wegen der Sicherheit festgesetzt wird, bzw. welche maximale Größenordnung aufgrund der Rechnerleistung praktikabel ist – insbesondere im Hinblick auf die Leistungsfähigkeit von spezialisierten Prozessoren auf Scheckkarten.
Früher wurden Größenordnungen von 256 Bit bzw. 512 Bit als sicher angegeben, heutzutage benutzt man Primzahlen in einem Bereich von mindestens 1024 Bit. Durch die technische Entwicklung sind einzelne Rechneranlagen zwar immer leistungsfähiger geworden, aber insbesondere massives Parallelisieren mittels großer verteilter Netze (wie sie auch Grids zur Verfügung stellen) führte dazu, dass selbst Zahlen von 256-bittiger Größe in wenigen Jahren oder gar Monaten faktorisiert werden konnten. Da es wünschenswert ist, dass asymmetrische Schlüssel auch bei rasanter Entwicklung über viele Jahrzehnte hinweg nicht kompromittiert werden, ist eine hohe Wahl von 1024 Bit

oder mehr durchaus gerechtfertigt.
Man kann sich natürlich fragen, ob denn für jede gewählte Größenordnung genügend Primzahlen existieren. Wenn vergleichsweise wenige Primzahlen existieren würden, dann könnte man anstatt zu faktorisieren einfach jedes mögliche Produkt zweier Primzahlen bilden und so das öffentliche n erhalten – damit hätte man wegen der Eindeutigkeit der Primfaktorzerlegung von n die beiden Primzahlen gefunden. Glücklicherweise können wir aus dem Primzahlensatz schließen, dass sich dies nicht lohnt. Selbst für 256-bittige Primzahlen gibt es $\pi(2^{256}) \approx 2^{248}$ Primzahlen.
Ergänzend soll hier noch gesagt werden, dass nicht alle Verfahren in der asymmetrischen Kryptographie auf Primzahlen beruhen. Die Sicherheit einiger Verfahren basiert beispielsweise auf der schwierigen Umkehrbarkeit des diskreten Logarithmus.

5.3 Grundlagen der Informatik zur Primzahlberechnung

In diesem Abschnitt sollen kurz die Möglichkeiten der informatischen Erfassung von Primzahlen vorgestellt werden. Anschließend wird die parallele Berechnung von Primzahlen näher erläutert und zum Abschluss des Kapitels werden die Unterschiede in den Anforderungen für Primzahlberechnungen bis zu einer festen Zahl bzw. Berechnungen ohne konkrete Abbruchbedingung näher untersucht.

5.3.1 Primzahldarstellungen

Bei der Entwicklung von Programmen zur Primzahlberechnung für didaktische Zwecke, wie etwa in der Schule oder auch an der Universität, wird normalerweise bis zu einer Obergrenze von 2^{32} gerechnet, was der üblichen Kapazität eines Long Integer entspricht, für welches sämtliche Rechenoperationen in den Standardbibliotheken zur Verfügung stehen. Einige Compiler, wie zum Beispiel GNU C, kennen noch 64-Bit Variablentypen und bieten auch entsprechende Bibliotheken an. Dieser Ansatz sieht also vor, dass eine Primzahl in einer elementaren Variable gespeichert wird und somit mit den schon vorhandenen Funktionen gerechnet werden kann.

Wie die praktischen Beispiele weiter oben aufzeigten, würde diese Methode selbst für einen 256-Bit RSA-Schlüssel nicht ausreichen – sie ist also für praktische Anwendungen nicht ausreichend.
Eine relativ naheliegende Lösung ist natürlich die Kaskadierung von 32-Bit-Variablen. So könnte man mit 32 Variablen von 32-bittiger Größe einen 1024-Bit-RSA-Schlüssel darstellen.
Wenn man eine maximale Größe anvisiert – wie hier die 1024 Bit – so ist es durchaus effektiv, eine Zahl bzw. Primzahl mittels eines Feldes mit 32 Einträgen von 32-bittigen Variablen zu erfassen. Da Felder normalerweise sehr platzsparend im Hauptspeicher angelegt werden und diese Datenstruktur in den Bibliotheken im Allgemeinen gut unterstützt wird, ist dieses Vorgehen sicherlich eine gute Wahl.
Der Nachteil an festen Feldern ist die mangelnde Fexibilität. Wenn viele Berechnungen mit wesentlich kleineren Zahlen durchgeführt werden und nur wenige wirklich diese hohe Kapazität ausnutzen oder gar nicht richtig abzuschätzen ist, welche Kapazität im laufenden Programm wirklich benötigt wird, sollte mit dynamischen Datenstrukturen gearbeitet werden. Beispielsweise wäre eine verkettete Liste von 32-Bit Variablen möglich, welcher die aktuelle Länge voransteht. Verkettete Listen können allerdings je nach System zu starken Speicherfragmentierungen führen und der Zugriff ist etwas komplizierter und damit auch langsamer.
In jedem der beiden obigen Fälle muss eine eigene Arithmetik für die neu geschaffene Zahlendarstellung geschrieben werden. Die elementaren Rechenoperationen sind noch überschaubar, problematischer wird es bei komplexeren Funktionen wie der Wurzelfunktion oder dem Logarithmus. Um möglichst effektive und schnelle Operationen zu gewährleisten, bieten sich entsprechende Verfahren aus der Numerik an. Bei der Benutzung von Hochsprachen ist es naheliegend, von den elementaren Variablentypen, wie zum Beispiel einer Long Integer, nur 31 Bit der 32 Bit zu benutzen, da viele Bibliotheksfunktionen das höchstwertigste Bit als Vorzeichen interpretieren und so unerwartete Rechenfehler auftreten können.

5.3.2 Speichern von Primzahlen

Bei Primzahlberechnungen wie dem Sieb des Eratosthenes ist es notwendig, eine große Menge an Zahlen zu speichern, um die Nicht-Primzahlen herauszustreichen.
Eine einfache Methode macht sich zunutze, dass auf Zahlen einer solchen Tabelle normalerweise nicht willkürlich zugegriffen wird, sondern – zum Beispiel für eine Ausgabe – die Zahlen nacheinander gelesen bzw. geschrieben werden.
Da wir wissen, dass die 2 die einzige gerade Primzahl ist, brauchen wir eigentlich nur noch jede ungerade Zahl speichern. Dies geschieht, indem wir jede ungerade Zahl – von 1 beginnend – als ein Bit speichern, wobei eine *Eins* eine "Primzahl" und eine *Null* entsprechend "keine Primzahl" bedeutet.

Beispiel:
0111011011 entspricht somit 1,3,5,7,9,11,13,15,17,19 und 3,5,7,11,13,17,19 sind davon Primzahlen.

Wie man sieht, ist es relativ einfach, die Zahlen zu rekonstruieren, denn der *x*-te Eintrag entspricht der Zahl $x \cdot 2 - 1$. Allerdings muss man auch noch die Fallunterscheidung treffen, ob auf die Zahl 2 zugegriffen werden soll, sonst würde sie als Primzahl wegfallen, denn sie taucht in dieser Darstellung nicht auf.
Diese Idee, Vielfache von kleinen Primzahlen von vornherein nicht zu speichern, lässt sich natürlich beliebig erweitern und spart so weiteren Speicherplatz ein.

Beispiel:
Neben Vielfachen der 2 werden auch Vielfache der 3 nicht mehr gespeichert:
1111111011 entspricht 5,7,11,13,17,19,23,25,29,31.

Beim Auslesen einer solchen Primzahlentabelle darf man natürlich nicht vergessen, dass noch 2 und 3 Primzahlen sind.
Es lässt sich mit dieser Methode also auf Kosten einer erhöhten Komplexität etwas Speicherplatz einsparen. In der Praxis werden daher nicht mehr als die Vielfachen von 2, 3 und 5 ausgeblendet.

Eine andere Möglichkeit Primzahlen (und nur diese) zu speichern ist, die Lücken zwischen den Primzahlen zu berechnen. Die Größen dieser Lücken wachsen erheblich langsamer als die Primzahlen selbst, so beträgt beispielsweise die größte Lücke bis zur Primzahl 2614941711251 nur 652 Zahlen [144].

Beispiel:
Zwischen den Primzahlen 2,3,5,7,11,13,17,19,23,29,31 bestehen die Lücken 1,2,2,4,2,4,2,4,6,2.

Da bis auf die Lücke zwischen den Primzahlen 2 und 3 alle Lücken gerade Zahlen sind, kann man natürlich auch nur die Hälfte des Betrages der Primzahllücke speichern, was allerdings keine große Ersparnis bringt: Es wird ein Bit weniger zur Speicherung einer Lücke verwendet.

Zum Abschluss dieser Darstellungen ist noch anzumerken, dass sich erstere "0/1"-Methode zum Arbeiten im Hauptspeicher gut verwenden lässt, zum Beispiel bei Durchführung eines Siebalgorithmus. Letztere Darstellung der Primzahlen ist als endgültige Speicherung auf einem Datenträger sinnvoll, da man mit ihr nicht mehr weiterarbeiten kann und sie nur dazu dient, die Primzahlen konsekutiv einzulesen.

5.3.3 Möglichkeiten der parallelen Berechnung von Primzahlen

Da man sich fast beliebig viele Möglichkeiten ausdenken kann, um Primzahlen zu berechnen, soll hier auf grundlegende Prinzipien eingegangen werden.
Bei effektiver paralleler Berechnung sollten alle Prozessoren möglichst gleichzeitig arbeiten, daher sollten ihre jeweiligen Ergebnisse nicht für die Berechnung anderer notwendig sein. Für einen allgemeinen Ansatz könnte auch die Betrachtung der beliebigen Skalierbarkeit interessant sein, da man die tatsächlich verfügbaren Ressourcen nicht vorhersehen kann.
Als Ausgangspunkt betrachten wir die Menge der Zahlen von 1 bis N und wollen gleichzeitig n Prozessoren arbeiten lassen.

Hier zwei Möglichkeiten:

- Eine sehr einfache Methode wäre, die Ausgangsmenge in $N \div n$ große Teilmengen zu unterteilen und so mit einem einfachen Algorithmus jeden Prozessor die ihm zugewiesene Menge nach Primzahlen zu durchsuchen. Dies würde aber nur bei sehr großem *n* etwas bringen, denn viele Informationen würden zur Filtrierung von jedem Prozessor mehrfach berechnet werden.
- Ein weiterer Ansatz ist die teilweise Parallelisierung. So würde ein Hauptprozess die Primzahlen bis $\sqrt{N}$ berechnen und bereits gefundene Primzahlen an weitere Prozesse melden, welche dann die Zahlen von $\sqrt{N}$ bis *N* sieben. Dies ist eine parallele Umsetzung des Siebes von Eratosthenes und berechnet keinerlei redundante Informationen. Die Ausnutzung der verfügbaren Anzahl *p* an Prozessen ist dabei natürlich nicht optimal, denn es kann bei großem *p* etwas dauern, bis alle Prozesse beschäftigt sind. Wenn allerdings erst einmal alle Prozesse mit Primzahlen versorgt sind, ist anzunehmen, dass der Hauptprozess wieder neue Primzahlen bereit hält, wenn ein anderer fertig geworden ist, denn er hat einen viel kleineren Bereich als diese zu sieben.

Ein zusätzlicher Unterschied zwischen den beiden obigen Verfahren ist, dass das erste problemlos mit verteiltem Speicher zurechtkommt, während letztere Methode besser bei der Verwendung von gemeinsamem Speicher geeignet ist.

5.3.4 Unterschiede von unbeschränkten zu beschränkten Berechnungen

Die obigen Beispiele gehen von einer Berechnung bis zu einer maximalen Grenze aus. Ohne eine solche Schranke ist der Ansatz für eine effiziente parallele Berechnung schwierig. Selbst nicht parallele Ansätze wie das Sieb des Eratosthenes gehen von einem vorgegebenen Bereich aus, welcher nach Primzahlen gesiebt wird.
Ein einfacher Ansatz wäre es, die oben zuerst genannte Methode zu modifizieren. Demnach würde jeder der *p* Prozesse die Zahlen $k \cdot p + Prozessnummer$ mit $k \geq 0$ nach Primzahlen überprüfen. Dies würde eine optimale Auslastung der zur Verfügung stehenden Prozesse bedeuten. Jedoch bleibt bei diesem Vorgehen kein Platz für Synergieeffekte der verschiedenen Berechnungen und benötigt einen gemeinsamen

Speicher.
Eine alternative Vorgehensweise wäre es, eine temporäre Schranke zu setzen, bis zu der die Primzahlen mit bekannten Algorithmen zu berechnen und dann eine neue obere Schranke zu bestimmen, zu welcher die Berechnung fortgesetzt wird. Am geeignetsten für eine solche neue Schranke N wäre das Quadrat der alten Schranke A, denn es gilt dann $\sqrt{N} = A$. Damit könnte man die zuvor berechneten Primzahlen zum Beispiel für das Sieb des Eratosthenes verwenden. Allerdings ist dabei zu beachten, dass der Speicherverbrauch auf Dauer zu hoch wird, um die Listen in einem lokalen Seicher zu halten. Dies kann zu weiteren Problemen führen, was eine andere Wahl der temporären Schranke nötig machen könnte.
Solche Berechnungen ohne festes Ende sind einerseits zwar sehr schwierig effizient durchzuführen, aber auf der anderen Seite sind es gerade diese, welche in der Praxis interessant sind – wie zum Beispiel für die Überprüfung der Goldbachschen Vermutung (siehe Abschnitt 5.2.1).

5.4 Algorithmenentwurf für Grids

Es soll hier noch einmal explizit untersucht werden, welche Anforderungen an einen Algorithmus zur parallelen (Primzahl-)Berechnung für Grids gestellt werden.
Bei Grids können wir von einer heterogenen Architektur ausgehen (wie verschiedene Prozessorgeschwindigkeiten) und prinzipiell verteiltem Speicher. Je nach Grad der Virtualisierung der Ressourcen könnte man auch mehrere "Recheneinheiten" mit der gleichen Geschwindigkeit anfordern oder einen beliebig großen Speicherblock. Letzterer könnte physikalisch aber auch auf verschiedenen Rechnern vereilt sein. Es ist also schwierig einen Algorithmus für ein virtuelles System zu entwerfen, denn die Abbildung auf die reale Hardware kann vermeintliche Optimierungen vereiteln.
Wenn man trotzdem spezielle Anpassungen vornehmen will, muss man also entweder direkt auf die untersten Schichten der Grid-Middleware zugreifen (sehr proprietär), oder Portierungen von Laufzeitumgebungen mit bekannter Verhaltensweise benutzen (wie beispielsweise MPICH-G2). Letzteres schränkt allerdings die Möglichkeiten der Algorithmusmodellierung auf die verfügbare API der Laufzeitumgebung ein, welche

normalerweise nur einen Bruchteil der spezielen Grid-API bietet. Man kann also sagen, dass im Algemeinen für Grids eine Anpassung des Algorithmus an eine erwartete Hardwarestruktur nicht ohne weiteres möglich (und durch die zunehmende Virtualisierung der Ressourcen im Grid auch nicht gewollt) ist. Andererseits allerdings läßt sich durch die Auswahl der Ressourcen die für den Algorithmus benötigte Umgebung erzeugen (wie verteilter oder gemeinsamer Speicher).
Ein weiteres Problem ist die Dynamik des Grids. Neben den (zeitweisen) Ausfallmöglichkeiten einzelner Ressourcen während der Laufzeit können auch mehr Ressourcen dauerhaft zur Verfügung stehen als zu Beginn der Programmausführung. Es ist also sinnvoll, darauf zu achten, dass ein paralleler Algorithmus sich – zum Beispiel über die Anzahl der benutzten Prozesse – an diese Veränderungen anpassen kann.
Bei dem Problemen wie der Primzahlberechnung würde letzteres bedeuten, dass möglichst wenig Informationen bei einzelnen Prozessen liegen sollten. Dies würde für eine Lösung mit gemeinsamem Speicher sprechen, da der Ausfall von nur einem beliebigen Prozess mit seinem Speicher wahrscheinlicher ist, als dass der global genutzte spezielle Speicherbereich ausfällt – es sei denn man würde entsprechende Probleme im Algorithmus vorsehen und ausgleichen (wünschenswert).
Leider gibt es zur Zeit noch keine Laufzeitbibliotheken für Grids, welche die oben angesprochenen Probleme der Dynamik für das Programm transparent halten bzw. dieses bei deren Bewältigung unterstützen. Es ist aber anzunehmen, dass in absehbarer Zeit solche Bibliotheken für verteilte Systeme (beispielsweise Phoenix [136]) auch an Grids angepasst bzw. fertiggestellt werden (beispielsweise Ibis [86]).

6. Programmentwicklung unter MPI und Globus

Dieses Kapitel beschreibt die exemplarische Entwicklung und Auswertung eines MPI-Programms, welches unter der in den vorigen Kapiteln beschriebenen MPICH-G2 Implementierung von MPI als Grid-Applikation über das Globus Toolkit ausgeführt wird. Dabei wird sowohl auf MPI-spezifische Aspekte als auch auf Anpassungen an das Globus Toolkit und die tatsächlich verwendete Hardware besondere Rücksicht genommen. Als Programmiersprache kommt – wie auch in den Beispielen zuvor – die allgemein bekannte Sprache C zum Einsatz. Da im Laufe des Grundstudiums C (oder das objektorientierte C++) bzw. das syntaxverwandte Java gelehrt wird, sollten keine Verständnisprobleme bei den Datentypen oder dem generellen Programmtext auftreten.

6.1 Entwicklung des Primzahlprogramms

In diesem Abschnitt wird die Idee eines Programms zur parallelen Primzahlberechnung vorgestellt und der verwendete Algorithmus formuliert. Dabei wird insbesondere auf die im 5. Kapitel vorgestellten Konzepte Rücksicht genommen.

6.1.1 Formulierung der Aufgabenstellung

Das Ziel dieses Programmierprojektes ist die Entwicklung eines MPI-Programms auf Basis der Sprache C, mittels dessen große Primzahlen berechnet werden können. Da das Programm in einer Grid-Umgebung des Globus Toolkits ausgeführt werden soll, können auch dessen Schnittstellen verwendet werden – insbesondere bei Grid-weiten Datenzugriffen. Um unnötige Komplexität zu vermeiden sowie die Übersichtlichkeit und die damit verbundene Nachvollziehbarkeit zu erhöhen, sollten zu komplexe und spezialisierte Funktionen vermieden und deshalb möglichst grundlegende Bibliotheken benutzt werden, wie sie zum Beispiel im 4. Kapitel vorgestellt wurden.

Das Programm soll die Möglichkeit besitzen, Primzahlen bis zu einer praktisch beliebigen Höhe zu berechnen. Keine die beiden im vorigen Kapitel vorgestellten Anwendugsgebiete – Überprüfung der Goldbach'schen Vermutung und die großen Primzahlen für sichere RSA-Schlüssel – kann spezielle Primzahlen in de Form brauchen, wie sie in 5.1.8 vorgestellt wurden. Prinzipiell wären solche Primzahlen für den RSA aus mathematischer Sicht nicht schlecht, aber wenn jemand erfährt, wie diese Primzahlen berechnet wurden, dann bräuchte er das Produkt nur nach solchen Zahlen faktorisieren. Das wäre also ein weiteres Geheimnis zu dem privaten Schlüssel – eine offensichtliche Schwächung der Sicherheit. Also bleibt nur das Sieb des Eratosthenes übrig.
Zum Erstellen von Benchmarks soll die Option bestehen bis zu einer vorgegebenen Grenze Berechnungen durchzuführen und die verbrauchte Zeit dafür zu messen.
Zur Steuerung des Programmablaufs sollte ein Prozess die Kontrolle auf Daten und Ergebnisse ausüben. Somit kann dieser auch die Koordination der Daten und die Zeitmessung beim Benchmark übernehmen.
Es sollte möglichst eine gute Anpassung an vorhandene Rechenkapazität erreicht werden, also sollte eine beliebige Anzahl von Rechnern bzw. Prozessoren gleichzeitig einbezogen werden können. Das bedeutet natürlich auch, dass mehrere Prozesse auf einem Prozessor laufen können – aber es sollten keine festgelegten Sprünge von benötigten Prozessen vorgegeben sein – wie etwa nur Potenzen der Zahl 2.

6.1.2 Anforderungen an die Datenstrukturen

Durch die spezielle Hardware-Konfiguration der Testrechner ergeben sich noch technische Einschränkungen, welche in den Programmentwurf einfließen müssen.
Ein Problem besteht darin, dass die Prozessoren nur mit etwas über einhundert Megahertz getaktet sind, was jeglichen unnötigen Overhead von vornherein verbietet. Es ist aber davon auszugehen, dass die Prozesse zum Erledigen ihrer Teilaufgaben signifikant mehr Zeit benötigen als zur Kommunikation mit dem Hauptprozess, so dass sich eine nicht ganz optimale Kommunikation (wie zum Beispiel blockierende Kommunikationsfunktionen) nicht so stark auf die Performance auswirkt.
Ein weiteres grundlegendes Problem ist die geringe Ausstattung an Hauptspeichern der

verwendeten Testrechner. Die 32 bzw. 64 Megabyte RAM werden in der Regel schon durch das Betriebssystem mit der grundlegenden Softwareausstattung wie der Netzwerkverwaltung und das Globus Toolkit verbraucht. Die Prozesse sollten also mit einem Minimum an RAM auskommen, was große Zahlentabellen im RAM ausschließen. Der gewählte Lösungsansatz beinhaltet den Zugriff auf frühere Ergebnisse, welche auf einem Datenträger gespeichert sind. Diese Methode gewährleistet zwar eine kontinuierliche Performance auch bei sehr großen Zahlen, aber die Zugriffsgeschwindigkeit hängt sehr stark sowohl von dem Cache-System des Datenträgers als auch von dem des Betriebssystems ab. Auf jeden Fall wird so der benutzte Speicher eines Prozesses minimiert. Bei zu hohem Speicherverbrauch würde das Betriebssystem ohnehin Hauptspeicher auf einen Datenträger auslagern und es käme zu äquivalenten Verzögerungen.

6.1.3 Programmentwurf

Die weiter oben gestellten Anforderungen an das Programm zur parallelen Primzahlberechnung wurden in dem nun folgenden Programmentwurf umgesetzt.
Es existieren ein Hauptprozess P_0 und weitere p Prozesse P_1 bis P_p. Kommunikation mittels MPI-Funktionen findet nur zwischen dem Hauptprozess und den einzelnen anderen Prozessen statt. Als eine Art gemeinsamer Speicher existiert eine Datei, auf welche durch das Grid jeder Prozess zugreifen kann. Um Inkonsistenzen zu vermeiden, darf nur der Hauptprozess auf diese schreibend zugreifen – die anderen Prozesse dürfen diese nur lesen.

Der Programmablauf kann folgendermaßen beschrieben werden:

- Der Hauptprozess berechnet sequentiell mehrere Primzahlen a und speichert sie aufsteigend in die somit initialisierte Primzahlen-Datei. Dies ist notwendig, damit für die p Prozesse schon Primzahlen vorliegen, mit denen sie die ihnen zugeteilten Zahlen überprüfen können. Somit sind die Zahlen bis a schon nach Primzahlen durchsiebt worden.
- An die p Prozesse wird je eine Zahl der Zahlen a+1 bis a+p gesendet, so dass P_1 die

Zahl *a*+1 erhält, P_2 die Zahl *a*+2 und so weiter. Als mögliche Variante können – auf Kosten des Hauptspeicherverbrauchs – auch mehrere Zahlen gleichzeitig zum Überprüfen an einen Prozess gesendet werden.

- Der Hauptprozess wartet nun auf die Rückmeldung der einzelnen Prozesse, welche erklären, ob die ihnen gesendete Zahl (oder Zahlen) eine Primzahl ist oder nicht.
- Die *p* Prozesse lesen nacheinander die Primzahlen der zuvor angelegten Datei ein und überprüfen, ob die ihnen zugeteilte Zahl durch diese teilbar ist. Wenn sie alle Primzahlen bis zur Wurzel ihrer Zahl (oder Zahlen) überprüft haben und davon keine die vorgegebene Zahl teilt, so handelt es sich offensichtlich um eine Primzahl.
- Das Ergebnis ihrer Untersuchung wird von den *p* Prozessen an den Hauptprozess gesendet.
- Der Hauptprozess speichert die gefundenen Primzahlen in der Primzahl-Datei ab, indem er sie aufsteigend an die schon vorhandenen anhängt.
- Anschließend sendet der Hauptprozess die nächsten *p* (oder mehr) Zahlen an die Prozesse P_1 bis P_p.
- Damit beginnt der Berechnungskreislauf von vorne. Der Hauptprozess wird die Suche nach Primzahlen abbrechen, wenn eine vorgegebene Grenze erreicht ist oder die Größe der Datenstrukturen nicht mehr ausreicht. Dann übermittelt er den anderen Prozessen statt einer zu untersuchenden Zahl den Befehl sich zu beenden und beendet sich anschließend selbst.

6.1.4 Das Programm im Pseudocode (vereinfacht)

Wir können davon ausgehen, dass den Prozessen die eigene Prozessnummer, der Ort der gemeinsamen Primzahldatei `DATEI` und die Anzahl der Prozesse *p* bekannt ist. Tatsächlich brauchen die rechnenden Prozesse hier nur den Zugriff auf die Datei. Man kann also alle sonstigen Variablen als *lokal* ansehen.

Zunächst das Hauptprogramm für den Prozess P_0:

```
a:=1+floor(sqrt(p))                 //initialisiere die Laufvariable
write(DATEI, 2)                     //schreibe Primzahl 2
write(DATEI, 3)                     //schreibe Primzahl 3
for j=4 to a do                     //berechne bis a sequentiell
  begin
    primzahl:=true
    for k=2 to floor(sqrt(j)) do
      begin
        if j%k=0 then
          begin
            primzahl:=false
            break
          end
      end
    if primzahl then                //ist gestestete Zahl eine Primzahl?
      begin
        write(DATEI, j)             //schreibe gefundene Primzahl
      end
  end                               //sequentielle Berechnung fertig
for h=1 to floor((n-a)/p) do        //berechne ab a parallel
  begin
    for j=1 to p do                 //sende jedem Prozess eine neue Zahl
      send(P[j], ++a)
    for j=1 to p do                 //empfange von Prozessen das Ergebnis
      receive(P[j], ERGEBNIS[j])
    for j=1 to p do                 //überprüfe Ergebnisse
      if (ERGEBNIS[j]>0) then
        write(DATEI, ERGEBNIS[j])   //schreibe gefundene Primzahlen
  end
for j=1 to p do                     //sende "Abbruch" an jeden Prozess
  send(P[j], 0)
```

Der Pseudocode oben berechnet Primzahlen bis zu einem festen Wert *n*. Dieser Wert kann natürlich auf unendlich (bzw. auf ein durch die Datenstrukturen vorgegebenes Maximum) gesetzt werden da er nur zur Durchlaufsberechnung in der äußeren Schleife benutzt wird – man könnte also auch eine Endlosschleife konstruieren.

Die Initialberechnung der ersten Primzahlen muss mindestens bis zur Zahl $a = 1 + \sqrt{p}$ geschehen. Dies erklärt sich aus der Tatsache, dass alle Primzahlen bis zur Wurzel der Zahl, welche der letzte Prozess P_p erhält (nämlich $a + p$), bekannt sein müssen, um diese als mögliche Primzahl auszusieben.

Diese Schranke leitet sich folgendermaßen her:

Sei *a* die Zahl, bis zu der die Primzahlen berechnet werden müssen. Dann muss gelten:

$a + p \leq a^2$ bzw. die quadratische Ungleichung: $0 \leq a^2 - a - p$

Die Nullstellen der Funktion $a^2 - a - p$ sind:

$$\frac{1}{2} \pm \sqrt{\frac{1}{4} + p}$$

Um die Ungleichung zu erfüllen, reicht eine Abschätzung nach oben:

$$\frac{1}{2} \pm \sqrt{\frac{1}{4} + p} \;\leq\; \frac{1}{2} \pm \sqrt{\frac{1}{4}} + \sqrt{p} \;=\; 1 + \sqrt{p} \;\geq\; a$$

Damit ist die Zahl *a*, bis zu der die Primzahlen vorberechnet werden müssen, hergeleitet.

Der Pseudocode für die Prozesse P_1 bis P_p:

```
receive (P[0], TESTZAHL)           //empfange die zu testende Zahl
while TESTZAHL!=0 do               //starte Berechnung wenn kein Abbruch
  begin
    seek(DATEI, 0)                 //lese Primzahldatei ab Beginn
    read(DATEI, ZAHL)              //lese erste Primzahl ein
    while ZAHL<=sqrt(TESTZAHL) do  //solange gelesene Primzahl sinnvoll
      begin
        if TESTZAHL%ZAHL=0 then    //überprüfe auf Teilbarkeit
          begin
            TESTZAHL:=0            //setze negatives Ergebnis
            break                  //breche Berechnung ab
          end
        read(DATEI, ZAHL)          //lese nächste Primzahl ein
      end
    send(P[0], TESTZAHL)           //sende Ergebnis an Hauptprozess
    receive (P[0], TESTZAHL)       //empfange neue Zahl
  end
```

6.2 Implementierung und Anpassung des Programms

Der eben entworfene Algorithmus und seine Ausformulierung im Pseudocode geben den Rahmen vor, in dem die endgültige Implementierung in C sich bewegt. Es bleibt also die genaue Ausformulierung der benutzten Datenstrukturen zu klären und den Algorithmus an einige technische Details anzupassen.

6.2.1 Die gemeinsame Primzahldatei

Die Primzahldatei dient als gemeinsamer Speicher aller Prozesse. Diese stellt natürlich zusätzlich die Ausgabe der gesamten Berechnung dar, denn sie enthält alle berechneten Primzahlen und wird bei jedem Ergebnis praktisch sofort auf den neuesten Stand gebracht.
Um das Speichern und Einlesen zu vereinfachen sowie den Zeitaufwand gering zu halten werden die Primzahlen genauso gespeichert, wie sie im Speicher dargestellt werden, also als Sequenz von Long Integer (32 Bit). Wenn man zum Beispiel Primzahlen im Bereich von 1024 Bit berechnen will (zum Beispiel bei dem im Abschnitt 5.2.2.2 vorgestellten asymmetrischen Verschlüsselungsverfahren RSA), kann man trotz dieser unoptimierten Darstellung in gut 100 MB fast eine Millionen Primzahlen speichern. Dieses Verhältnis bedeutet selbst bei älteren Computern mit einer Festplattenkapazität in einem Bereich von Gigabytes keine praktischen Probleme.
Die Größenordnung des obigen Beispiels zeigt, dass es praktisch unmöglich ist, eine solche Menge im Hauptspeicher zu behalten (insbesondere wenn der Hauptspeicher nur 32 Megabytes beträgt). Außerdem wird der Kommunikationsaufwand sehr hoch, wenn man diese Daten dynamisch mittels klassischer MPI-Kommunikationsroutinen zwischen den Prozessen fortwährend verschiebt. Die Lösung einer gemeinsamen Datei scheint den Erfordernissen angemessen, also muss der gemeinsame Zugriff sichergestellt werden. In einem klassischen lokalen Netzwerk könnte die Datei sich auf einem Datenträger befinden, welcher zum Beispiel mittels des Network File System (NFS) allen teilnehmenden Rechnern zugänglich gemacht wird. In einem größeren Maßstab wie einem Grid mit sehr unterschiedlichen Teilnehmern und deren Systemen bieten sich die speziellen Routinen der Grid-Middleware an, insbesondere im Hinblick auf die gemeinsame Authentifikation für die Ressourcen, wie Prozessorzeit und Speicherplatz.
Das Globus Toolkit bietet verschiedene Möglichkeiten des gemeinsamen Datenzugriffs im Grid. Um auf Dateien zugreifen zu können bietet es sich an, die entsprechenden Funktionen des Global Access to Secondary Storage (GASS) zu benutzen. Praktisch kann man die entsprechenden C-Funktionen `fopen()` und `fclose()` durch ihre Entsprechungen `globus_gass_fopen()` bzw. `globus_gass_fclose()` mit den gleichen Parametern ersetzen. Der einzige Unterschied besteht darin, dass bei den GASS-

Varianten anstatt eines Pfades im Dateisystem auch eine URL angegeben werden kann, damit der Zugriff auch auf beliebige Dateien im Netzwerk gelingt. Da die Dateien recht groß werden, bietet es sich an, mittels einer LDAP-Abfrage des GIIS die Festplattenkapazitäten der im Grid partizipierenden Computer festzustellen. So ist es theoretisch auch möglich, bei signifikanten Veränderungen der Platzverhältnisse die Orginaldatei mit Hilfe der GridFTP-Funktionen zu einem besseren Platz zu transferieren. Letzteres wird hier allerdings nicht berücksichtigt.
Diese einfache und intuitive Methode der Anpassung von C-Programmen an eine Grid-Umgebung ist bei neueren Versionen des Globus Toolkits leider nicht mehr in den Standard-Bibliotheken vorgesehen.
Da es sehr schwierig ist, MPI-Programme während der Laufzeit an ein dynamisches System anzupassen, soll in das Programm noch die Funktion integriert werden, bei einem Neustart (insbesondere mit veränderter Anzahl der Client-Prozesse) mittels Übergabe einer noch existierenden Primzahldatei die angefangene Berechnung fortzusetzen.

6.2.2 Interne Darstellung der Primzahlen

Wie vorhin schon bei dem Dateiaufbau angedeutet, werden die Primzahlen (bzw. alle großen Zahlen in der Berechnung) als ein Feld mit vorzeichenlosen Long Integer Variablen dargestellt, wobei von den 32 Bit aus technischen Gründen nur 31 Bit für die tatsächliche Zahl benutzt werden, da das höchstwertige Bit als Übertrags-Bit bei Berechnungen reserviert wird. Diese Art der Darstellung ist zwar nicht sehr anpassungsfähig, da auch für kleinere Zahlen zum Beispiel 1024 Bit belegt werden, allerdings erleichtert es die Übergabe bei Funktionsaufrufen und der Zugriff ist schneller als etwa eine Lösung mittels verketteter Listen.
Natürlich gibt es bereits vefügbare Bibliotheken für eine Arithmetik mit großen Zahlen unter C wie GMP (GNU Multiple Precision Arithmetic Library) [70], aber sowohl die Notwendigkeit der eigenen Kontrolle über die maximale Größe der Zahlendarstellung als auch mögliche lizenzrechtliche Komplikationen haben mich dazu bewogen, eigene, entsprechend angepasste Funktionen zu schreiben.
Um mit diesen großen Zahlen wie mit normalen Integer rechnen zu können, müssen

natürlich alle verwendeten Rechenoperationen neu geschrieben werden. Aufgrund dessen nenne ich diese Zahlen auch *Binärzahlen*, denn letztlich werden sie in den Berechnungen als lange Binärketten behandelt und es gibt keine effektiver lesbare Ausgabe als ihre Binärdarstellung, da eine Umrechnug in die Dezimalschreibweise verhältnismäßig lange dauert (obwohl diese natürlich auch implementiert wurde). Im Gegenzug kann man dann von Zahlen, welche man mit Standard-C Funktionen in ihrer Dezimalschreibweise ausgeben kann, als von *Dezimalzahlen* reden.

Hier ein Auszug der wichtigsten Funktionen, welche speziell für diese Primzahlberechnung entwickelt wurden:

```
void load_bin_with_dec (unsigned long bnumber[],
                        unsigned long decimal)
```

Damit wird ein normaler Long Integer in eine Binärzahl umgewandelt.

```
void load_binnumber (unsigned long bnumber1[],
                     unsigned long bnumber2[])
```

Diese Funktion entspricht der Zuweisung `bnumber1=bnumber2`.

```
void show_binnumber_bin (unsigned long bnumber[])
```

Die Binärzahl `bnumber` wird in ihrer Binärschreibweise ausgegeben.

```
int add_binnumber (unsigned long bnumber1[],
                   unsigned long bnumber2[],
                   unsigned long solution[])
```

Diese Funktion ist gleich der Berechnung `solution=bnumber1+bnumber2`.
Der Rückgabewert ist normalerweise 0, es sei denn, bei der Berechnung wurde die maximal zulässige Größe von `solution` überschritten, dann beinhaltet er dem Übertrag.

```
int sub_binnumber (unsigned long bnumber1[],
                   unsigned long bnumber2[],
                   unsigned long solution[])
```

Diese Funktion ist gleich der Berechnung `solution=bnumber1-bnumber2`.
Der Rückgabewert ist normalerweise 0, es sei denn, die Berechnung ergibt einen Wert unter 0 für `solution`, dann enthält sie den letzten Übertrag der Berechnung.

Diese Funktion lässt sich somit auch als Test für eine "<"-Beziehung gebrauchen, denn wenn `bnumber1<bnumber2`, so ist der Rückgabewert verschieden von 0.

```
int mul_binnumber (unsigned long bnumber1[],
                   unsigned long bnumber2[],
                   unsigned long solution[])
```

Diese Funktion bezieht sich auf die Berechnung `solution=bnumber1*bnumber2`. Der Rückgabewert ist normalerweise 0, es sei denn bei der Berechnung wurde die maximal zulässige Größe von `solution` überschritten, dann entspricht er dem letzten Übertrag der sukkzessiv aufgerufenen Funktion `add_binnumber()`.

```
int mul_binnumber (unsigned long bnumber1[],
                   unsigned long bnumber2[])
```

Diese Funktion entspricht der Berechnung `bnumber1%bnumber2`, wobei nicht das tatsächliche Ergebnis interessiert, sondern ob `bnumber1` durch `bnumber2` teilbar ist. In diesem Fall ist der Rückgabewert 0, sonst entspricht er dem letzten Übertrag der sukkzessiv aufgerufenen Funktion `sub_binnumber()`.

```
void div_binnumber (unsigned long bnumber[],
                    unsigned long solution[],
                    int shifts)
```

Diese Funktion entspricht der Berechnung `solution=bnumber/2^shifts`, ist also eine Funktion, um Binärzahlen durch ein Vielfaches von 2 zu teilen. Man beachte, dass natürlich nur der ganzzahlige Teil des Ergebnisses in solution gespeichert wird.

```
int equal_binnumber (unsigned long bnumber1[],
                     unsigned long bnumber2[])
```

Diese Funktion entspricht dem Ausdruck `bnumber1==bnumber2`. Allerdings ist der Rückgabewert 0, wenn die beiden Zahlen gleich sind, und 1 wenn sie verschieden sind.

```
void sqrt_binnumber (unsigned long bnumber[],
                     unsigned long solution[])
```

Wie im vorigen Kapitel ausgeführt hat bei der Primzahlberechnung die Wurzelfunktion eine große Bedeutung und so entspricht diese Funktion auch der (ganzzahligen) Berechnung `solution=sqrt(bnumber)`.

```
void bin_to_dec (unsigned long bnumber[],
                 unsigned dnumber[])
```

Diese Funktion rechnet eine Binärzahl bnumber in eine Dezimalzahl dnumber um, wobei die Dezimalzahl ein Feld ist, dessen Einträge den Dezimalstellen entsprechen, also: (`dnumber[]=[1,2,3,4]` wäre die Zahl `4321`).

```
void show_decnumber (unsigned dnumber[])
```

Die beispielsweise mittels `bin_to_dec()` errechnete Dezimalzahl `dnumber` wird ausgegeben. Dabei übernimt die Funktion bewusst die direkte Ausgabe und liefert keinen Rückgabeparameter vom Typ String zurück, da die Länge der Zahl in Dezimalschreibweise mögliche Feldgrößen sprengen könnte.

Da prinzipiell nur ganze positive Zahlen als Ausgangsparameter vorkommen, sind dafür die obigen Funktionen ausreichend. Die Berechnungen im Programm sind so konstruiert, dass der maximale Rundungsfehler 1 beträgt. Die Funktionen für die Berechnungen sind sehr kanonisch umgesetzt und daher kaum optimierbar – lediglich eine andere interne Zahlendarstellung würde weitere sinnvolle Ansätze zulassen.

6.2.3 Synchronisation der Kommunikation

Eines der großen Probleme der Synchronisation der Prozesse ist die Regelung des Zugriffs auf die gemeinsame Primzahldatei. Einerseits müssen die Prozesse gleichzeitig daraus lesen (sonst wäre die Parallelisierung nicht sehr effektiv) und andererseits müssen auch neue Primzahlen hineingeschrieben werden – ohne natürlich zu Inkonsistenzen der Datei zu führen.

Diesem Problem wird im vorgelegten Entwurf mit zwei Maßnahmen begegnet:

- Den Schreibzugriff darf nur der Hauptprozess durchführen.
- Während der Hauptprozess auf die Datei zugreift, dürfen die anderen Prozesse nicht lesen, um fehlerhafte Daten zu vermeiden, denn es könnte gerade an derselben Stelle gelesen und geschrieben werden.

Dies bedeutet allerdings eine gewisse Einschränkung in der Parallelität. Der Hauptprozess übergibt den Prozessen zwar die zu untersuchenden Zahlen praktisch gleichzeitig, allerdings speichert er die zurückgemeldeten Ergebnisse in einem Cache und wartet, bis alle rechnenden Prozesse fertig sind, bevor er ihnen neue Zahlen übermittelt. Der Hauptprozess kann also ungestört den Primzahl-Cache an die Primzahldatei anfügen, ohne von den anderen Prozessen unterbrochen zu werden.
Da die getesteten Zahlen direkt aufeinander folgen, kann man zwar theoretisch die gleiche Bearbeitungszeit im Mittel erwarten, allerdings sind in der Praxis zum Beispiel Prozesse, welche eine durch drei teilbare Zahl überprüfen, sehr viel früher fertig als andere. Hier wird die Parallelität nicht sehr gut ausgenutzt, allerdings hat dieses Vorgehen noch einen anderen Vorteil. Bei der Kommunikation über MPI-Funktionen ist nicht gewährleistet, dass die Nachrichten in der gleichen Reihenfolge empfangen werden, in der sie losgeschickt wurden. Bei nicht gefiltertem Empfang kommen im schlechtesten Fall Nachrichten von einem Prozess gar nicht erst zum Hauptprozess durch, da sich die Nachrichten von anderen Prozessen immer wieder "vordrängeln". Bei entsprechenden Testläufen konnte ein solches Verhalten häufig beobachtet werden. Da aber im nun vorliegenden Algorithmus der Hauptprozess zwischen den Berechnungsschritten auf je eine Antwort von jedem Prozess wartet, werden solche systembedingten Probleme umgangen und eine Synchronisation der Kommunikation quasi erzwungen.
Um die Kommunikation genau zu synchronisieren wird die MPI-Funktion `MPI_Recv()` verwendet, welche zu den blockierenden Kommunikationsaufrufen gehört, die also wartet, bis ihre Nachricht gesendet wurde. Dies bedeutet keinerlei Einschränkung der Parallelität der berechnenden Prozesse, denn zwischen dem Senden ihrer Ergebnisse und dem Zuteilen einer neuen Zahl (bzw. neuer Zahlen) können diese ohnehin nichts berechnen.

6.2.4 Ausgabe der Primzahlen

Wie schon oben erläutert ist die eigentliche Ausgabe die Primzahldatei. Während die Berechnung noch voranschreitet, kann man dort die bisherigen Ergebnisse einsehen. Zwar sind diese dort fast in ihrer Binärschreibweise gespeichert – man muss nur jedes

zweiunddreißigste Bit ignorieren – aber eine Visualisierung als Dezimalzahl ist für Menschen natürlich vertrauter. Obwohl dies schon eine recht nützliche Funktion ist, kommt ihr doch im Rahmen des Projektes nur eine sehr untergeordnete Rolle für Testzwecke zu und soll hier nur der Vollständigkeit halber erwähnt werden.
Eine große Dezimalzahl wird als Feld mit einer Belegung von Kombinationen der Ziffern 0 bis 9 dargestellt, welche problemlos nacheinander als Character ausgegeben werden können. Um nun eine Binärzahl umzuwandeln, wird die vorinitialisierte Zahl 0 sooft kontinuierlich erhöht, wie die Binärzahl erniedrigt werden kann.
Diese Vorgehensweise ist zwar sehr einfach, doch kostet sie relativ viel Zeit, weswegen diese Darstellung bei den Berechnungen auch keine Rolle spielt. Wenn man die Berechnungen verstärkt interaktiv gestalten möchte – zum Beispiel mit lesbarer Ausgabe aller Ergebnisse – würde es sich empfehlen, im Hauptprozess mit der zu testenden Binärzahl auch die entsprechende Dezimalzahl immer zu erhöhen. Damit wäre keine Umrechnung mehr nötig, denn die Dezimalzahl würde durch die gleichzeitige Erhöhung automatisch dem richtigen Wert der Binärzahl entsprechen.

6.2.5 Mögliche Optimierungen

Sicherlich gibt es mehrere Optimierungsansätze bei den beschriebenen Implementierungen. Allerdings sollte man nicht vergessen, dass die obigen Beschreibungen auf ein – wie anfangs erwähnt – sehr eingeschränktes Hardwarekonzept zugeschnitten sind.
Im Vordergrund stand bei den Datenstrukturen ein absolut minimaler Hauptspeicherverbrauch und bei der Umsetzung des Algorithmus eine vertretbare Übersichtlichkeit, was Performanceverluste nur im konstant abschätzbaren Bereich bedeutet.

Wenn man das gleiche Konzept zur Berechnung verfolgt, so wird man nur im Detail Änderungen durchführen können. Möglich wären unter anderem:

- Wenn auf einen regelnden Hauptprozess verzichtet wird, so bezahlt man dies durch eine komplexe Kommunikationsstruktur unter den Prozessen, welche nötig ist um zu

entscheiden, wer wann welche Zahl überprüft und wer wann in den gemeinsamen Speicher schreiben kann.

- Statt eines gemeinsamen Speichers könnte jeder Prozess sein Ergebnis mittels eines *Broadcasts* allen anderen Prozessen mitteilen. Dies würde einen Teil des Synchronisationsproblems lösen, aber die Speicherkosten insgesamt – je nach Anzahl der Prozesse – drastisch erhöhen.
- Die am wenigsten problematische und langfristig sinnvollste Optimierung (Dateien haben eine vom Dateisystem festgeschriebene Maximalgröße) wäre ein anderer Aufbau der Primzahldatei. Wie im vorigen Kapitel angesprochen gibt es dazu mehrere Ansätze, wie zum Beispiel die Darstellung der ungeraden Zahlen mit 1 bzw. 0 – je nachdem, ob diese eine Primzahl ist oder nicht.
- Die interne Zahlendarstellung kann von Feldern zu (beispielsweise) verketteten Listen umgestellt werden. Die relativ geringe Platzersparnis würde allerdings durch komplexere (und damit effektiv langsamere) Berechnungsroutinen erkauft werden.
- Der vorliegende Algorithmus wurde unter dem Spezialfall untersucht, dass jeder Prozess jeweils nur eine Zahl zum Untersuchen vom Hauptprozess geschickt bekommt. Dies geschah unter der Bedingung der minimalen Hauptspeicherbelastung. Natürlich ist im realen Programm durch Übergabe eines entsprechenden Parameters (`-s`) auch die Möglichkeit vorgesehen diese zu skalieren. Allerdings variiert so die Laufzeit des Programms nur um einen konstanten Faktor und ist daher in der theoretischen Analyse des Programms nicht von Bedeutung.

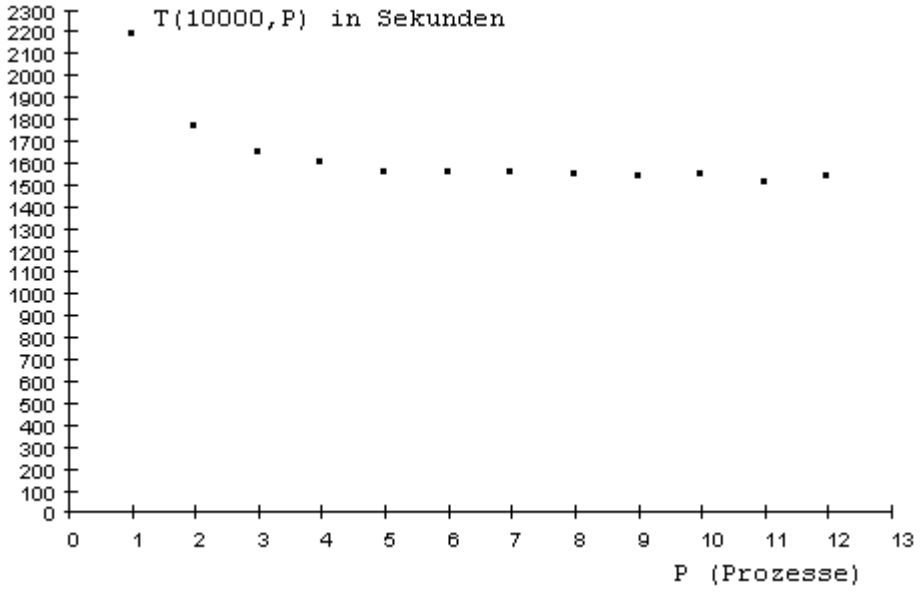
Anwenderprogramme, welche große Primzahlen benötigen, bringen in der Regel weder große Primzahlbibliotheken mit noch rechnen sie tagelang Primzahlen aus. Tatsächlich erstellen sie zufällige, entsprechend große Zahlen und testen diese mit bestimmten Verfahren darauf, ob es mögliche Primzahlen sind (zum Beispiel mit dem Miller-Rabin-Test im Abschnitt 5.1.9). Allerdings können diese Verfahren nicht mit absoluter Sicherheit garantieren, dass es sich auch um eine Primzahl handelt.
Dies zeigt, dass es mit geringem Ressourcenaufwand nicht ohne weiteres möglich ist, große Primzahlen zu berechnen. Jegliche Optimierung kann also die Berechnung nur für bestimmte Ressourcenverhältnisse optimieren, aber die Berechnung selbst kann in ihrer Größenordnung nicht signifikant beschleunigt werden.

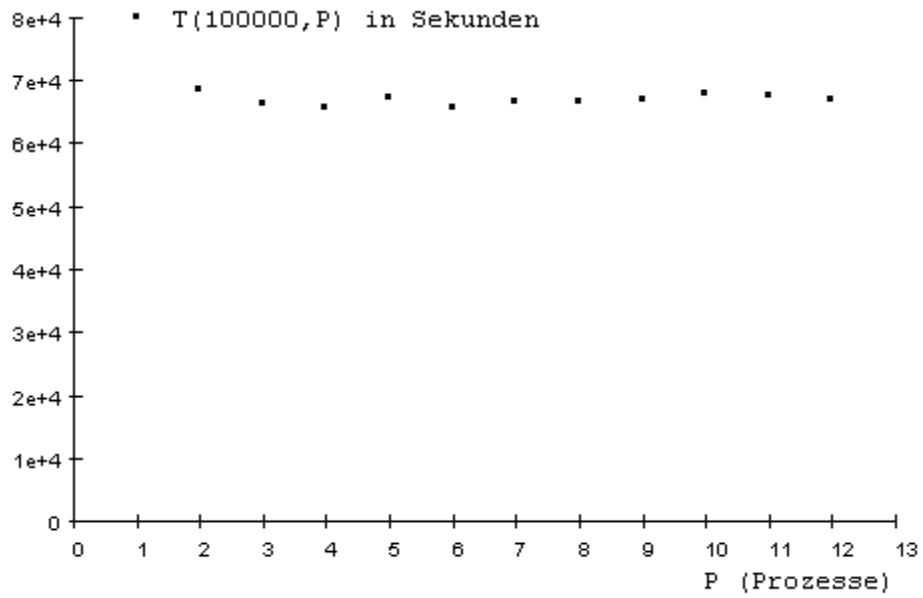
6.3 Auswertung der Programmergebnisse

Der genaue Sourcecode des entwickelten Programms `gridprime.c` kann im Anhang eingesehen werden.
Dieser Abschnitt befasst sich mit der Auswertung der Berechnungen für feste Werte von n. Es ist sehr interessant auszuloten, wie der Algorithmus auf verschiedene Parameter wie die Anzahl der Prozesse reagiert, und auch den tatsächlichen Geschwindigkeitsgewinn durch die Parallelisierung zu erfahren.
Zu diesem Zwecke wird die MPI-Funktion `MPI_Time()` herangezogen. Diese wurde schon im 4. Kapitel vorgestellt und für die folgende Auswertung ist lediglich das gemessene Ergebnis in Sekunden von Bedeutung.

6.3.1 Mehrere Prozesse auf einem Rechner

Die MPI-Umgebung gestattet es, die Prozesse auf beliebig viele Prozessoren zu verteilen – welche letztlich natürlich über das Grid erreicht werden. Da für die Testläufe nur zwei Prozessoren zur Verfügung standen, bot es sich an, zu untersuchen, ob mehrere Prozesse auf einem System einen Synergieeffekt hervorbringen.
Dazu geben folgende Diagramme ein typisches Bild ab:



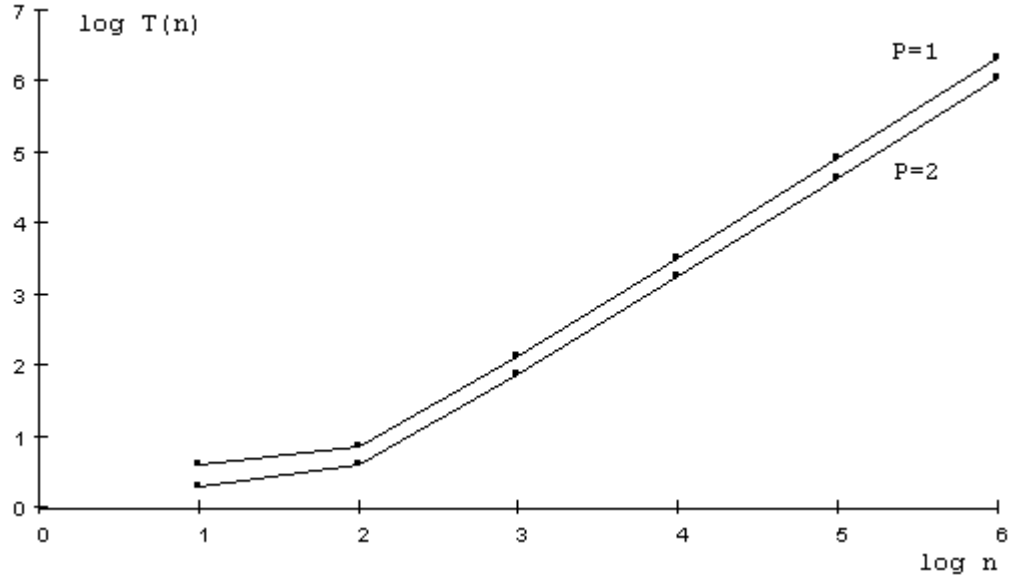


Unabhängig von der Siebgröße n erreicht man bei etwa drei rechnenden Prozessen noch einen nennenswerten Geschwindigkeitsvorteil. Dass dies unabhängig von der Variable n ist, verwundert insofern nicht, da das Programm – wie oben beschrieben – immer den gleichen Hauptspeicherbedarf hat und die Primzahldatei über das Grid in einem lokalen Cache gehalten wird – auch hier kann also keine nennenswerte Verzögerung auftreten.
Bei diesem Testsystem ist ein Vorteil bei bis zu drei Prozessen spürbar, das liegt offenbar in der Performance des Gesamtsystems, denn eine andere Betriebssystemstruktur bzw. Prozessverwaltung könnte einem einzigen Prozess die nahezu gesamte Prozessorzeit zur Verfügung stellen, so dass mehrere Prozesse nicht mehr sinnvoll wären. Das als Server eingerichtete Linux und die Leistungssicherung des Globus Toolkits balancieren die Prozesse offenbar aus.
Die Zeitersparnis liegt bei etwa 10 bis 20 Prozent. Diese Werte sind vermutlich auch dem Scheduling zuzuschreiben, welches versucht die Prozessorzeit auf die Prozesse gleichmäßig zu verteilen. In diesem Fall wird dem parallelen Programm insgesamt mehr Prozessorzeit zugeteilt – ermöglicht durch mehrere ihm zugehörige Prozesse – schließlich kommt es bei dem Programm praktisch nur auf die Gesamtzeit an, in der es Berechnungen durchführen kann, und weniger darauf, wie viel Prozessorzeit die einzelnen Prozesse genau erhalten (sofern diesen in etwa die gleiche Prozessorzeit zugeteilt wird).

6.3.2 Laufzeit für zwei Prozessoren bei ansteigender Siebgröße

Im Folgenden soll die reale Laufzeit der Testdurchläufe dokumentiert und untersucht werden. Später im Kapitel werden die hier gewonnenen Erkenntnisse mit der theoretischen Auswertung des verwendeten Algorithmus verglichen.

Zunächst einmal werden die gemessenen Daten graphisch und numerisch dargestellt. Dabei werden nur zur besseren Skalierung der Grafik die Logarithmen der Werte betrachtet:



Komplexität n (Siebgröße)	T(n,P=1) in Sekunden	T(n,P=2) in Sekunden
10	4	2
100	7	4
1000	131	73
10000	3187	1738
100000	79977	41119
1000000	2107634	1094554

Aus der Wertetabelle ist abzulesen, dass der Einsatz von zwei Prozessoren die Laufzeit im Gegensatz zum sequentiellen Abarbeiten fast halbiert. Dies verwundert nicht umbedingt, denn die zur Verfügung stehende Rechenleistung hat sich schließlich fast verdoppelt (je

ein 120 MHz und ein 133 MHz Prozessor). Allerdings erzeugen der Kommunikationsaufwand, der Zugriff auf die Primzahldatei und sequentielle Teilabschnitte des Programms einen Overhead, der sich negativ auf die Performance auswirkt. Dazu muss noch bedacht werden, dass es sich nicht um dedizierte Prozessoren handelt, sondern um den jeweiligen Hauptprozessor eines Servers, welcher in verschiedenen Zeitintervallen vermehrt Prozessorzeit für andere Prozesse verbraucht. Am Diagramm ist abzulesen, dass die niedrigen Werte von n (10 und 100) sich nicht ganz in das Gesamtbild einfügen. Dies ist ein zusätzliches Indiz dafür, dass der Overhead im Vergleich zur tatsächlichen Rechnung sehr groß ist. Ebenso würde das Diagramm einen linearen Zusammenhang der logarithmierten Größen n und T(n) vermuten lassen, was allerdings nicht ganz stimmt, wie die anschließende theoretische Auswertung des Algorithmus aufzeigen wird.

6.4 Definition und Berechnungsgrundlage von Kostenmaßen

Um Algorithmen formal zu beschreiben und auch die Möglichkeit zu haben Algorithmen quantitativ miteinander zu vergleichen braucht man ein – in der Regel abstraktes – Computermodell. Dieses sollte einerseits möglichst einfach gehalten sein, um theoretische Untersuchungen zu unterstützen, und andererseits die Rechenmöglichkeiten eines wirklichen Computers hinreichend gut abbilden. Letzeres ist vor allem notwendig, damit die gewonnenen Erkenntnisse praktische Relevanz besitzen.

6.4.1 Das PRAM Modell

Je nach Aufbau eines speziellen Rechners kann man sich natürlich ein Modell überlegen, welches die Besonderheiten dieses Rechners implizit abbildet. Im Folgenden soll aber ein sehr grundlegendes Modell verwendet werden – das sogenannte PRAM-Modell (Parallel Random Access Machine). Dabei wird eine endliche Anzahl von Prozessoren angenommen, welche auf einen gemeinsamen Speicher zugreifen können. Dieses Modell passt sehr gut, da die vorhandene Hardewarekonfiguration eher einem Mini-Cluster als

einem klassischen heterogenen Grid entspricht.
Eine PRAM besitzt gegenüber ihrem sequentiellen Gegenstück (RAM) noch einige modellbedingte Besonderheiten, welche einer Klarstellung bedürfen:

- Es ist es unerheblich, ob die Prozessoren noch einen lokalen Speicher haben – man kann dann genausogut davon ausgehen, dass jeder Prozessor einen "privaten" Speicherbereich im gemeinsamen Speicher hat.
- Das PRAM-Modell ist ein synchrones Modell für parallele Berechnungen. Das heißt, dass die Prozessoren gleichzeitig mit der Bearbeitung eines Befehls beginnen und auch gleichzeitig fertig sind.
- Eine PRAM gehört entweder zur Klasse der SIMD-Modelle (Single Instruction Multiple Data) oder zu den MIMD-Modellen (Multiple Instruction Multiple Data). Ersteres bedeutet, dass in einem Takt alle Prozessoren den gleichen Befehl ausführen, der allerdings auf verschiedene Daten angewendet werden darf. Analog dazu kann ein MIMD-Modell für jeden Prozessor ein separates Programm ausführen.
  Andere derartige Modelle gibt es eigentlich nicht, denn ein SISD-Modell wäre gleichbedeutend mit sequentieller Abarbeitung und für ein MISD-Modell gibt es in der Realität keine Entsprechung.

Da mehrere – oder alle – Prozessoren theoretisch auf die gleiche Speicherzelle simultan zugreifen können, benötigt eine PRAM noch eine genaue Richtlinie hinsichtlich der Speicherzugriffe:

- Eine EREW-PRAM (Exclusive Read Exclusive Write) erlaubt keinerlei gleichzeitige Speicherzugriffe. Es darf also immer nur genau ein Prozessor eine Speicherzelle lesen bzw. schreiben.
- Eine CREW-PRAM (Concurrent Read Exclusive Write) gewährt den gemeinsamen Lesezugriff, aber gleichzeitig in ein und dieselbe Speicherzelle zu schreiben ist noch immer nur einem Prozess erlaubt.
- Eine CRCW-PRAM (Concurrent Read Concurrent Write) gestattet zwar das gleichzeitige Lesen bzw. das Schreiben verschiedener Prozesse in die gleiche Speicherzelle. Jedoch muss noch vorgegeben werden, welchen Wert die Speicherzelle

erhalten soll, falls tatsächlich mehrere Prozessoren gleichzeitig schreiben. Zahlreiche Möglichkeiten beinhalten die "common CRCW-PRAM", welche nur dann einen gemeinsamen Schreibzugriff erlauben, falls alle Prozessoren den gleichen Wert schreiben wollen; des Weiteren gibt es die "arbitrary CRCW-PRAM", welche einem beliebigen Prozessor den Vorzug gibt; zuletzt soll noch die "priority CRCW-PRAM" genannt werden, welche mittels einer bestimmten Vorschrift entscheidet, welcher Prozessor seinen Wert schreiben darf – zum Beispiel der Prozessor mit der geringsten Prozessornummer.

6.4.2 Darstellung der Algorithmen

Die Algorithmen werden häufig in elementarer Pascal-Notation formuliert, da diese einerseits klar strukturiert ist und andererseits einen hohen Bekanntheitsgrad besitzt. Allerdings gibt es in Pascal natürlich keine explizite Struktur, um Anweisungen auch parallel ausführen zu können.
Dazu dient folgender neuer Befehl (füre eine SIMD-PRAM):
`for <Bedingung> pardo <Anweisungen>`

Dies bedeutet schlicht, dass alle Prozessoren, für welche die `<Bedingung>` erfüllt ist, die `<Anweisungen>` parallel ausführen. Normalerweise gibt man für die `<Bedingung>` einen Ausdruck der Form $0 \leq i \leq n$ an, was heißt, dass die `<Anweisungen>` in den Prozessoren 1 bis n parallel abgearbeitet werden, denn es gibt für jeden Prozessor eine lokale Variable – hier i genannt – welche die eigene Prozessornummer enthält.
Diese Konvention zeigt auch, dass für konkrete Algorithmen die Frage gestellt werden muss, ob die Anzahl der tatsächlich vorhandenen Prozessoren in den Entwurf einfließen soll oder generell beliebig, aber endlich viele Prozessoren vorausgesetzt werden können. Letzteres führt normalerweise zu einem übersichtlicheren Algorithmus, allerdings kann bei bestimmten Algorithmen gerade die real beschränkte Prozessoranzahl das eigentliche Problem ausmachen.

6.4.3 Komplexitätsmaße im PRAM-Modell

Wenn man die Effizienz eines Algorithmus ermitteln will, so interessiert man sich in der Regel für die – im abstrakten Modell ermittelte – *Laufzeit* t(n) bei einem vorgegebenen Parameter n – auch *Komplexität* genannt. Sinnvoll kann es sein, erst den *Aufwand* w(n) zu ermitteln, welcher der Gesamtzeit aller durchgeführten Operationen entspricht. Bei einem sequentiellen Algorithmus gilt daher offensichtlich t(n)=w(n).
Für parallele Algorithmen ist die für sie benötigte *Prozessoranzahl* p(n) häufig von Bedeutung, welche bei beliebig skalierbaren Algorithmen natürlich 1 für jede Komplexität n beträgt (der Algorithmus ist auch sequenziell ausführbar).
Offensichtlich gilt die folgende Beziehung:

$$w(n) \leq p(n) \cdot t(n)$$

Ein weiteres interessantes Komplexitätsmaß ist auch der gesamte bzw. zur Laufzeit benötigte Speicher. Auf dieses sehr spezielle Maß wollen wir hier aber nicht weiter eingehen, da der Algorithmus nach seiner Konstruktion nur einen konstanten Speicherverbrauch hat.

Mit einem realen Computer kann man die Laufzeit für den Algorithmus für jedes n konkret in Sekunden angeben. Jedoch können sich durch unberücksichtigte Störfaktoren – wie ein anders verlaufendes Prozess-Scheduling – selbst bei gleichen Eingabeparametern die Ergebnisse verändern. Auch würde ein Algorithmenvergleich immer die exakt gleichen Computerressourcen voraussetzen, da sich bei veränderter Hardware keine gültige Aussage mehr ableiten ließe.
Daher interessiert man sich bei den Komplexitätsmaßen nicht für konkrete Zeitangaben, sondern für deren asymptotisches Verhalten in Abhängigkeit von n.

6.4.4 Definition der O-Notation

Seien f,g: $\mathbb{N} \to \mathbb{R}^+$ Funktionen. Dann steht die Schreibweise $h = O(g)$ dafür, dass es eine Konstante $k > 0$ gibt, so dass für alle (bis auf endlich viele) n aus $\mathbb{N}$ gilt:

$$h(n) \leq k \cdot g(n)$$

$O(g)$ steht also stellvertretend für eine Menge von entsprechenden Funktionen – das Gleichheitszeichen ist nur eine gebräuchliche Schreibkonvention und nicht mathematisch zu interpretieren.

Beispiel:

$$42 \cdot n \cdot n^{42} + 42 = O(n^{43})$$

Natürlich gilt auch:

$$42 \cdot n \cdot n^{42} + 42 = O(n^{101})$$

bzw:

$$O(n^{43}) = O(n^{101}) = O(n^{1001})$$

Damit sollte man sich in das Gedächtnis zurückrufen, dass nach oben natürlich beliebig großzügig abgeschätzt werden kann. Wenn man die Komplexitätsmaße von Algorithmen untersucht, muss also darauf geachtet werden, dass nicht zu grob abgeschätzt wird, um eine sinnvolle Aussage über den Algorithmus treffen zu können.

6.4.5 Das uniforme Kostenmaß

Das vermutlich einfachste bzw. überschaubarste Kostenmaß ist das sogenannte *uniforme Kostenmaß*. Dabei steht der Parameter *n* für die Anzahl der Zahlen, welche die Eingabe der PRAM bilden. Die Laufzeit t(*n*) bezeichnet die Anzahl der nötigen Bearbeitungsschritte, um den Algorithmus mit der entsprechenden Eingabe zu bearbeiten. Zu beachten ist dabei natürlich, dass eine PRAM in einem Bearbeitungsschritt einen Befehl auf mehreren Prozessoren gleichzeitig ausführen kann.
Diese Indifferenzierung der Rechenschritte und der benutzten Zahlen sollte natürlich bei jedem Algorithmus hinterfragt werden, um keine unrealistischen Annahmen zu machen:

- Wenn die Zahlen der Eingabe und die berechneten Zwischenergebnisse eine gemeinsame größenordnungsmäßige Beschränkung aufweisen (was häufig der Fall ist), so ist die Annahme diesbezüglich gerechtfertigt.
- Die Annahme, dass alle Rechenschritte der gleichen Größenordnung entsprechen, hängt im Allgemeinen davon ab, welche Operationen in einem Rechenschritt der

PRAM zugelassen sind. Offensichtlich sind arithmetrische Operationen – wie Addition und Subtraktion – als äquivalent anzusehen, allerdings sind je nach Hardwareimplementierung Multiplikationen oder noch komplexere Funktionen – wie Logarithmen – nur schwerlich mit diesen gleichzusetzen. Auch Speicherzugriffe lassen sich noch untereinander vernünftig abschätzen (Probleme treten unter anderem bei Vergleichen mit komplexen Speichermodellen auf), allerdings sind diese mit den arithmetischen Operationen noch schwieriger zu vergleichen. Letztlich kommt es also auch auf das konkrete Modell der PRAM an, ob das uniforme Kostenmaß angebracht ist. Insbesondere ist es natürlich nicht unerheblich, welche Operationen im Algorithmus verwendet werden. Werden nur sehr elementare Operationen durchgeführt, so ist dies natürlich günstiger für das uniforme Kostenmaß – ebenso wenn hauptsächlich arithmetische Operationen oder Speicherzugriffe stattfinden.

6.5 Untersuchung des entworfenen Algorithmus im PRAM-Modell

Da der im vorigen Abschnitt vorgestellte Algorithmus zur Primzahlberechnung im Folgenden weiter untersucht werden soll, brauchen wir eine Konvertierung desselben in das PRAM-Modell. Genau genommen wird von einer CREW SIMD-PRAM ausgegangen.
Dies hat den Vorteil, dass die parallel ausgeführten Programmabschnitte sich leicht abschätzen lassen.

6.5.1 Konvertierung des Primzahlprogramms in das PRAM-Modell

Auf den ersten Blick scheint der Algorithmus nicht in ein SIMD-Modell zu passen – wurde doch zwischen dem Hauptprozess und den Rechenprozessen unterschieden. Tatsächlich wurde aber durch die erzwungene Synchronisation bewirkt, dass entweder die Rechenprozesse oder der Hauptprozess tatsächlich arbeiten. Also kann die Aufgaben des Hauptprozesses auch ein Rechenprozess übernehmen (in der Praxis lässt man den Hauptprozess mit einem Rechenprozess auf einem Prozessor laufen, da sie sich

gegenseitig nicht beeinträchtigen). Die gemeinsame Datei entspricht im PRAM-Modell natürlich dem gemeinsamen Speicher. Ebenso gibt es keine explizite Kommunikation mehr, denn die Prozesse können ihre Ergebnisse über den gemeinsamen Speicher austauschen.
Lediglich in der tatsächlichen Berechnung müssen an das Modell größere Zugeständnisse gemacht werden, denn bei dem SIMD-Modell können die parallel arbeitenden Prozessoren keine unterschiedlichen Befehle ausführen, also auch nicht die Berechnung früher abbrechen als ein anderer Prozessor. Dies entspricht zwar im Prinzip der geforderten Synchronisation im Ausgangsprogramm, hat aber eine noch größere Tragweite: Wenn kein Prozessor die Berechnung vorzeitig abbrechen darf, so rechnen alle Prozessoren so lange, als hätten sie eine Primzahl überprüft. Das würde in diesem Fall keinen nennenswerten Unterschied machen, wenn mindestens ein Prozessor tatsächlich eine Primzahl findet, allerdings ist dies bei praktisch geringer Prozessoranzahl sehr selten gegeben. Dieser Umstand mag die nachfolgenden Berechnungen der Komplexitätsmaße vereinfachen, allerdings beeinträchtigt dies die Genauigkeit – insbesondere bei großen Zahlen n und kleiner Prozessoranzahl p.
Fast schon überraschend gelingt es dennoch, das unter großen Freiheiten entworfene Programm in das doch sehr beschränkte SIMD-PRAM-Modell zu übertragen, ohne den eigentlichen Algorithmus zu verändern, was natürlich die wichtigste Voraussetzung für eine aussagekräftige Untersuchung in diesem Modell ist.

6.5.2 Anwendung des uniformen Kostenmaßes

Für die nachfolgenden Berechnungen der Komplexitätsmaße soll das uniforme Kostenmaß verwendet werden.
Im Bezug auf die verwendeten Zahlen und Zwischenergebnisse ist es in diesem Falle klar, dass sie derselben Größenordnung angehören, denn es wird für die Umsetzung des Algorithmus mit den speziellen Binärzahlen gerechnet, welche schließlich durch ihre Konstruktion als Feld beschränkt sind.
Auch was die Durchführung der arithmetischen Operationen angeht, werden hauptsächlich die weiter oben beschriebenen Funktionen für Binärzahlen verwendet.

Davon soll nur die Wurzelfunktion zur Kostenberechnung näher betrachtet werden, denn allein für deren Umsetzung gibt es vielerlei Algorithmen. Die restlichen in 6.6.2 vorgestellten Funktionen haben nach Initialisierung der Binärzahlen eine maximale Laufzeit und können so mit einer Konstante abgeschätzt werden.
Die übrigen Operationen sowie Speicherzugriffe werden vornehmlich mit kleineren Indexvariablen durchgeführt, allerdings auch um auf Binärzahlen zuzugreifen.
Man kann also dem Algorithmus durchaus zugestehen, dass das uniforme Kostenmaß angebracht ist.

6.5.2.1 Die Wurzelfunktion im PRAM-Modell

Im Hauptprogramm wird an verschiedenen Stellen die Wurzel von natürlichen positiven Zahlen berechnet. Da dies eine vergleichsweise komplexe Operation ist und es praktisch keinen Prozessor gibt, der diese als eigenständigen Befehl zur Verfügung stellt, ist es nicht sinnvoll, ihn als selbständige Operation einer PRAM aufzufassen.
Da das Modell eines gemeinsamen Speichers vorausgesetzt wird, handelt es sich bei allen Variablen um globale Variablen (mit Ausnahme des lokalen Prozessorindex i, welcher hier aber nicht verwendet wird).
Im folgenden Pseudocode ist zu jeder Zeile der Aufwand der Übersichtlichkeit halber bereits angegeben. Eine genauere Besprechung der Berechnung wird anschließend vorgenommen.

Pseudocode der Wurzelfunktion:

```
//Berechne sqrt(ZAHL)
//Das Ergebnis ist abschließend in der Variable GUESS enthalten
//Anweisungen                       Kosten
OLDGUESS:=0                         //1
GUESS:=ZAHL/2                       //2
MAX:=ZAHL                           //1
MIN:=0                              //1
while (GUESS!=OLDGUESS) do                  //1*ld(ZAHL)
  //Iterationsstart
  begin
    DELTA:=GUESS*GUESS              //2*ld(ZAHL)
    if (DELTA>ZAHL) then                    //1*ld(ZAHL)
      begin
        MAX:=GUESS                          //1*ld(ZAHL)
```

```
    end
  if (DELTA<ZAHL) then                  //1*ld(ZAHL)
    begin
      MIN:=GUESS                        //1*ld(ZAHL)
    end
  if(DELTA=ZAHL)                        //1*ld(ZAHL)
    begin
      break                        //1*ld(ZAHL)
    end
  OLDGUESS:=GUESS                       //1*ld(ZAHL)
  GUESS:=MIN+MAX                        //2*ld(ZAHL)
  GUESS:=GUESS/2                        //2*ld(ZAHL)
end
```

6.5.2.2 Aufwand und Laufzeit der Wurzelfunktion

Um den Aufwand w(ZAHL) der ausgeführten Wurzelfunktion zu bestimmen, muss der angewendete Algorithmus näher untersucht werden.
Das Ergebnis wird durch immer stärkere Einschränkung des Lösungsintervalls bestimmt. Dazu werden das Anfangsintervall (hier von 0 bis ZAHL) bei jeder Iteration weiter halbiert und die Zahl in der Mitte des Intervalls als mögliche Lösung überprüft. Da das Intervall immer kleiner wird, konvergiert das Verfahren zwangsläufig nach einer bestimmten Anzahl an Durchgängen.
Lässt man als ZAHL nur Zweierpotenzen wie 2^n zu, dann wird das Intervall maximal *n*-Mal halbiert, da nur mit ganzzahliger Genauigkeit gerechnet wird. Für jede Zahl lässt sich die Anzahl der Iterationen also mit dem Exponenten der nächstgrößeren Zweierpotenz abschätzen.
Insgesamt erhalten wir für eine gegebene Größe ZAHL aufgerundet höchstens ld(ZAHL) an Iterationen.
Während eines Iterationsdurchganges trifft nur eine der drei if-Bedingungen zu. Wenn das Verfahren vorzeitig abgebrochen wird, ist bei diesem Wert von ZAHL der reale Aufwand natürlich geringer.
Da diese Berechnung sequentiell durchgefüht wird, unterscheiden sich die ermittelten Werte für Aufwand und Laufzeit nicht:

$$t(ZAHL) = w(ZAHL) = 5 + 12 \cdot ld(ZAHL)$$

6.5.2.3 Die Primzahlberechnung im PRAM-Modell

Hier wird auf die gleiche Weise vorgegangen wie bei der eben durchgeführten Kostenabschätzung der Wurzelfunktion. Dabei wird die einzige lokale Variable i zur Unterscheidung der Prozessoren verwendet. In dem pardo-Anweisungsblock wurde darauf geachtet, dass bei konditionalen Abfragen alle Prozessoren die gleichen Verzweigungen ausführen.
Auch hier ist zu jeder Zeile der Aufwand bereits im Pseudocode angegeben. Die abkürzenden Multiplikatoren *Schleife0*, *Schleife1*, *Schleife2*, *Schleife3* und *Schleife4* werden in der anschließenden Kostenberechnung angegeben und hergeleitet.

Pseudocode der Primzahlberechnung:

```
//Berechne Primzahlen bis n mit p Prozessoren
//Die Primzahlen werden in dem Feld DATEI[] aufsteigend gespeichert
//Anweisungen                       Kosten
a:=sqrt(p)                        //6+ld(p)*12
DATEI[1]:=2                       //1
DATEI[2]:=3                       //1
POINTER:=2                        //1
for j=4 to a do                       //2*Schleife0
  begin
    sqrt_j:=sqrt(j)                         //(6+ld(p)*12)*Schleife0
    for k=2 to sqrt_j do                    //2*Schleife1
      begin
        if j%k=0 then                 //2*Schleife1
          begin
            break                     //1*Schleife1
          end
      end
    if k=sqrt_j then                        //1*Schleife0
      begin
        DATEI[++POINTER]:=j           //2*Schleife0
      end
  end
for h=1 to (n-a)/p do             //4*Schleife2
  begin
    for 1<= i <= p pardo          //pardo-Loop*p bei Aufwand
      //Beginn parallele Berechnung
      begin
        //untersuche Zahl a+(h-1)*p+i
        TESTZAHL[i]:=a+(h-1)*p+i        //5*Schleife2
        POINTER[i]:=1             //1*Schleife2
        ERGEBNIS[i,0]:=TESTZAHL[i]      //1*Schleife2
          begin
            ZAHL[i]:=DATEI[POINTER]     //1*Schleife2
            sqrt_ahp:=sqrt(a+h*p)       //(6+ld(n)*12)*Schleife2
            while ZAHL[p]<=sqrt_ahp do          //1*Schleife4
              //Prozesse berechnen bis sqrt(a+h*p)
              begin
                ERGEBNIS[i,TESTZAHL[i]%ZAHL[i]]:=0  //2*Schleife4
```

```
                //an der Stelle ERGEBNIS[i,0]
                //steht am Schluss entweder 0 oder
                //TESTZAHL[i] ist eine Primzahl
                ZAHL[i]:=DATEI[++POINTER]            //2*Schleife4
              end
          end
      end                      //pardo-Loop Ende
      //Ende parallele Berechnung
    for j=1 to p do                        //2*Schleife3*Schleife2
      //überprüfe auf gefundene Primzahlen
      begin
        if (ERGEBNIS[j,0]!=0) then         //1*Schleife3*Schleife2
          begin
            DATEI[++POINTER]:=ERGEBNIS[j,0]     //2*Schleife3*Schleife2
          end
      end
end
```

6.5.2.4 Aufwand und Laufzeit der Primzahlberechnung

Damit der Aufwand und die Laufzeit in einem Ansatz ermittelt werden können, wird das Programm in den sequentiellen und den parallelen Teil zerlegt und der Aufwand für beide Abschnitte separat berechnet.

- Der sequentielle Abschnitt:

In diesen Teil des Programms fällt sowohl die sequentielle Initialisierung des Primzahlfeldes als auch die Vor- und Nachbereitung der Daten für das parallele Sieben der Primzahlen.

Im Folgenden werden die Laufzeiten der Schleifen berechnet, wobei für Schleife1 eine starke Abschätzung nach oben notwendig ist, um einen für weitere Rechnungen brauchbaren Term zu erhalten.

$$Schleife0 = \sum_{j=4}^{\sqrt{p}} 1 = \sqrt{p}-3$$

$$Schleife1 = \sum_{j=4}^{\sqrt{p}} \sum_{k=2}^{\sqrt{j}} 1 = \sum_{j=4}^{\sqrt{p}} \sqrt{j}-1 \leq (\sqrt{p}-3)\cdot(\sqrt{\sqrt{p}}-1) = \sqrt{p}\cdot\sqrt{\sqrt{p}}-\sqrt{p}-3\cdot\sqrt{\sqrt{p}}+3$$

$$Schleife2 = \sum_{h=1}^{\frac{n-\sqrt{p}}{p}} 1 = \frac{n-\sqrt{p}}{p}$$

$$Schleife3 = \sum_{j=1}^{p} 1 = p$$

Kosten außerhalb von Schleifen:

$$6+ld(p)\cdot 12+1+1+1 \;=\; 9+ld(p)\cdot 12$$

Kosten innerhalb *Schleife0* ohne *Schleife1*:

$$(2+6+ld(p)\cdot 12+1+2)\cdot(\sqrt{p}-3) \;=\; 12\cdot\sqrt{p}\cdot ld(p)+11\cdot\sqrt{p}+36\cdot ld(p)-33$$

Kosten innerhalb *Schleife1*:

$$(2+2+1)\cdot(\sqrt{p}\cdot\sqrt{\sqrt{p}}-\sqrt{p}-3\cdot\sqrt{\sqrt{p}}+3) \;=\; 5\cdot\sqrt{p}\cdot\sqrt{\sqrt{p}}-5\cdot\sqrt{p}-15\cdot\sqrt{\sqrt{p}}+15$$

Kosten innerhalb *Schleife2* ohne parallelen Abschnitt:

$$(4+2+1+2)\cdot\left(\frac{n-\sqrt{p}}{p}\right) \;=\; 9\cdot\frac{n-\sqrt{p}}{p} \;=\; 9\cdot n\cdot p^{-1}-9\cdot p^{-\frac{1}{2}}$$

Kosten innerhalb *Schleife3*:

$$(2+1+2)\cdot p \;=\; 5\cdot p$$

Insgesamt ergibt sich somit für die Laufzeit bzw. den Aufwand für diesen Teil:

$$\begin{aligned} t_1(n) \;&=\; 9+12\cdot ld(p) \\ &+\; 12\cdot\sqrt{p}\cdot ld(p)+11\cdot\sqrt{p}+36\cdot ld(p)-33 \\ &+\; 5\cdot\sqrt{p}\cdot\sqrt{\sqrt{p}}-5\cdot\sqrt{p}-15\cdot\sqrt{\sqrt{p}}+15 \\ &+\; 9\cdot\frac{n-\sqrt{p}}{p} \\ &+\; 5\cdot p \\ &=\; 9\cdot\frac{n-\sqrt{p}}{p}+5\cdot p+5\cdot p^{\frac{3}{4}}+12\cdot\sqrt{p}\cdot ld(p)+6\cdot\sqrt{p}-15\cdot\sqrt{\sqrt{p}}+48\cdot ld(p)-9 \end{aligned}$$

- Der parallele Abschnitt:

Dieser Teil des Programms ist sozusagen der Kern der Primzahlberechnung. Hier kommt insbesondere der Parameter n für die Kosten zum Tragen. Um eine handhabbare Formel zu erhalten wird bei *Schleife4* wieder eine großzügige Abschätzung durchgeführt.

$$\begin{aligned} Schleife4 \;&=\; \sum_{h=1}^{\frac{n-\sqrt{p}}{p}} \left[\, Primzahlen \le \sqrt{\sqrt{p}+h\cdot p} \,\right] \;\approx\; \sum_{h=1}^{\frac{n-\sqrt{p}}{p}} \frac{\sqrt{\sqrt{p}+h\cdot p}}{\ln\left(\sqrt{\sqrt{p}+h\cdot p}\right)} \\ &\le\; \frac{n-\sqrt{p}}{p}\cdot\frac{\sqrt{\sqrt{p}+\frac{n-\sqrt{p}}{p}\cdot p}}{\ln\left(\sqrt{\sqrt{p}+\frac{n-\sqrt{p}}{p}\cdot p}\right)} \;=\; \frac{n\cdot\sqrt{n}-\sqrt{n}\cdot\sqrt{p}}{p\cdot\ln(\sqrt{n})} \end{aligned}$$

Kosten innerhalb *Schleife2* ohne *Schleife4*:

$$(5+1+1+1+6+12\cdot ld(n))\cdot\frac{n-\sqrt{p}}{p} \;=\; 14\cdot\frac{n-\sqrt{p}}{p}+12\cdot ld(n)\cdot\frac{n-\sqrt{p}}{p}$$

Kosten innerhalb *Schleife4*:

$$(1+2+2)\cdot\frac{n\cdot\sqrt{n}-\sqrt{n}\cdot\sqrt{p}}{p\cdot\ln(\sqrt{n})} \;=\; 5\cdot\frac{n\cdot\sqrt{n}-\sqrt{n}\cdot\sqrt{p}}{p\cdot\ln(\sqrt{n})}$$

Insgesamt ergibt sich für die Laufzeit des parallelen Abschnitts:

$$t_2(n) \;=\; 14\cdot\frac{n-\sqrt{p}}{p}+12\cdot ld(n)\cdot\frac{n-\sqrt{p}}{p} \;+\; 5\cdot\frac{n\cdot\sqrt{n}-\sqrt{n}\cdot\sqrt{p}}{p\cdot\ln(\sqrt{n})}$$

- Zusammenfassend:

Damit beträgt die Laufzeit des Algorithmus:

$$\begin{aligned}
&t(n) \;=\; t_1(n)+t_2(n) \;=\\
&9\cdot\frac{n-\sqrt{p}}{p}+5\cdot p+5\cdot p^{\frac{3}{4}}+12\cdot\sqrt{p}\cdot ld(p)+6\cdot\sqrt{p}-15\cdot\sqrt{\sqrt{p}}+48\cdot ld(p)-9\\
&+\; 14\cdot\frac{n-\sqrt{p}}{p}+12\cdot ld(n)\cdot\frac{n-\sqrt{p}}{p} \;+\; 5\cdot\frac{n\cdot\sqrt{n}-\sqrt{n}\cdot\sqrt{p}}{p\cdot\ln(\sqrt{n})}\\
&=\; O\left(\frac{n\cdot\sqrt{n}}{\ln(\sqrt{n})}\right)
\end{aligned}$$

Und der gesamte Aufwand des Algorithmus ist somit:

$$\begin{aligned}
&w(n) \;=\; t_1(n)+p\cdot t_2(n) \;=\\
&9\cdot\frac{n-\sqrt{p}}{p}+5\cdot p+5\cdot p^{\frac{3}{4}}+12\cdot\sqrt{p}\cdot ld(p)+6\cdot\sqrt{p}-15\cdot\sqrt{\sqrt{p}}+48\cdot ld(p)-9\\
&+\; 14\cdot(n-\sqrt{p})+12\cdot ld(n)\cdot(n-\sqrt{p}) \;+\; 5\cdot\frac{n\cdot\sqrt{n}-\sqrt{n}\cdot\sqrt{p}}{\ln(\sqrt{n})}\\
&=\; O\left(\frac{n\cdot\sqrt{n}}{\ln(\sqrt{n})}\right)
\end{aligned}$$

Zur Übersicht wird hier noch die Laufzeit für die sequentielle Ausführung des Algorithmus (p=1) angegeben, zumal auf diesen Spezialfall später noch zurückgegriffen wirde:

$$\begin{aligned}
&t_{seq}(n) \;=\; t_1(n)+t_2(n) \;=\\
&(23+12\cdot ld(n))\cdot n+5+5+6-15-5\cdot\frac{\sqrt{n}}{\ln(\sqrt{n})}+5\cdot\frac{n\cdot\sqrt{n}}{\ln(\sqrt{n})}-9\\
&=\; 23\cdot n+12\cdot n\cdot ld(n)+5\cdot\frac{n\cdot\sqrt{n}-\sqrt{n}}{\ln(\sqrt{n})}-8
\end{aligned}$$

Ergänzend soll hier noch darauf hingewiesen werden, dass zur Primzahlsiebung parallele Verfahren existieren, die eine Laufzeit von $O(\log(n))$ bzw. $O(\sqrt{n})$ für ein festes *n* unter Berücksichtigung der benötigten Prozessoranzahl $O(n \cdot \log \log n / \log n)$ bzw. $O(\sqrt{n})$ haben [145]. Aber die hier vorausgesetzte Prozessoranzahl $O(1)$ und der wachsende Parameter *n* ermöglichen keinen Bezug zu diesen bekannten Algorithmen.

6.5.3 Laufzeitoptimierung bezüglich der Prozessoranzahl

Bisher haben wir den Algorithmus hauptsächlich unter dem Parameter n betrachtet. Allerdings hängt die Laufzeit für ein festes *n* stark von der Prozessoranzahl *p* ab. Für ein festes *n* kann *p* nur diskrete Werte aus folgendem Intervall annehmen:

$p \in [1, \ldots, n - \sqrt{(p)}]$

Klar ist, dass für aufsteigende Werte von *p* eine geringere Laufzeit erwartet werden kann, allerdings ist bei Werten in der Nähe von *n* zu vermuten, dass die Laufzeit wieder länger wird, denn dann wird der sequentielle Teil des Algorithmus – welcher unter anderem die Primzahlen bis zur Wurzel von *p* vorberechnet – einen größeren Anteil an der Laufzeit haben: schließlich hängt er fast nur von *p* ab.

Da die Formel rein algebraisch nicht auflösbar ist, wird die optimale Prozessoranzahl P(*n*) numerisch berechnet:

$P(n) = \{p \mid t(n,p) = min\{t(n,k) \mid k \in [1, \ldots, n - \sqrt{k}]\}\}$

n	P(n)	geschätzt: ln(n)*sqrt(n)
10	6	7
100	37	46
1000	168	218
10000	701	921
100000	2985	3641
1000000	13543	13816
10000000	65692	50970

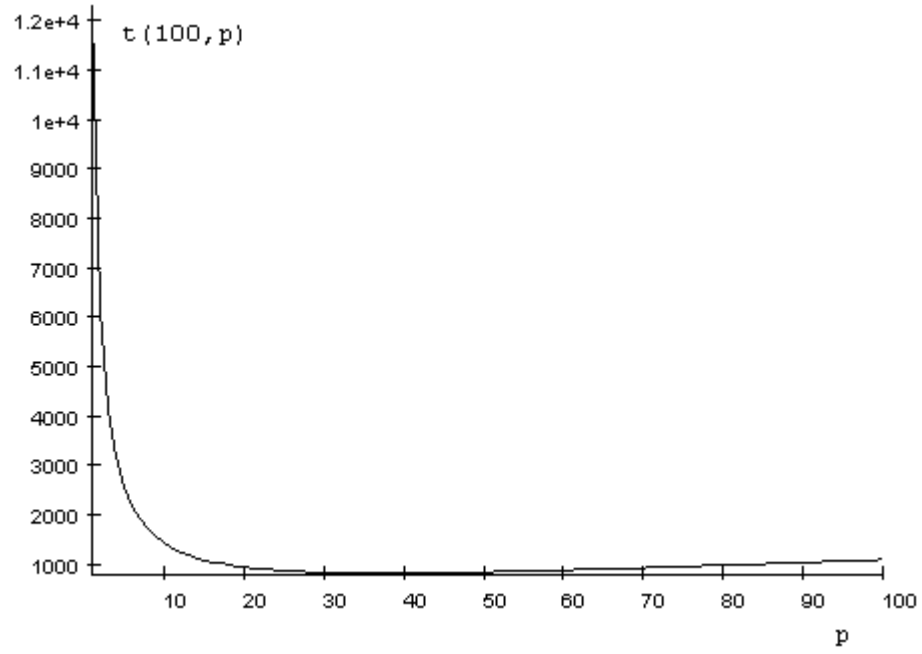
Die obigen Beispielwerte legen die Vermutung nahe, dass für die optimale

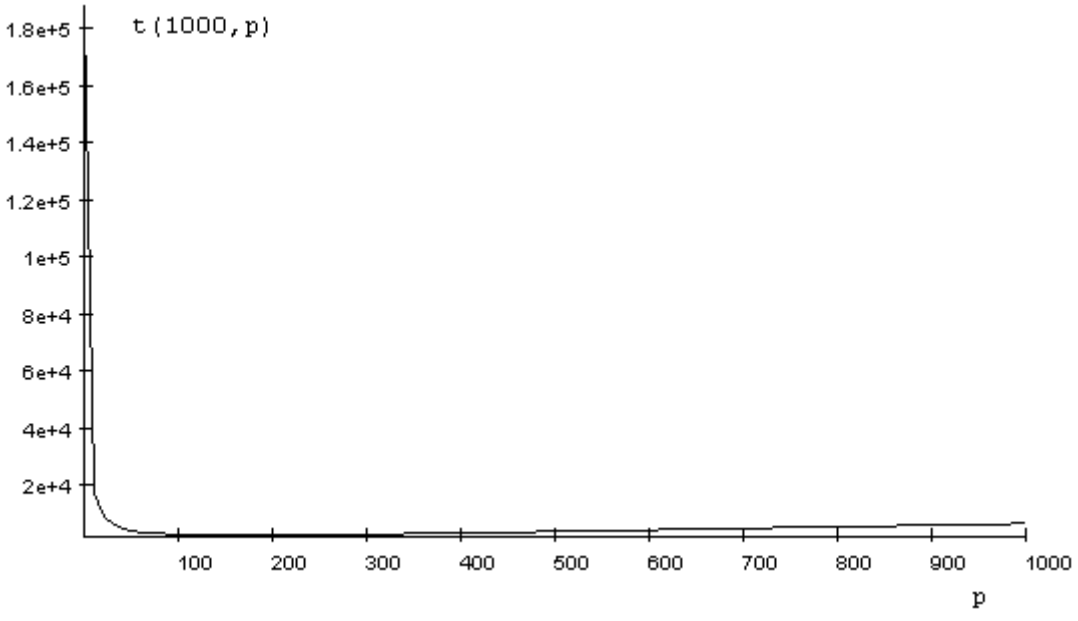
Prozessoranzahl in Abhängigkeit vom Parameter *n* folgendes Verhältnis gilt:

$$P(n) = O(\ln(n) \cdot \sqrt{n})$$

Dies lässt sich hier ohne weiteres leider nicht beweisen, aber der Vergleich mit den obigen Abschätzungen geben dieser Vermutung ein gewisses Gewicht.

Ergänzend zur optimalen Prozessoranzahl soll noch gezeigt werden, dass mit wesentlich weniger Prozessoren eine fast ebenso gute Laufzeit erreicht werden kann.
Zur Veranschaulichung dienen folgende Funktionsverläufe:





Der scharfe Knick, den die Funktion am Anfang beschreibt, wird mit zunehmendem Parameter *n* an die Ordinate gedrängt. Eine Prozessoranzahl im unteren Bereich des Knicks würde also fast die gleiche Laufzeit haben wie die optimale Prozessoranzahl, welche sich in dem anschließenden sehr gestreckten Kurvenabschnitt befindet. Geschätzte Angaben sind je nach gesetzter oberer Schranke sehr verschieden, aber für große Werte von *n* bringt eine Prozessoranzahl von mehr als der Wurzel von *n* keine wesentliche Verbesserung mehr.

6.5.4 Vergleiche mit den ermittelten Daten

Die in vorigen Abschnitten vorgestellten Messdaten des realisierten Programms sollen hier mit den theoretischen Werten verglichen werden.

6.5.4.1 Die Laufzeit

Ein wichtiges Kriterium für die Beurteilung des Algorithmus war die Laufzeit. Nun muss natürlich überprüft werden, ob die Laufzeitabschätzung mit der realen Laufzeit genügend übereinstimmt. Dazu wird ermittelt, mit welcher Konstante die Laufzeit abgeschätzt werden kann. Wenn sich für alle gemessenen Werte eine nahezu gleiche Konstante ergibt, so ist die ermittelte Abhängigkeit der Laufzeit von dem Parameter *n* praktisch erwiesen. Um die Störungen so gering wie möglich zu halten, betrachen wir nur den sequentiellen Fall:

$$\frac{T_{seq}(n)}{\frac{n\cdot\sqrt{n}}{\ln(\sqrt{n})}} = \frac{T_{seq}(n)\cdot\ln(\sqrt{n})}{n\cdot\sqrt{n}} \leq k$$

$$\frac{T_{seq}(10)\cdot\ln(\sqrt{10})}{10\cdot\sqrt{10}} = 0{,}1456$$

$$\frac{T_{seq}(100)\cdot\ln(\sqrt{100})}{100\cdot\sqrt{100}} = 0{,}01612$$

$$\frac{T_{seq}(1000)\cdot\ln(\sqrt{1000})}{1000\cdot\sqrt{1000}} = 0{,}01431$$

$$\frac{T_{seq}(10000)\cdot\ln(\sqrt{10000})}{10000\cdot\sqrt{10000}} = 0{,}01468$$

$$\frac{T_{seq}(100000)\cdot\ln(\sqrt{100000})}{100000\cdot\sqrt{100000}} = 0{,}01456$$

$$\frac{T_{seq}(1000000)\cdot\ln(\sqrt{1000000})}{1000000\cdot\sqrt{1000000}} = 0{,}01456$$

Wie bei den anderen Vergleichen auch sind die Ergebnisse für niedrige Werte von *n* (10 und 100) sehr ungenau. Mit steigendem Parameter *n* erwartet man eine aussagekräftige Abschätzung der Konstante, welche augenscheinlich auch eintritt, denn der Quotient wird kleiner. Also würde *k*=0,2 genügen. Allerdings muss bedacht werden, dass in der Praxis außerhalb des Algorithmus Verzögerungen entstehen können, welche ebenfalls von der Komplexität *n* abhängen, wie zum Beispiel der Zugriff auf die Primzahldatei. Auch wenn dieser Faktor hier noch unbemerkt bleibt, so ist für sehr große *n* die Limitierung der Hardware möglicherweise entscheidend einschränkend.
Für die vorliegenden Ergebnisse kann man die ermittelte Abschätzung für die Laufzeit aus der Analyse des Algorithmus als erwiesen ansehen.

6.5.4.2 Die Beschleunigung im PRAM-Modell

Es soll ermittelt werden, welchen Geschwindigkeitsvorteil die Ausführung des Algorithmus mit zwei anstatt nur einem Prozessor in der Theorie bringt. Im anschließenden Abschnitt werden dann zum Vergleich die ermittelten Laufzeiten der Testläufe verglichen.

Man betrachtet folgenden Quotienten:

$$Beschleunigung(Speedup) = S_2(n) = \frac{t_{seq}(n)}{t_{p=2}(n)} = \frac{t_1(n|p=1)+t_2(n|p=1)}{t_1(n|p=2)+t_2(n|p=2)}$$

Jetzt wird für ausgewählte Werte von *n* der Speedup berechnet:

$$S_2(10) = \frac{t_{seq}(10)}{t_{p=2}(10)} \approx \frac{681}{377} \approx 1{,}81$$

$$S_2(100) = \frac{t_{seq}(100)}{t_{p=2}(100)} \approx \frac{12312}{6182} \approx 1{,}99$$

$$S_2(10000) = \frac{t_{seq}(10000)}{t_{p=2}(10000)} \approx \frac{2909963}{1454973} \approx 2{,}00$$

$$S_2(100000000) = \frac{t_{seq}(100000000)}{t_{p=2}(100000000)} \approx \frac{577058606300}{288529302000} \approx 2{,}00$$

Aus diesen errechneten Werten wird deutlich, dass sich die Laufzeit bei Benutzung von zwei Prozessoren etwa halbiert – bei wachsendem *n* wird dies noch unterstrichen, da der parallele Abschnitt des Algorithmus den größten Anteil am Aufwand hat.
Leider läßt sich bei diesem Algorithmus die Beschleunigung nicht gut abschätzen, denn weder das prozentuale Verhältnis des sequentiellen Anteils zur Laufzeit (Amdahlsches Gesetz) noch der sequentielle Anteil der Laufzeit an sich ist konstant (Gustafson-Gesetz) [130].
Den höchsten Speedup bei festem *n* hat man trivialerweise bei der optimalen Prozessoranzahl. Aber ähnlich dem Gustafson-Gesetz nähert sich die Beschleunigung der Prozessoranzahl *p* offensichtlich asymptotisch an (wenn *n* sehr viel größer als *p* ist).

6.5.4.3 Die Beschleunigung im realen Programm

Analog zu den obigen Vergleichen soll der Speedup zwischen der sequentiellen und der parallelen Laufzeit ermittelt werden:

$$\frac{T_{seq}(10)}{T_{p=2}(10)} \approx \frac{4}{2} \approx 2{,}00$$

$$\frac{T_{seq}(100)}{T_{p=2}(100)} \approx \frac{7}{4} \approx 1{,}75$$

$$\frac{T_{seq}(1000)}{T_{p=2}(1000)} \approx \frac{131}{73} \approx 1{,}79$$

$$\frac{T_{seq}(10000)}{T_{p=2}(10000)} \approx \frac{3187}{1738} \approx 1{,}83$$

$$\frac{T_{seq}(100000)}{T_{p=2}(100000)} \approx \frac{79977}{41119} \approx 1{,}95$$

$$\frac{T_{seq}(1000000)}{T_{p=2}(1000000)} \approx \frac{2107634}{1094554} \approx 1{,}93$$

Auch hier wird deutlich, dass sich die Beschleunigung bei wachsendem *n* dem Wert 2 annähert. Allerdings ist der – nicht im Algorithmus enthaltene – systembedingte Overhead deutlich größer, weswegen eine geringe Abweichung von der theoretischen Laufzeit immer auftritt..
Der Wert für *n*=10 ist aus messtechnischen Gründen sehr ungenau und beinhaltet daher keine Aussagekraft.

6.6 Zusammenfassung und Ausblick

In dem vorigen Kapitel habe ich bereits betont, dass die konkrete Berechnung einer Primzahl von einer bestimmten Größe nicht trivial ist. Die Aufgabe des oben entwickelten Programms ist die reale parallele Berechnung von Primzahlen mit einem Algorithmus, der möglichst gut die gegebenen Anforderungen umsetzt. Dennoch wird durch die Auswertung des Algorithmus deutlich, dass es eines sehr hohen Aufwandes bedarf, um Primzahlen in der gewünschten Größe zu berechnen. Dies mag einer der Gründe dafür sein, dass Primzahlberechnungen hauptsächlich in Forschungseinrichtungen mit entsprechender Hardwareausstattung durchgeführt werden. Die bisherigen Ergebnisse können dann anderen – beispielsweise durch Primzahldateien – zugänglich gemacht werden.

In dem im 5. Kapitel beschriebenen Einsatzgebiet von großen Primzahlen, den asymmetrischen Verschlüsselungsverfahren in der Kryptographie (speziell der RSA), kann es aber aus Gründen der Sicherheit notwendig sein, selbst entsprechende Primzahlen zu berechnen: Bei einem so heiklen Einsatzgebiet will man unter Umständen den möglicherweise kompromittierten Primzahlsammlungen anderer nicht vertrauen.
In einem solchen Szenario würde man also für einen begrenzten Zeitraum extrem viel Rechenleistung benötigen (bis man zum Beispiel alle Primzahlen bis 2^{4096} berechnet hat). Für diese – vermutlich einmalige – Aufgabe würde es sich für ein Unternehmen nicht lohnen zusätzliche Hardware zu kaufen. Hier wäre ein sehr gutes Einsatzgebiet für Grids. Das Unternehmen könnte alle ihm zur Verfügung stehenden Rechner – ungeachtet ihrer wahrscheinlichen Inhomogenität – in ihren weltweiten Niederlassungen zu einem Computational Grid zusammenschließen. Die tatsächlich zur Verfügung gestellte Rechenleistung müsste lokal gesteuert werden, um andere Projekte nicht zu beeinträchtigen bzw. um ungenutzte Rechenleistung außerhalb der Arbeitszeiten globusweit aktivieren zu können (zum Beispiel nach Geschäftsschluss).
Ergänzend wäre auch ein Erweitern der eigenen Rechenkapazität für diese Aufgabe möglich, indem Ressourcen von anderen Firmen angemietet und in das Grid integriert werden.
Tatsächlich könnten sich kleinere Unternehmen mit einer Grid-Infrastruktur bei Bedarf gegenseitig aushelfen und so die Anschaffung und Wartung von selten genutzten Überkapazitäten vermeiden.
Unter diesem Gesichtspunkt sollte man also die vorgenommene Realisierung der Primzahlberechnung auf einem Grid sehen. Sie soll stellvertretend für Projekte mit enormen Ressourcenbedarf stehen, welche allerdings nur einmalig oder selten ausgeführt werden müssen. Die mir zur Verfügung stehende Hardware war für diese Aufgabe natürlich völlig unzureichend – zwei langsame Prozessoren und minimaler Hauptspeicher ließen keine aussagekräftigen Vergleiche zwischen dem Algorithmus und der real gemessenen Laufzeit zu. Auch dauerte ein Testlauf mit relativ niedrigen Zahlen mehrere Tage, so dass das asymptotische Verhalten nicht ausreichend überprüft werden konnte. Dennoch entsprechen die aufgezeigten Entwicklungsschritte und Überlegungen denen anderer vergleichbarer Projekte und können als solche nicht nur für sich selbst stehen sondern auch die Vorgehensweise für andere Entwicklungen aufzeigen.

Schlusswort

In dieser Arbeit wurde ein Thema behandelt, das innerhalb weniger Jahre vom theoretischen Konzept bis zur kommerziellen Anwendung gereift ist. Dies ist einerseits natürlich dem schnell wachsenden Globus Projekt zu verdanken, das aufzeigte, dass ein Grid im Sinne von Ian Foster et al. praktisch umsetzbar und effektiv ist. Andererseits ist die rasante Entwicklung auch auf das enorme Potential, welches offensichtlich in ihm liegt, zurückzuführen. Alle großen IT-Unternehmen beschäftigen sich mit diesem Thema und einige schmücken ihre Produkte gerne mit dem Ausdruck "Grid", auch wenn diese gar nichts mit dem allgemein anerkannten Begriff – wie er im ersten Kapitel vorgestellt wurde – zu tun haben. Ein weiterer Grund für die zunehmende Verbreitung von Grids ist die dafür leicht zu erfüllende Anforderung an die Infrastruktur, denn alle klassischen Ressourcen der IT-Branche sind durch ihre Anbindung an das Internet schon miteinander vernetzt, wobei die Bandbreite stetig steigt (zurzeit führt die deutsche Telekom DSL-Pilotprojekte mit bis zu 60 MBit Bandbreite durch). Auch die nötige Middleware ist für klassische Rechnerarchitekturen durch das Globus Toolkit kostenlos zu bekommen. Interessierte kommerzielle Ressourcen-Provider (wie IBM) müssen sich lediglich um die Vertriebswege kümmern und das Produkt vermarkten.
In diesem Zusammenhang besteht das Hauptanliegen der vorliegenden Arbeit im Informieren und Motivieren bezüglich des Grid Computing.
Es wurde von der (gebräuchlichen) Definition über die frühen Pilotprojekte bis hin zu Grids in der Forschung und für kommerzielle Anwendung ein Überblick gegeben, welcher mehrere aktuelle Anwendungsgebiete von Grids umreißt.
Auch wurden verschiedene Grid-Middleware Lösungen vorgestellt, von denen einige zukünftig mittels der OGSI-Spezifikationen miteinander harmonieren werden - man muss sich also nicht auf ein Produkt festlegen.
Kommerzielle Anbieter wie beispielsweise Sun oder Entropia versprechen mit ihren Produkten Komplettlösungen für bestimmte Anwendungsgebiete. Aber durch die hier vorgestellte Installationsbeschreibung des Globus Toolkits (welches ja auch Bestandteil einiger kommerzieller Distributionen ist) konnte gezeigt werden, dass die kostenlose Open-Source-Lösung problemlos verwendet werden kann.

Spezielle Erweiterungen für Grids, wie MPICH-G2 für das Globus Toolkit, versprechen eine problemlose Anpassung von bisherigen (parallelen) Programmen an eine Grid-Architektur. Die Entwicklung von Werkzeugen und Erweiterungen für Grids ist zwar noch sehr jung und bisher hauptsächlich proprietär – schließlich wurden bis vor kurzem die Grids an sich noch entwickelt – aber durch eine middlewareunabhängige Schnittstelle, wie sie die OGSA verspricht, wird dieser Sektor noch weiter belebt.
Die Seite des Anwenders bzw. Nutzers eines Grids wurde durch die Programmentwicklung am Ende der Arbeit gezeigt. Dabei ist aufgefallen, dass insbesondere bei der Nutzung von bekannten Schnittstellen wie MPI der Gedanke an ein Grid keine wirkliche Rolle gespielt hat. Für paralleles Rechnen ist die Grid-Struktur also quasi transparent und kann problemlos genutzt werden.
Ich hoffe also auch die Nutzung von Grids genügend motiviert und die Schwelle zum Umstieg gemindert zu haben.
Ein Problem allerdings haftet jedem Überblick an: die Aktualität. Gerade in dem sich sehr dynamisch entwickelnden Bereich des Grid Computing dauerte eine Generation bisher etwa zwei Jahre. Der Vorreiter im Bereich der Middleware ist und bleibt wohl vorerst das Globus Toolkit, was insbesondere für Entwickler eine gewisse Kontinuität verspricht und die Generationswechsel somit verlangsamen sollte.
Abschließend bleibt noch anzumerken, dass auch die hier vorgestellten technischen Konzepte und deren Umsetzungen in Zukunft zwar erweitert, jedoch nicht ungültig werden, denn die heutige Entwicklung für die Programme von morgen beruht auf ihnen.

Anhang

Der Programmtext von `ring.c`

```
/* Parameteruebergabe: ring {-t <ntrips>} {-v} */
#include <stdio.h>
#include <mpi.h>
int Ntrips;  /* -t <ntrips> */
int Verbose; /* -v */

int parse_command_line_args(int argc, char **argv, int my_id)
{
  int i;
  int error;
  Ntrips = 1;
  Verbose = 0;
  for (i = 1, error = 0; !error && i < argc; i ++)
  {
    if (!strcmp(argv[i], "-t"))
    {
      if (i + 1 < argc && (Ntrips = atoi(argv[i+1])) > 0)
        i ++;
      else
        error = 1;
    }
    else
      if (!strcmp(argv[i], "-v"))
        Verbose = 1;
      else
        error = 1;
  } /* endfor */
  if (error && !my_id)
  {
    /* Nur der Hauptprozess gibt die Parameterliste aus */
    fprintf(stderr, "\n\tAusführung: %s {-t <ntrips>} {-v}\n\n",
argv[0]);
    fprintf(stderr, "wobei\n\n");
    fprintf(stderr, "\t-t <ntrips>\t- Anzahl der zu durchlaufenden
Runden im Ring. Standardwert ist 1.\n");
    fprintf(stderr, "\t-v\t\t- Ausführlich. Der Hauptprozess und alle
anderen Prozesse loggen jeden Schritt. \n");
    fprintf(stderr, "\t\t\t  Standardwert ist NEIN.\n\n");
  } /* endif */
  return error;
} /* Ende von parse_command_line_args() */

main(int argc, char **argv)
{
  int numprocs, my_id, passed_num;
  int trip;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
  if (parse_command_line_args(argc, argv, my_id))
  {
```

```
    MPI_Finalize();
    exit(1);
  } /* endif */
  if (Verbose)
    printf("my_id %d numprocs %d\n", my_id, numprocs);
  if (numprocs > 1)
  {
    if (my_id == 0) /* Ich bin der Hauptprozess */
    {
      passed_num = 0;
      for (trip = 1; trip <= Ntrips; trip ++)
      {
        passed_num ++;
        if (Verbose)
          printf("Hauptprozess: starte Runde %d von %d: vor dem Senden
von num=%d zu Ziel=%d\n", trip, Ntrips, passed_num, 1);
        MPI_Send(&passed_num,    /* buff  */
                 1,              /* count */
                 MPI_INT,        /* type  */
                 1,              /* dest  */
                 0,              /* tag   */
                 MPI_COMM_WORLD); /* comm  */
        if (Verbose)
          printf("Hauptprozess: innerhalb Runde %d von %d: vor dem
Empfangen von Quelle=%d\n", trip, Ntrips, numprocs-1);
        MPI_Recv(&passed_num,    /* buff   */
                 1,              /* count  */
                 MPI_INT,        /* type   */
                 numprocs-1,     /* source */
                 0,              /* tag    */
                 MPI_COMM_WORLD, /* comm   */
                 &status);       /* status */
        printf("Hauptprozess: Ende der Runde %d von %d: nach dem
Empfangen von passed_num=%d (sollte =Runde*numprocs=%d) von Quelle=%
d\n", trip, Ntrips, passed_num, trip*numprocs, numprocs-1);
      } /* Ende der for-Anweisung */
    }
    else /* Ich bin ein normaler Prozess */
    {
      for (trip = 1; trip <= Ntrips; trip ++)
      {
        if (Verbose)
          printf("Prozess %d: Anfang der Runde %d von %d: vor dem
Empfangen von Quelle=%d\n", my_id, trip, Ntrips, my_id-1);
        MPI_Recv(&passed_num,    /* buff   */
                 1,              /* count  */
                 MPI_INT,        /* type   */
                 my_id-1,        /* source */
                 0,              /* tag    */
                 MPI_COMM_WORLD, /* comm   */
                 &status);       /* status */
        if (Verbose)
          printf("Prozess %d: innerhalb Runde %d von %d: nach dem
Empfangen von passed_num=%d von Quelle=%d\n", my_id, trip, Ntrips,
passed_num, my_id-1);
        passed_num ++;
        if (Verbose)
          printf("Prozess %d: innerhalb Runde %d von %d: vor dem Senden
von passed_num=%d zu Ziel=%d\n", my_id, trip, Ntrips, passed_num,
(my_id+1)%numprocs);
        MPI_Send(&passed_num,       /* buff  */
                 1,                 /* count */
                 MPI_INT,           /* type  */
```

```
                (my_id+1)%numprocs, /* dest  */
                0,                  /* tag   */
                MPI_COMM_WORLD);    /* comm  */
        if (Verbose)
          printf("Prozess %d: Ende der Runde %d von %d: nach dem Senden
zu Ziel=%d\n", my_id,trip, Ntrips, (my_id+1)%numprocs);
      } /* Ende der for-Anweisung */
    } /* Ende der if-Anweisung */
  }
  else
    printf("numprocs = %d, sollte nur mit numprocs > 1 gestartet
werden!\n",numprocs);
  MPI_Finalize();
} /* Ende von  main() */
```

Der Programmtext von `ldaptest.c`

```
/* Es werden keine Parameter uebergeben! */
#include <stdio.h>
#include <stdlib.h>
#include <lber.h>
#include <ldap.h>

/* setzte maximale Filterlaenge */
#define LDAP_FILTER_NAME_MAX 1024
/* setzte maximale Rechner im GIIS (Grid) */

/* Funktion zum Auslesen eines Wertes zu einem Attribut */
/* AttGetVal (SessionID, CurrentPosition, SearchString) */
double AttGetVal(LDAP *ld, LDAPMessage *ldap_msg_obj,char *searchstring)
{
  BerElement *ber;
  char *ldapAttr;
  char **char_attr;

  ldapAttr = ldap_first_attribute(ld, ldap_msg_obj, &ber);
  if (!strcmp(searchstring, ldapAttr)) {
        char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
        return(atof(char_attr[0]));
  }
  while ((ldapAttr = ldap_next_attribute(ld,ldap_msg_obj, ber))!=NULL) {
        if (!strcmp(searchstring, ldapAttr)) {
          char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
          return(atof(char_attr[0]));
        }
  }
  return(0);
}

/* Funktion zum Auslesen des Attributnamens */
/* AttGetStr (SessionID, CurrentPosition, SearchString) */
char * AttGetStr(LDAP *ld, LDAPMessage *ldap_msg_obj, char
*searchstring)
{
  BerElement *ber;
  char *ldapAttr;
  char **char_attr;
```

```
  ldapAttr = ldap_first_attribute(ld, ldap_msg_obj, &ber);
  if (!strcmp(searchstring, ldapAttr)) {
        char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
        return(* char_attr);
  }
  while ((ldapAttr = ldap_next_attribute(ld, ldap_msg_obj,ber))!=NULL) {
        if (!strcmp(searchstring, ldapAttr)) {
          char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
          return(* char_attr);
        }
  }
  return("");
}

/* Hauptprogramm */
main(int argc, char *argv[]) {

  char *hostName;
  int result, count, i;
  char *myLdapServer;
  char *myLdapBase;
  char *mySite;
  int myLdapScope;
  char myLdapFilter[LDAP_FILTER_NAME_MAX];
  char **myLdapAttr = NULL;
  int myLdapPort = 2135;
  char *char_dn;
  LDAP *ld;
  LDAPMessage *res;
  LDAPMessage *ldap_msg;
  char *strptr;
  double val;

  /* Vorgabe: GIIS Suche */
  /* Diese Werte koennen/muessen angepasst werden! */
  myLdapServer = (char *)strdup("kafka.informatik.uni-giessen.de");
  myLdapBase = (char *)strdup("mds-vo-name=site, o=grid");
  mySite = (char *)strdup("Mds-Host-hn=*");

  /* Setze die Ausdehnung der Suche */
  /* myLdapScope = LDAP_SCOPE_ONELEVEL; */
  /* myLdapScope = LDAP_SCOPE_BASE; */
  myLdapScope = LDAP_SCOPE_SUBTREE;

  printf("\nConnecting to GIIS (%s) at %s:%d ...\n\n", myLdapBase,
myLdapServer, myLdapPort);

  /* Stelle die Verbindung zum LDAP-Verzeichnis her */
  if ((ld = ldap_init(myLdapServer, myLdapPort)) == NULL) {
      printf("Error: ldap_open failed");
      exit(-1);
  }

  /* Fuehre Testzugriff auf das LDAP-Verzeichnis durch */
  result = ldap_simple_bind_s(ld, "", "");
  if(result != LDAP_SUCCESS) {
      printf("Error: ldap_simple_bind_s failed!\n");
      exit(-1);
  }

  /* Setze den Suchfilter zusammen */
  sprintf(myLdapFilter, "%s", mySite);
```

```
  /* Die eigentliche Suche */
  result=ldap_search_s(ld, myLdapBase, myLdapScope, myLdapFilter,
myLdapAttr, 0, &res);

  if(result != LDAP_SUCCESS) {
      printf("Error: ldap_search_s failed!\n");
      exit(-1);
  }

  /* Anzahl der gefundenen Eintraege */
  count = ldap_count_entries(ld, res);
  printf("Entries found for %s: %d\n", myLdapFilter, count);
  if(count < 1) {
      printf("No matches found!\n");
      exit(0);
  }

  /* Auslesen der gefundenen Eintraege */
  /* Erster Eintrag */
  i=1;
  ldap_msg = ldap_first_entry(ld, res);
  char_dn = ldap_get_dn(ld, ldap_msg);
  printf("\nQuery found DN (%d): %s\n",i, char_dn);

  hostName = AttGetStr(ld, ldap_msg, "Mds-Host-hn");
  printf("%s for %s is %s\n", "Mds-Host-hn", hostName, hostName);
  val = AttGetVal(ld, ldap_msg, "Mds-Cpu-Total-count");
  printf("%s for %s is %d\n","Mds-Cpu-Total-count",hostName,(int)val);
  val = AttGetVal(ld, ldap_msg, "Mds-Cpu-Total-Free-15minX100");
  printf("%s for %s is %d\n", "Mds-Cpu-Total-Free-15minX100", hostName,
(int)val);
  val = AttGetVal(ld, ldap_msg, "Mds-Memory-Ram-Total-freeMB");
  printf("%s for %s is %d\n", "Mds-Memory-Ram-Total-freeMB", hostName,
(int)val);
  val = AttGetVal(ld, ldap_msg, "Mds-Memory-Vm-Total-freeMB");
  printf("%s for %s is %d\n", "Mds-Memory-Vm-Total-freeMB", hostName,
(int)val);
  val = AttGetVal(ld, ldap_msg, "Mds-Fs-Total-freeMB");
  printf("%s for %s is %d\n", "Mds-Fs-Total-freeMB", hostName, (int)
val);

  /* Restliche Eintraege  */
  while (i<count) {
    i++;
    ldap_msg = ldap_next_entry(ld, ldap_msg);
    char_dn = ldap_get_dn(ld, ldap_msg);
    printf("\nQuery found DN (%d): %s\n",i, char_dn);

    hostName = AttGetStr(ld, ldap_msg, "Mds-Host-hn");
    printf("%s for %s is %s\n", "Mds-Host-hn", hostName,hostName);
    val = AttGetVal(ld, ldap_msg, "Mds-Cpu-Total-count");
    printf("%s for %s is %d\n", "Mds-Cpu-Total-count", hostName, (int)
val);
    val = AttGetVal(ld, ldap_msg, "Mds-Cpu-Total-Free-15minX100");
    printf("%s for %s is %d\n", "Mds-Cpu-Total-Free-15minX100",
hostName, (int)val);
    val = AttGetVal(ld, ldap_msg, "Mds-Memory-Ram-Total-freeMB");
    printf("%s for %s is %d\n", "Mds-Memory-Ram-Total-freeMB", hostName,
(int)val);
    val = AttGetVal(ld, ldap_msg, "Mds-Memory-Vm-Total-freeMB");
    printf("%s for %s is %d\n", "Mds-Memory-Vm-Total-freeMB", hostName,
(int)val);
```

```
    val = AttGetVal(ld, ldap_msg, "Mds-Fs-Total-freeMB");
    printf("%s for %s is %d\n", "Mds-Fs-Total-freeMB", hostName, (int)
val);
  }
}
```

Der Programmtext von `gridprime.c`

```
/* Parameteruebergabe: gridprime {-n <int Zahl>}
                                 {-v}
                                 {-r <URL Datei>}
                                 {-s <int Schrittweite>}
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <lber.h>
#include <ldap.h>
#include <mpi.h>
#include <globus_gass_file.h>

#define MASTER 0
#define MAX_PROCESSES 255
#define MAX_BIN_LONG 17 /* rechne bis max. 17*31=527 Bit */
#define MAX_DEC_INT 159 /* 159>lg(2^527) */
#define MAX_STEP 100    /* maximale Schrittweite */

#define FILE_NAME "/tmp/primes.dat" /* Standardname der Primzahldatei */

/* LDAP-Definitionen */
#define LDAP_FILTER_NAME_MAX 1024
/* Maximum hosts in GIIS (Grid) */

unsigned long number[MAX_STEP][MAX_BIN_LONG];  /* die Testzahl(en)  */
unsigned long binnumber0[MAX_BIN_LONG]; /* Zahl 0 */
unsigned long binnumber1[MAX_BIN_LONG]; /* Zahl 1 */
unsigned long solution[MAX_STEP][MAX_BIN_LONG]; /* das Ergebnis */
unsigned long loopnumber[MAX_BIN_LONG]; /* Zahl aus der Primzahldatei*/
unsigned long maxnumber[MAX_BIN_LONG];  /* maximal moegliche Zahl */
unsigned long sqrtnumber[MAX_BIN_LONG]; /* Wurzel von number */
unsigned long tmp[MAX_BIN_LONG];
unsigned long prime_cache[MAX_PROCESSES][MAX_STEP][MAX_BIN_LONG];

unsigned decsolution[MAX_DEC_INT];      /* Dezimalzahl zu solution */
unsigned int schrittweite=1;            /* eine Zahl pro Prozess */

main (int argc, char **argv) {
  unsigned long i,j,s,N=0;
  durchgesiebt=0;
  int zahlenanzeige=0;
  int wiederaufnahme=0;
  int client, me, total, tag=99;
  MPI_Status status;
  FILE *fp;
  char FILE_PATH[255];
  /* Stoppuhr */
  double starttime, endtime;
```

```
  globus_module_activate(GLOBUS_GASS_FILE_MODULE);

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &me);
  MPI_Comm_size(MPI_COMM_WORLD, &total);

  load_bin_with_dec(binnumber0,0);  /* statische Zahl 0 */
  load_bin_with_dec(binnumber1,1);  /* statische Zahl 1 */
  load_bin_with_dec(number[0],2);   /* Initialisiere Testzahl mit 2 */

  /* setze technische Obergrenze der Berechnung fest */
  for (i=0;i<MAX_BIN_LONG;i++) {
    maxnumber[i]=(unsigned long)0x7FFFFFFF;
  }

  /* Werte Parameter aus */
  for (i=2;i<=argc;i++) {
    /* Benchmark fuer ein festes N */
    if (vergleiche(argv[i-1],"-n")==1 && argc>i) {
      N=(unsigned int) atoi(argv[i]);
      load_bin_with_dec(maxnumber,N);
    }
    /* erhoehe Schrittweite */
    if (vergleiche(argv[i-1],"-s")==1 && argc>i) {
      schrittweite=(unsigned int) atoi(argv[i]);
    }
    /* gebe jede Primzahl aus */
    if (vergleiche(argv[i-1],"-v")==1) {
      zahlenausgabe=1;
    }
    /* setze begonnene Rechnung fort */
    if (vergleiche(argv[i-1],"-r")==1 && argc>i) {
      wiederaufnahme=1;
      /* lese Pfad fuer Primzahldatei ein */
      kopiere(FILE_PATH, argv[i]);
      /* lese zuletzt berechnete Primzahl ein */
      fp = globus_gass_fopen(FILE_PATH, "rb");
      fseek(fp,-1*(MAX_BIN_LONG+1)*sizeof(number[0][0]),SEEK_END);
      for (i=0;i<MAX_BIN_LONG;i++) {
        j=fread(&number[0][i],sizeof(number[0][i]),1,fp);
      }
      globus_gass_fclose(fp);
      /* setze naechste Zahl zum Ueberpruefen fest */
      j=add_binnumber(number[0],binnumber1,number[0]);
      if (get_bit(number[0],0,0)==0) {
        j=add_binnumber(number[0],binnumber1,number[0]);
      }
    }
  }
  if (total==1) {
    printf("Error: At least 2 processes must be started!\n");
    printf("Fehler: Mindestens 2 Prozesse muessen gestartet werden!\n");
    exit(1);
  }

if (wiederaufnahme!=1) {
    /* Lege Ort fuer Primzahldatei fest */
    /* Sehr stark von den Bibliotheken des Toolkits abhaengig! */
    kopiere(FILE_PATH, GetFileHost());
    if (vergleiche(FILE_PATH,"NULL")==0) {
      printf("No GIIS found! Force local File with one real
Machine!\n");
```

```
      printf("Kein GIIS gefunden! Erzwinge lokale Datei!\n");
      sprintf(FILE_PATH,"%s",FILE_NAME);
    }
    else {
      printf("Best Host: %s\n",FILE_PATH);
      printf("Bester Rechner: %s\n",FILE_PATH);
      sprintf(FILE_PATH,"gsiftp://%s:2811%s",FILE_PATH,FILE_NAME);
    }
  }

  /* Hauptprozess */
  if (me==MASTER) {

    if (wiederaufnahme!=1) {
      /* Berechne die ersten Primzahlen sequentiell */
      number[0][0]= generate_initial_file(total*schrittweite);
    }

    /* Starte Stoppuhr */
    if (N>0) {
      starttime=MPI_Wtime();
    }

    /* Sende jedem Prozess <schrittweite> Zahlen */
    for (i=1;i<total;i++) {
      for (s=1;s<=schrittweite;s++) {
        for (j=0;j<MAX_BIN_LONG;j++) {
          MPI_Send(&number[0][j], 1, MPI_LONG, i, tag, MPI_COMM_WORLD);
        }
        j=add_binnumber(number[0],binnumber1,number[0]);
        if (get_bit(number[0],0,0)==0) {
          j=add_binnumber(number[0],binnumber1,number[0]);
        }
      }
    }

    /* Hauptschleife */
    while (sub_binnumber(maxnumber,number[0],tmp)==0) {
      for (i=1;i<total;i++) {
        for (s=1;s<=schrittweite;s++) {
          if (s==1) {
            MPI_Recv(&solution[s][0], 1, MPI_LONG, MPI_ANY_SOURCE, tag,
MPI_COMM_WORLD, &status);
          }
          else {
            MPI_Recv(&solution[s][0], 1, MPI_LONG, client, tag,
MPI_COMM_WORLD, &status);
          }
          client=status.MPI_SOURCE;
          for (j=1;j<MAX_BIN_LONG;j++) {
            MPI_Recv(&solution[s][j], 1, MPI_LONG, client, tag,
MPI_COMM_WORLD, &status);
          }
          if (equal_binnumber(solution[s],binnumber0)!=0) {
            if (zahlenausgabe!=0) {
              printf("Process %d reports prime ", status.MPI_SOURCE);
              bin_to_dec(solution[s],decsolution);
              show_decnumber(decsolution);
              printf("\n");
            }
            load_binnumber(prime_cache[status.MPI_SOURCE][s],solution
[s]);
          }
```

```
          else {
            load_bin_with_dec(prime_cache[status.MPI_SOURCE][s],0);
          }
        }
      }
      /* Speichere den Primzahlencache */
      fp = globus_gass_fopen(FILE_PATH, "ab");
      for (j=1;j<total;j++) {
        for (s=1;s<=schrittweite;s++) {
          if (equal_binnumber(prime_cache[j][s],binnumber0)!=0) {
            for (i=0;i<MAX_BIN_LONG;i++) {
              fwrite(&prime_cache[j][s][i],sizeof(prime_cache[j][s][i]),
1,fp);
            }
          }
        }
      }
      globus_gass_fclose(fp);
      /* Sende jedem Prozess eine neue Zahl */
      for (i=1;i<total;i++) {
        for (s=1;s<=schrittweite;s++) {
          for (j=0;j<MAX_BIN_LONG;j++) {
            MPI_Send(&number[0][j], 1, MPI_LONG, i, tag,
MPI_COMM_WORLD);
          }
          j=add_binnumber(number[0],binnumber1,number[0]);
          if (get_bit(number[0],0,0)==0) {
            j=add_binnumber(number[0],binnumber1,number[0]);
          }
        }
      }
    }

    /* Ende der Hauptschleife */
    /* Warte auf die letzten <total*schrittweite> Ergebnisse */
    for (i=1;i<total;i++) {
      for (s=1;s<=schrittweite;s++) {
        if (s==1) {
          MPI_Recv(&solution[s][0], 1, MPI_LONG, MPI_ANY_SOURCE, tag,
MPI_COMM_WORLD, &status);
        }
        else {
          MPI_Recv(&solution[s][0], 1, MPI_LONG, client, tag,
MPI_COMM_WORLD, &status);
        }
        client=status.MPI_SOURCE;
        for (j=1;j<MAX_BIN_LONG;j++) {
          MPI_Recv(&solution[j], 1, MPI_LONG, client, tag,
MPI_COMM_WORLD, &status);
        }
        if (equal_binnumber(solution[s],binnumber0)!=0) {
          if (zahlenausgabe!=0) {
            printf("Process %d reports prime ", status.MPI_SOURCE);
            bin_to_dec(solution[s],decsolution);
            show_decnumber(decsolution);
            printf("\n");
          }
          load_binnumber(prime_cache[status.MPI_SOURCE][s],solution);
        }
        else {
          load_bin_with_dec(prime_cache[status.MPI_SOURCE][s],0);
        }
```

```
        /* Sende das Abbruchsignal an alle Prozesse */
        for (j=0;j<MAX_BIN_LONG;j++) {
          MPI_Send(&binnumber0[j], 1, MPI_LONG, status.MPI_SOURCE, tag,
MPI_COMM_WORLD);
        }
      }
    }

    /* Speichere den Primzahlencache */
    fp = globus_gass_fopen(FILE_PATH, "ab");
    for (j=1;j<total;j++) {
      for (s=1;s<=schrittweite;s++) {
        if (equal_binnumber(prime_cache[j][s],binnumber0)!=0) {
          for (i=0;i<MAX_BIN_LONG;i++) {
            fwrite(&prime_cache[j][s][i],sizeof(prime_cache[j][s][i]),
1,fp);
          }
        }
      }
    }
    globus_gass_fclose(fp);
    globus_module_deactivate(GLOBUS_GASS_FILE_MODULE);

    /* Stoppe Stoppuhr */
    if (N>0) {
      endtime=MPI_Wtime();
      printf("\nTime used for %d numbers by %d processes: %f
seconds\n",n,total,(endtime-starttime));
      printf("\nDas Siebenbis %d wurde mit %d Prozessen in %f Sekunden
durchgefuehrt!\n",n,total,(endtime-starttime));
    }

    printf("Shutting down MASTER!\n");
    printf("Beende den MASTER!\n");

  } else {

    /* CLIENT Prozess */
    while (1) {

      /* Empfange <schrittweite> Zahlen von MASTER */
      for (s=1;s<=schrittweite;s++) {
        for (j=0;j<MAX_BIN_LONG;j++) {
          MPI_Recv(&number[s][j], 1, MPI_LONG, MASTER, tag,
MPI_COMM_WORLD, &status);
        }
      }

      /* Ueberpruefe auf Abbruchsignal */
      if (equal_binnumber(number[schrittweite],binnumber0)==0) {
        break;
      }

      fp = globus_gass_fopen(FILE_PATH, "rb");
      rewind(fp);

      /* ueberspringe die Zahl 2 */
      for (i=0;i<MAX_BIN_LONG;i++) {
        j=fread(&loopnumber[i],sizeof(loopnumber[i]),1,fp);
      }
      for (i=0;i<MAX_BIN_LONG;i++) {
        j=fread(&loopnumber[i],sizeof(loopnumber[i]),1,fp);
      }
```

```
      /* Lese Primzahlen bis zur Wurzel von number[schrittweite] ein */
      sqrt_binnumber(number[schrittweite],sqrtnumber);
      add_binnumber(sqrtnumber,binnumber1,sqrtnumber);

      /* Primzahlen sieben */
      durchgesiebt=0;
      while (j!=0) {
        for (s=1;s<=schrittweite;s++) {
          if (equal_binnumber(number[s],binnumber0)==0) {
            if (mod_binnumber(number[s],loopnumber)==0) {
              load_bin_with_dec(number[s],0);
              durchgesiebt++;
            }
          }
        }
        if (durchgesiebt==schrittweite) {
          break;
        }

        for (i=0;i<MAX_BIN_LONG;i++) {
          j=fread(&loopnumber[i],sizeof(loopnumber[i]),1,fp);
        }
        if (sub_binnumber(sqrtnumber,loopnumber,tmp)!=0) {
          j=0;
        }
      }

      globus_gass_fclose(fp);

      /* Sende das Ergebnis an MASTER */
      for (s=1;s<=schrittweite;s++) {
        for (j=0;j<MAX_BIN_LONG;j++) {
          MPI_Send(&number[s][j], 1, MPI_LONG, MASTER, tag,
MPI_COMM_WORLD);
        }
      }
    }

    printf("Shutting down CLIENT %d!\n", me);
    printf("Beende CLIENT %d!\n", me);
  }
  MPI_Finalize();
  globus_module_deactivate(GLOBUS_GASS_FILE_MODULE);
}


/* Initialisiere die Primzahldatei fuer <total> Prozesse */
int generate_initial_file(int total)
{
  unsigned long init_prime=2;
  int i,k,zahl_prim;
  int max_primzahlen, sqrt_zahl;
  FILE *fp;

  fp = globus_gass_fopen(FILE_PATH, "wb");
  rewind(fp);

  /* Siebe sequentiell sqrt(total)+1 Primzahlen */
  max_primzahlen=sqrt(total)+1;
  if (max_primzahlen<3) {
     max_primzahlen=3;
  }
  fwrite(&init_prime,sizeof(init_prime),1,fp);
```

```
  for (i=1;i<MAX_BIN_LONG;i++) {
    fwrite(&binnumber0[1],sizeof(init_prime),1,fp);
  }
  if (zahlenausgabe!=0) {
    printf("Writing Prime %u \n",init_prime);
    printf("Schreibe Primzahl %u \n",init_prime);
  }
  init_prime=3;
  fwrite(&init_prime,sizeof(init_prime),1,fp);
  for (i=1;i<MAX_BIN_LONG;i++) {
    fwrite(&binnumber0[1],sizeof(init_prime),1,fp);
  }
  if (zahlenausgabe!=0) {
    printf("Writing Prime %u \n",init_prime);
    printf("Schreibe Primzahl %u \n",init_prime);
  }

  for (init_prime=4;j<=max_primzahlen;j++) {
    zahl_prim=1;
    sqrt_zahl=sqrt(init_prime);
    for (k=2;<=sqrt_zahl;k++) {
      if (j%k==0) {
        zahl_prim=0;
        break;
      }
    }
    if(zahl_prim==1) {
      fwrite(&init_prime,sizeof(init_prime),1,fp);
      for (i=1;i<MAX_BIN_LONG;i++) {
        fwrite(&binnumber0[1],sizeof(init_prime),1,fp);
      }
      if (zahlenausgabe!=0) {
        printf("Writing Prime %u \n",init_prime);
        printf("Schreibe Primzahl %u \n",init_prime);
      }
    }
  }

  globus_gass_fclose (fp);

  /* gebe naechste ungerade Zahl zurueck */
  if (init_prime%2==1) {
    return ((int)init_prime+2);
  } else {
    return ((int)init_prime+1);
  }
}


/* die nachfolgenden Funktionen sind in Kapitel 6 beschrieben*/
load_bin_with_dec(unsigned long bnumber[], unsigned long decimal)
{
  int i;
  bnumber[0]=decimal;
  for (i=1;i<MAX_BIN_LONG;i++) {
    bnumber[i]=0;
  }
}

show_binnumber(unsigned long bnumber[])
{
  int i;
  for (i=0;i<MAX_BIN_LONG;i++) {
```

```
    printf("binnumber[%d]: %u\n", i, bnumber[i]);
  }
  printf("\n");
}

show_binnumber_bin(unsigned long bnumber[])
{
  int i,j;
  for (i=0;i<MAX_BIN_LONG;i++) {
    printf("binnumber[%d]: ", i);
    for (j=31;j>=0;j--) {
      printf("%d",get_bit(bnumber,i,j));
    }
    printf("\n");
  }
  printf("\n");
}

show_decnumber(unsigned dnumber[])
{
  int i,j=0;
  for (i=(MAX_DEC_INT-1);i>=0;i--) {
    if (dnumber[i]>0 || j!=0) {
      printf("%d", dnumber[i]);
      if (dnumber[i]>0) {
        j=1;
      }
    }
  }
}

load_binnumber(unsigned long bnumber1[], unsigned long bnumber2[])
{
  int i;
  for (i=0;i<MAX_BIN_LONG;i++) {
    bnumber1[i]=bnumber2[i];
  }
}

int add_binnumber(unsigned long bnumber1[], unsigned long bnumber2[],
unsigned long solution[])
{
  int i;
  int carry=0;
  int newcarry;
  unsigned long tmp1[MAX_BIN_LONG];
  unsigned long tmp2[MAX_BIN_LONG];
  load_binnumber(tmp1,bnumber1);
  load_binnumber(tmp2,bnumber2);
  for (i=0;i<MAX_BIN_LONG;i++) {
    solution[i]=tmp1[i]+tmp2[i];
    newcarry=0;
    for (carry;carry>=0;carry--){
      if ((solution[i]>>31)>0) {
        newcarry++;
        solution[i]=solution[i]<<1;
        solution[i]=solution[i]>>1;
      }
      if (carry>0) {
        solution[i]++;
      }
    }
    carry=newcarry;
```

```
  }
  return(carry);
}

int sub_binnumber(unsigned long bnumber1[], unsigned long bnumber2[],
unsigned long solution[])
{
  int i;
  int carry=0;
  int newcarry;
  unsigned long tmp1[MAX_BIN_LONG];
  unsigned long tmp2[MAX_BIN_LONG];
  load_binnumber(tmp1,bnumber1);
  load_binnumber(tmp2,bnumber2);
  for (i=0;i<MAX_BIN_LONG;i++) {
    newcarry=0;
    if (tmp1[i]<tmp2[i]) {
      newcarry++;
      tmp1[i]=tmp1[i]+(unsigned long)0x80000000;
    }
    solution[i]=tmp1[i]-tmp2[i];
    if (carry>0) {
      if (solution[i]<carry) {
        newcarry++;
        solution[i]=solution[i]+(unsigned long)0x80000000;
      }
      solution[i]=solution[i]-carry;
    }
    carry=newcarry;
  }
  return(carry);
}

int mul_binnumber (unsigned long bnumber1[], unsigned long bnumber2[],
unsigned long solution[])
{
  int i=0;
  unsigned long tmp1[MAX_BIN_LONG];
  unsigned long tmp2[MAX_BIN_LONG];
  unsigned long tmp3[MAX_BIN_LONG];
  load_binnumber(tmp1,bnumber1);
  load_binnumber(tmp2,bnumber2);
  load_bin_with_dec(solution, 0);
  load_bin_with_dec(tmp3, 1);
  while (sub_binnumber(tmp2,tmp3,tmp2)==0 || i!=0)
  {
    i=add_binnumber(tmp1,solution,solution);
  }
  return(i);
}

int mod_binnumber (unsigned long bnumber1[], unsigned long bnumber2[])
{
  int i=0;
  unsigned long tmp1[MAX_BIN_LONG];
  unsigned long tmp2[MAX_BIN_LONG];
  unsigned long tmp3[MAX_BIN_LONG];
  load_binnumber(tmp1,bnumber1);
  load_binnumber(tmp2,bnumber2);
  load_bin_with_dec(tmp3, 0);
  while (i==0)
  {
    if (equal_binnumber(tmp1,tmp3)==0) {
```

```
      break;
    }
    i=sub_binnumber(tmp1,tmp2,tmp1);
  }
  return(i);
}

div_binnumber (unsigned long bnumber1[], unsigned long bnumber2[], int
times)
{
  int i,j;
  unsigned long tmp[MAX_BIN_LONG];
  load_binnumber(tmp,bnumber1);
  for (j=1;j<=times;j++)
  {
    for (i=0;i<MAX_BIN_LONG;i++)
    {
      bnumber2[i]=(tmp[i]>>1);
    }
  }
}

int get_bit (unsigned long bnumber[], int pos_integer, int pos_bit)
{
  unsigned long temp=0;
  temp=bnumber[pos_integer]<<(31-pos_bit);
  temp=temp>>31;
  return(temp);
}

bin_to_dec (unsigned long bnumber[], unsigned dnumber[])
{
  int i,j;
  unsigned long tmp[MAX_BIN_LONG];
  load_binnumber(tmp,bnumber);
  clear_decnumber(dnumber);
  while (sub_binnumber(bnumber,binnumber1,bnumber)==0) {
    inc_dec(dnumber,0);
  }
}

inc_dec (unsigned dnumber[], int number)
{
  if (dnumber[number]==9) {
    dnumber[number]=0;
    inc_dec(dnumber,(number+1));
  } else {
    dnumber[number]=dnumber[number]+1;
  }
}

int equal_binnumber(unsigned long bnumber1[], unsigned long bnumber2[])
{
  int i,j=0;
  for (i=0;i<MAX_BIN_LONG;i++) {
    if (bnumber1[i]!=bnumber2[i]) {
      j=1;
      break;
    }
  }
  return(j);
}
```

```
clear_decnumber (unsigned dnumber[])
{
  int i;
  for (i=0;i<MAX_DEC_INT;i++) {
    dnumber[i]=0;
  }
}

sqrt_binnumber(unsigned long bnumber1[], unsigned long guess[])
{
    unsigned long min[MAX_BIN_LONG];
    unsigned long max[MAX_BIN_LONG];
    unsigned long value[MAX_BIN_LONG];
    unsigned long delta[MAX_BIN_LONG];
    unsigned long tmp[MAX_BIN_LONG];
    unsigned long oldguess[MAX_BIN_LONG];

    load_bin_with_dec(oldguess,0);
    load_binnumber(value,bnumber1);
    div_binnumber(value,guess,1);
    load_binnumber(max,value);
    load_bin_with_dec(min,0);

    while (equal_binnumber(guess,oldguess)!=0)
    {
        mul_binnumber(guess,guess,delta);

        if (sub_binnumber(value,delta,tmp)!=0) {
            load_binnumber(max,guess);
        } else
        if (sub_binnumber(delta,value,tmp)!=0) {
            load_binnumber(min,guess);
        } else {
            break;
        }
        load_binnumber(oldguess,guess);
        add_binnumber(min,max,guess);
        div_binnumber(guess,guess,1);
    }
}

/* Funktion zum Auslesen eines Wertes zu einem Attribut */
/* AttGetVal (SessionID, CurrentPosition, SearchString) */
double AttGetVal(LDAP *ld, LDAPMessage *ldap_msg_obj, char
*searchstring)
{
  BerElement *ber;
  char *ldapAttr;
  char **char_attr;
  ldapAttr = ldap_first_attribute(ld, ldap_msg_obj, &ber);
  if (!strcmp(searchstring, ldapAttr)) {
        char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
        return(atof(char_attr[0]));
  }
  while ((ldapAttr = ldap_next_attribute(ld, ldap_msg_obj, ber)) !=
NULL) {
        if (!strcmp(searchstring, ldapAttr)) {
          char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
          return(atof(char_attr[0]));
        }
  }
  return(0);
}
```

```
/* Funktion zum Auslesen des Attributnamens */
/* AttGetStr (SessionID, CurrentPosition, SearchString) */
char * AttGetStr(LDAP *ld, LDAPMessage *ldap_msg_obj, char
*searchstring)
{
  BerElement *ber;
  char *ldapAttr;
  char **char_attr;

  ldapAttr = ldap_first_attribute(ld, ldap_msg_obj, &ber);
  if (!strcmp(searchstring, ldapAttr)) {
        char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
        return(* char_attr);
  }
  while ((ldapAttr = ldap_next_attribute(ld, ldap_msg_obj, ber)) !=
NULL) {
        if (!strcmp(searchstring, ldapAttr)) {
          char_attr = ldap_get_values(ld, ldap_msg_obj, ldapAttr);
          return(* char_attr);
        }
  }
  return(NULL);
}


/* Diese Suche ist sehr einfach gehalten und garantiert nicht den
   Speicher, der tatsaechlich gebraucht wird (kann sich noch waehrend
   der Laufzeit aendern) */
char * GetFileHost() {
  char *hostName;
  char *besthost;
  int freespace;
  int result, count, i;
  char *myLdapServer;
  char *myLdapBase;
  char *mySite;
  int myLdapScope;
  char myLdapFilter[LDAP_FILTER_NAME_MAX];
  char **myLdapAttr = NULL;
  int myLdapPort = 2135;
  char *char_dn;
  LDAP *ld;
  LDAPMessage *res;
  LDAPMessage *ldap_msg;
  char *strptr;
  double val;

  /* Hier die GIIS Daten eintragen */
  myLdapServer = (char *)strdup("kafka.informatik.uni-giessen.de");
  myLdapBase = (char *)strdup("mds-vo-name=site, o=grid");
  mySite = (char *)strdup("Mds-Host-hn=*");

  myLdapScope = LDAP_SCOPE_SUBTREE;

  if ((ld = ldap_init(myLdapServer, myLdapPort)) == NULL) {
        return("NULL");
  }

  result = ldap_simple_bind_s(ld, "", "");
  if(result != LDAP_SUCCESS) {
        return("NULL");
  }
```

```
  /* Setze Filter */
  sprintf(myLdapFilter, "%s", mySite);

  /* Suche starten */
  result=ldap_search_s(ld, myLdapBase, myLdapScope, myLdapFilter,
myLdapAttr, 0, &res);

  if(result != LDAP_SUCCESS) {
        return("NULL");
  }
  /* Anzahl der gefundenen Eintraege */
  count = ldap_count_entries(ld, res);
  if(count < 1) {
        return("NULL");
  }

  /* Auswertung */
  /* Erster Eintrag */
  i=1;
  ldap_msg = ldap_first_entry(ld, res);
  char_dn = ldap_get_dn(ld, ldap_msg);
  hostName = AttGetStr(ld, ldap_msg, "Mds-Host-hn");
  sprintf(besthost,"%s",hostName);
  val = AttGetVal(ld, ldap_msg, "Mds-Fs-Total-freeMB");
  freespace=(int)val;

  /* Restliche Eintraege */
  while (i<count) {
    i++;
    ldap_msg = ldap_next_entry(ld, ldap_msg);
    char_dn = ldap_get_dn(ld, ldap_msg);
    hostName = AttGetStr(ld, ldap_msg, "Mds-Host-hn");
    val = AttGetVal(ld, ldap_msg, "Mds-Fs-Total-freeMB");
    if ((int)val>freespace) {
      sprintf(besthost,"%s",hostName);
      freespace=(int)val;
    }
  }
  return(besthost);
}

/* einfacher Zeichenkettenvergleich */
int vergleiche(zeichenkette1, zeichenkette2)
char *zeichenkette1, *zeichenkette2;
{
  while (*zeichenkette1) {
    if (*zeichenkette1++ != zeichenkette2++){
      return(0);
    }
  }
  return(1);
}

/* einfache Zeichenkettenkopie */
/* kopiere zeichenkette2 nach zeichenkette1 */
void kopiere (zeichenkette1, zeichenkette2)
char *zeichenkette1, zeichenkette2;
{
  while (*zeichenkette1++ = zeichenkette2++);
}
```

Quellenangaben

[1] *A method for obtaining digital signatures and public key cryptosystems.*
Von R. L. Rivest, A. Shamir und L. Adleman,
In: Communications of the ACM, Band 21, 1978.

[2] *An Introduction to Parallel Algorithms.*
Joseph Jaja, Addison-Wesley, 1992.

[3] *Avaki Data Grid Architecture.*
Internet, www.avaki.com/products/adgarchitecture.html,
Stand vom 23.9.2003.

[4] *Avaki Data Grid Data Sheet.*
Internet, www.avaki.com/file/pdf/public/adg40_data_sheet.html,
Stand vom 23.9.2003.

[5] *Avaki Data Grid Features.*
Internet, www.avaki.com/products/adgfeatures.html, Stand vom 23.9.2003.

[6] *Avaki Data Grid Overview.*
Internet, www.avaki.com/products/adgoverview.html, Stand vom 23.9.2003.

[7] *Avaki Data Grid Whitepaper.* Internet,
www.avaki.com/papers/register.html?rdr_to=/file/pdf/protected/adgwp.html,
Stand vom 23.9.2003.

[8] *Avaki Homepage.*
Internet, www.avaki.com, Stand vom 22.6.2005.

[9] *Butterfly Grid - Aufbau des Grid.*
Internet, www.butterfly.net/platform/technology/griddiagram.html,
Stand vom 23.9.2003.

[10] *Butterfly Grid - Komponenten.* Internet,
www.butterfly.net/platform/technology/components.html, Stand vom 23.9.2003.

[11] *Butterfly Grid - Vernetzung.*
Internet, www.butterfly.net/platform/technology/mesh.html, Stand vom 23.9.2003.

[12] *Butterfly Grid - Übersicht.*
Internet, www.butterfly.net/platform/index.html, Stand vom 23.9.2003.

[13] *Butterfly Grid Homepage.*
Internet, www.butterfly.net, Stand vom 23.9.2003.

[14] *Butterfly.net Uses Intel Architecture to Build a Global Gaming Grid.*
Intel Business Center Case Study, Februar 2003.

[15] *Butterfly.net: Grid soll Millionen Online-Spieler verbinden.*
Artikel bei www.golem.de, 10.05.2002.

[16] *Butterfly.net: Powering Next-Generation Gaming with Computing On-Demand.*
IDC e-business Case Study, 2002.

[17] *CASA Gigabit Testbed - Benutzte Supercomputer.*
Internet, www.nlanr.net/CASA/family.gif, Stand vom 27.06.2005.

[18] *CASA Gigabit Testbed - Ergebnisse.*
Internet, www.nlanr.net/CASA/results.html, Stand vom 27.06.2005.

[19] *CASA Gigabit Testbed - Technologie.*
Internet, www.nlanr.net/CASA/technology.html, Stand vom 27.06.2005.

[20] *CASA Gigabit Testbed Homepage.*
Internet, www.nlanr.net/CASA/, Stand vom 27.06.2005.

[21] *Characterizing Grids: Attributes, Definitions, and Formalisms.*
Von Zsolt Nemeth und Vaidy Sunderam,
Journal of Grid Computing 1, Kluwer Academic Publishers, 23.9.2003.

[22] *Common High-Level Interface to Message Passing (CHIMP).*
Edinburgh Parallel Computing Center,
Internet, www.epcc.ed.ac.uk/industry/CHIMP/index.html, Stand vom 4.7.2005.

[23] *Computational Grids.* In: The Grid: Blueprint for a Future Computing Infrastructure. Von Ian Foster und Carl Kesselman,
Morgan Kaufmann Publishers, 1998.

[24] *CrossGrid - Projektübersicht.*
Internet, www.eu-crossgrid.org/project.htm, Stand vom 23.9.2003.

[25] *CrossGrid Homepage.*
Internet, www.crossgrid.org, Stand vom 27.06.2005.

[26] *DATA GRID - Angeschlossene Netzwerke.* Internet,
web.datagrid.cnr.it/servlet/page?_pageid=873,877&_dad=portal30&_schema=PORTAL30&_mode=3, Stand vom 27.06.2005.

[27] *DATA GRID - Einführung.*
Internet, web.datagrid.cnr.it/LearnMore/index.jsp, Stand vom 27.06.2005.

[28] *DATA GRID - Verwendete Middleware.* Internet, web.datagrid.cnr.it/servlet/page?_pageid=873,881&_dad=portal30&_schema=PORTAL30&_mode=3, Stand vom 27.06.2005.

[29] *DATA GRID - What is Grid Middleware?.* Internet, web.datagrid.cnr.it/LearnMore/LearnMore6.jsp, Stand vom 22.6.2005.

[30] *DATA GRID - Ziele.* Internet, web.datagrid.cnr.it/servlet/page?_pageid=873,883&_dad=portal30&_schema=PORTAL30&_mode=3, Stand vom 27.06.2005.

[31] *DATA GRID - Überblick.* Internet, web.datagrid.cnr.it/servlet/page?_pageid=873,875&_dad=portal30&_schema=PORTAL30&_mode=3, Stand vom 27.06.2005.

[32] *DATA GRID Homepage.* Internet, web.datagrid.cnr.it, Stand vom 27.06.2005.

[33] *DataTAG Trans Atlantic Grid - Übersicht.* Internet, datatag.web.cern.ch/datatag/project.html, Stand vom 23.9.2003.

[34] *DataTAG Trans Atlantic Grid Homepage.* Internet, datatag.web.cern.ch/datatag/index.html, Stand vom 27.06.2005.

[35] *Der Miller-Rabin Primzahlentest.* Von Florian Rienhardt, 31.5.2005

[36] *Die Eine-Million-Dollar-Frage: Neues Interesse an der Goldbach'schen Vermutung.* Ausgabe der Süddeutschen Zeitung vom 16.05.2000.

[37] *DOE Science Grid - Ressourcen.* Internet, doesciencegrid.org/Grid/resources.html, Stand vom 23.9.2003.

[38] *DOE Science Grid homepage.* Internet, doesciencegrid.org, Stand vom 27.06.2005.

[39] *Einführung in die Kryptographie.* Johannes Buchmann, 2. erweiterte Auflage 2001, Springer-Verlag.

[40] *Entropia - What is PC Grid Computing.* Internet, www.entropia.com/what_is_pc_grid.asp, Stand vom 23.9.2003.

[41] *Entropia - Why choose PC Grid Computing.* Internet, www.entropia.com/why_choose_pc_grid.asp, Stand vom 23.9.2003.

[42] *Entropia DCGrid 5.1 - Technische Übersicht von 2003.* http://www.entropia.com/pdf/dcgrid_data_sheet.pdf, Stand vom 23.9.2003.

[43] *Entropia DCGrid Platform.*
Internet, www.entropia.com/dcgrid_platform.asp, Stand vom 23.9.2003.

[44] *Entropia Desktop Grid Computing Platform - Produktbroschüre 2002.*
http://www.entropia.com/pdf/dcgridBrochure0402.pdf, Stand vom 23.9.2003.

[45] *Entropia Homepage.* Internet, www.entropia.com, Stand vom 23.9.2003.

[46] *eXeGrid Homepage.* Internet, www.exegrid.net, Stand vom 27.06.2005.

[47] *Experimental Analysis of Algorithms.*
Von Catherine C. McGeoch, Notices of the AMS, Volume 48, Number 3, 2001.

[48] *File and Object Replication in Data Grids.* Von Heinz Stockinger, Asad Samar, Bill Allcock, Ian Foster, Koen Holtman und Brian Tierney,
10. International Symposium on High Performance Distributed Computing, IEEE Press, 2001.

[49] *Fraunhofer Resource Grid Homepage.*
Internet, www.fhrg.frg.de/index_en.html, Stand vom 3.7.2005.

[50] *Global Grid Forum.*
Internet, www.gridforum.org bzw. www.ggf.org, Stand vom 4.7.2005.

[51] *Globus Toolkit 2.0 Packaging Technology.*
Folienübersicht, veröffentlicht bei www.globus.org, 9.8.2001.

[52] *Globus Toolkit 2.2 Technology Brief.*
Veröffentlicht bei www.globus.org, Stand Draft 4 vom 30. Januar 2003.

[53] *Globus Toolkit 2.4: The GridFTP Protocol and Software.*
Internet, www.globus.org/toolkit/docs/2.4/datagrid/gridftp.html,
Stand vom 24.7.2005.

[54] *Globus Toolkit 2.4: The Dynamically-Updated Request Online Coallocator.*
Internet, www.globus.org/toolkit/docs/2.4/duroc/, Stand vom 24.7.2005.

[55] *Globus Toolkit 2.4: Global Access to Secondary Storage.*
Internet, www.globus.org/toolkit/docs/2.4/gass/, Stand vom 24.7.2005.

[56] *Globus Toolkit 2.4: Resource Management.*
Internet, www.globus.org/toolkit/docs/2.4/gram/, Stand vom 24.7.2005.

[57] *Globus Toolkit 2.4: Grid Security Infrastructure (GSI) v2 Notes.*
Internet, www.globus.org/toolkit/docs/2.4/gsi/, Stand vom 24.7.2005.

[58] *Globus Toolkit 2.4: MDS 2.4 in the Globus Toolkit 2.4 Release.*
Internet, www.globus.org/toolkit/docs/2.4/mds/, Stand vom 24.7.2005.

[59] *Globus Toolkit 2.4: Resource Specification Language.*
Internet, www-fp.globus.org/gram/rsl_spec1.html, Stand vom 24.7.2005.

[60] *Globus Toolkit 3 Core - A Grid Service Container Framework.*
Von Thomas Sandholm und Jarek Gawor, 2.7.2003.

[61] *Globus Toolkit 3.2: Documentation.*
Internet, www.globus.org/toolkit/docs/3.2/, Stand vom 24.7.2005.

[62] *Globus Toolkit 4.0: A Primer - Discribing Globus Toolkit Version 4 (An Early and Incomplete Draft v0.6).*
Von Ian Foster, Verfügbar unter www.globus.org, 8.5.2005.

[63] *Globus Toolkit 4.0: Security.*
Internet, www.globus.org/toolkit/docs/4.0/security/, Stand vom 24.7.2005.

[64] *Globus Toolkit 4.0: Execution Management.*
Internet, www.globus.org/toolkit/docs/4.0/execution/, Stand vom 24.7.2005.

[65] *Globus Toolkit 4.0: Information Services (MDS) - Key Concepts.*
Internet, www.globus.org/toolkit/docs/4.0/info/key/, Stand vom 24.7.2005.

[66] *Globus Toolkit 4.0: Data Management.*
Internet, www.globus.org/toolkit/docs/4.0/data/ , Stand vom 24.7.2005.

[67] *Globus Toolkit 4.0: Data Replication Service.*
Internet, www.globus.org/toolkit/docs/4.0/techpreview/datarep/,
Stand vom 24.7.2005.

[68] *Globus Toolkit 4.0: OGSA Data Integration (OGSA-DAI).*
Internet, www.globus.org/toolkit/docs/4.0/techpreview/ogsadai/,
Stand vom 24.7.2005.

[69] *Globus: A Metacomputing Infrastructure Toolkit.*
Von Ian Foster und Karl Kesselman, 1998.

[70] *GMP GNU Multiple Precision Arithmetic Library.*
Internet, www.swox.com/gmp/, Stand vom 14.7.2005.

[71] *Goldbachsche Vermutung.* Aus der freien Enzyklopädie Wikipedia,
Internet, de.wikipedia.org/wiki/Goldbachsche_Vermutung, 27.11.2004.

[72] *Grid Information Services for Distributed Resource Sharing.*
Von Karl Czajkowski, Steven Fitzgerald, Ian Foster und Carl Kesselman
veröffentlicht in: Proc. 10th IEEE International Symposium on High-Performance Distributed Computing, IEEE Press, 2001.

[73] *Grid on Tap Homepage.* Internet, www.gridontap.com, Stand vom 23.9.2003.

[74] *Grid Packaging Toolkit (GPT).* Internet, www.ncsa.uiuc.edu/Divisions/ACES/GPT, 2.7.2003.

[75] *Grid Physics Network - Einleitung.* Internet, www.griphyn.org/projinfo/intro, Stand vom 27.06.2005.

[76] *Grid Physics Network - Physikalische Projekte.* Internet, www.griphyn.org/projinfo/physics, Stand vom 27.06.2005.

[77] *Grid Physics Network - Übersicht der Netzwerkteilnehmer.* Internet, www.phys.utb.edu/~ariel/PseudoWeb/web/Gridweb/info/places.php, Stand vom 23.9.2003

[78] *Grid Physics Network - Übersicht.* Internet, www.griphyn.org/projinfo/summary.php, Stand vom 27.06.2005.

[79] *Grid Physics Network Homepage.* Internet, www.griphyn.org, Stand vom 27.06.2005.

[80] *Grid Service Specification.* Von Steven Tuecke, Karl Czajkowski, Ian Foster, Jeffrey Frey, Steve Graham und Carl Kesselman, Draft 3 vom 17.7.2002.

[81] *Grid Services for Distributed System Integration.* Von Ian Foster, Carl Kesselman, Jeffrey M. Nick und Steven Tuecke, IEEE Press, Zeitschriftartikel vom Juni 2002.

[82] *GridFTP: Universal Data Transfer for the Grid.* Veröffentlicht bei www.globus.org, 5.7.2000.

[83] *HARNESS: A Next Generation Distributed Virtual Machine.* Von Micah Beck, Jack J. Dongarra, Graham E. Fagg, G. Al Geist, Paul Gray, James Kohl, MauroMigliardi, Keith Moore, Terry Moore, Philip Papadopoulous, Stephen L. Scott und Vaidy Sunderam, 24.7.1998.

[84] *I-Lab Projekt Homepage.* Internet, www.fhrg.frg.de/english/Main/ilab/ilab.html, Stand vom 3.7.2005.

[85] *I-WAY Homepage.* Internet, www.iway.org, Stand vom 23.9.2003.

[86] *Ibis: An Efficient Java-based Grid Programming Environment.* Von Rob V. van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann und Henri E. Bal, Veröffentlicht zur JGI 2002.

[87] *IBM - The Smallpox Research Grid - Übersicht.* Internet, www-3.ibm.com/solutions/lifesciences/research, Stand vom 27.06.2005.

[88] *IBM and grid.* Internet,
www-1.ibm.com/grid/about_grid/ibm_grid/index.shtml, Stand vom 23.9.2003.

[89] *IBM bietet Supercomputing on demand als Mietleistung.*
Artikel bei www.golem.de, 09.01.2003.

[90] *IBM eröffnet erstes Grid Innovation Center in Montpellier.*
Artikel bei www.golem.de, 24.04.2002.

[91] *IBM Grid computing.*
Internet, www-1.ibm.com/grid/index.shtml, Stand vom 22.6.2005.

[92] *IBM Grid Toolbox.*
Internet, www-1.ibm.com/grid/solutions/grid_toolbox.shtml,
Stand vom 23.9.2003.

[93] *IBM Homepage.* Internet, www.ibm.com, Stand vom 22.6.2005.

[94] *IBM und Cisco wollen Grid-Computing vorantreiben.*
Artikel bei www.golem.de, 29.04.2003.

[95] *IBM: e-business on demand.* Internet,
www-1.ibm.com/grid/about_grid/ibm_grid/ebod.shtml, Stand vom 22.6.2005.

[96] *IBM: Grid-Computing-Lösungen für die Wirtschaft.*
Artikel bei www.golem.de, 27.01.2003.

[97] *IBM: What is grid computing.*
Internet, www-1.ibm.com/grid/about_grid/what_is.shtml, Stand vom 22.6.2005.

[98] *In Search of Clusters.* Von Gregory Pfister, Prentice Hall PTR, 2. Auflage 1997.

[99] *Information Power Grid - Eigenschaften.*
Internet, www.nas.nasa.gov/About/IPG/capabilities.html, Stand vom 23.9.2003.

[100] *Information Power Grid - Übersicht.*
Internet, www.nas.nasa.gov/About/IPG/aboutipg.html, Stand vom 23.9.2003.

[101] *Information Power Grid Homepage.*
Internet, www.nas.nasa.gov/About/IPG/ipg.html, Stand vom 23.9.2003.

[102] *Integration with Grid Security - PKI-based access control for grid computing.*
Internet, www.cs.wisc.edu/pkilab/condor/index.html, Stand vom 22.6.2005.

[103] *International Virtual Data Grid Laboratory - Übersicht.*
Internet, www.ivdgl.org/projinfo, Stand vom 27.06.2005.

[104] *International Virtual Data Grid Laboratory Homepage.*
Internet, www.ivdgl.org, Stand vom 27.06.2005.

[105] *Introductinon to Grid Computing with Globus.* Von Luis Ferreira, Viktors Berstis, Jonathan Armstrong, Mike Kendzierski, Anderas Neukoetter, Masanobu Takagi, Richard Bing-Wo, Adeeb Amir, Ryo Murakawa, Olegario Hernandez, James Magowan und Norbert Bieberstein, Dezember 2002.

[106] *Legion Applications.*
Internet, legion.virginia.edu/overview_apps.htm, Stand vom 22.6.2005.

[107] *Legion Architecture.*
Internet, legion.virginia.edu/overview_arch.htm, Stand vom 22.6.2005.

[108] *Legion High Performance.*
Internet, legion.virginia.edu/overview_high.htm, Stand vom 22.6.2005.

[109] *Legion Objectives and Constraints.*
Internet, legion.virginia.edu/overview_obj.html, Stand vom 22.6.2005.

[110] *Legion Overview.*
Internet, legion.virginia.edu/overview.htm, Stand vom 22.6.2005.

[111] *Legion Scheduling.*
Internet, legion.virginia.edu/overview_sched.html, Stand vom 22.6.2005.

[112] *Legion Security.*
Internet, legion.virginia.edu/overview_security.html, Stand vom 22.6.2005.

[113] *Legion technical papers.*
Internet, legion.virginia.edu/papers.html, Stand vom 22.6.2005.

[114] *Legion.* Internet, legion.virginia.edu/index.html, Stand vom 22.6.2005.

[115] *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names.*
Internet, www.rfc-editor.org/rfc/rfc2253.txt, Stand vom 4.7.2005.

[116] *Local Area Multicomputer (LAM).* Vom Ohio Supercomputing Center.
Internet, www.mpi.nd.edu/lam, Stand vom 4.7.2005.

[117] *MDS 2.2 User's Guide.* Veröffentlicht bei www.globus.org, Stand vom 10.3.2003.

[118] *Miller-Rabin-Test.* Aus der freien Enzyklopädie Wikipedia,
http://de.wikipedia.org/wiki/Miller-Rabin-Test, 10.7.2005.

[119] *Motivating Computational Grids*.
Von D.B. Sillicorn, External Technical Report, November 2001.

[120] *MPI Forum.* Internet, www.mpi-forum.org, Stand vom 4.7.2005.

[121] *MPICH.* Vom Argonne National Lab und der Mississipi State University, Internet, www-unix.mcs.anl.gov/mpi/mpich, Stand vom 4.7.2005.

[122] *NASA Advanced Supercomputing Homepage.* Internet, www.nas.nasa.gov/index.html, Stand vom 27.06.2005.

[123] *NASA Advanced Supercomputing Research - Forschung an Grids.* Internet, www.nas.nasa.gov/Research/Tasks/gridtasks.html, Stand vom 27.06.2005.

[124] *NASA Advanced Supercomputing Research - Forschungsprojekte.* Internet, www.nas.nasa.gov/Research/Tasks/tasks.html, Stand vom 27.06.2005.

[125] *Open Grid Service Infrastructure (OGSI) 1.0.* Editiert von S. Tuecke, K. Czajkowski, Foster, J. Frey, S. Graham, C.Kesselmann, T. Maquire, T. Sandholm, D. Snellin und P.Vanderbilt, 27.6.2003.

[126] *Overview of Condor.* Internet, www.cs.wisc.edu/condor/overview/, Stand vom 22.6.2005.

[127] *Parallel and Distributed Computing.* Von Claudia Leopold; John Wiley and Sons Inc., 1. Auflage 2001.

[128] *Parallel Virtual Machine (PVM).* Internet, www.netlib.org/pvm3/book/pvm-book.html, Stand vom 4.7.2005.

[129] *Parallele Algorithmen.* Von P. Damaschke, Skript, FernUniversität Hagen, 1996.

[130] *Parallele und verteilte Programmierung.* Von Thomas Rauber und Gundula Rünger, Springer Verlag, 1. Auflage 2000.

[131] *Particle Physics Data Grid - Grids und Software.* Internet, www.ppdg.net/experiment_grids.htm, Stand vom 23.9.2003.

[132] *Particle Physics Data Grid - Projekte.* Internet, www.ppdg.net/pa/ppdg-pa/projects.htm, Stand vom 23.9.2003.

[133] *Particle Physics Data Grid - Teilnehmer.* Internet, www.ppdg.net/list_of_participants.htm, Stand vom 23.9.2003.

[134] *Particle Physics Data Grid Homepage.* Internet, www.ppdg.net, Stand vom 27.06.2005.

[135] *Performance Prediction in a Grid Environment.* Von Rosa M. Badia, Francesc Escale, Edgar Gabriel, Judit Gimenez, Rainer Keller, Jesus Labarta und Matthias S. Müller, DAMIEN Project im Internet, 2003.

[136] *Phoenix: A Parallel Programming Modell for Accomodating Dynamically Joining/Leaving Resources.* Von Kenjiro Taura, Kenji Kaneda, Toshio Endo und Akinori Yonezawa, veröffentlicht zur PPPoPP 2003.

[137] *Platform - The Evolution of the grid.*
Internet, www.platform.com/grid/evolution.asp, Stand vom 22.6.2005.

[138] *Platform Computing Website.* Internet, www.platform.com, Stand vom 22.6.2005.

[139] *Platform Globus Toolkit.*
Internet, www.platform.com/products/Globus/, Stand vom 22.6.2005.

[140] *Platform Grid Resource Center.*
Internet, www.platform.com/grid/index.asp, Stand vom 22.6.2005.

[141] *Platform Standards and OGSA.*
Internet, www.platform.com/resources/standards_ogsa/, Stand vom 22.6.2005.

[142] *Platform teams with IBM to provide Grid Solutions for worldwide Grid Innovation Center.* Artikel bei www.hoise.com, 23.4.2002.

[143] *Platform's Grid Solution Stack.*
Internet, www.platform.com/grid/gridstack.asp, Stand vom 23.9.2003.

[144] *Prime Numbers and Computer Methods for Factorization.*
Von Hans Riesel, 1. Auflage 1985, Birkäuser Boston.

[145] *Primzahlen und Parallelrechner.* Von Martin Kutrib und Jörg Richstein, Spiegel der Forschung, 12. Jahrgang, Nummer 2, November 1995.

[146] *Primzahlen,* Von Ernst Trost, 1. Auflage 1953 bei Verlag Birkhäuser AG, Basel.

[147] *Quadratisches Siebverfahren.*
Von Bernhard Helmes, Internet, www.devalco.de/liste_x%5E2+x+1.htm, Stand vom 14.7.2005.

[148] *Riding the WAVE: I-WAY in Real Time at SC'95.*
Holly Korab Artikel in Access, Ausgabe Januar 1996. Auch verfügbar unter: www.ncsa.uiuc.edu/News/Access/Archive/backissues/96.1/iway.sc95.html Stand vom 27.06.2005.

[149] *SETI@home: Massively Distributed Computing for SETI.*
Von E. Korpela, Werthimer, D. Anderson, J. Cobb und M. Lebofsky Computing in Science and Engineering, No. 1, 2001.

[150] *Software Infrastructure of the I-WAY High-Performance Distributed Computing Experiments.*
Von Ian Foster, Jonathan Geisler, Bill Nickless, Warren Smith und Steven Tuecke.

[151] *Sony setzt auf Butterfly-Grid für Online-Spiele.*
Artikel bei www.golem.de, 27.02.2003.

[152] *Sun Grid Engine 5.3.*
Internet, www.sun.com/software/gridware/5.3/index.xml, Stand vom 22.6.2005.

[153] *Sun ONE Grid Engine Software.*
Internet, wwws.sun.com/software/gridware/index.html, Stand vom 23.9.2003.

[154] *Taschenbuch der Mathematik.*
Von I.N. Bronstein, K.A. Semidjajew, G. Musiol und H. Mühling,
Verlag Harri Deutsch, 4. Auflage der Neubearbeitung, 1999.

[155] *TeraGrid Homepage.* Internet, www.teragrid.org, Stand vom 27.06.2005.

[156] *TeraGrid Übersicht.*
Internet, www.teragrid.org/about/index.html, Stand vom 23.9.2003.

[157] *The Anatomy of the Grid - Enabling Scalable Virtual Organizaions.*
Von Ian Foster, Carl Kesselman und Steven Tuecke
International Journal of Supercomputer Applications, Vol. 15, No. 3, 2001.

[158] *The Condor Project Homepage.*
Internet, www.cs.wisc.edu/condor/, Stand vom 22.6.2005.

[159] *The CondorG Homepage.*
Internet, www.cs.wisc.edu/condor/condorg/, Stand vom 22.6.2005.

[160] *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets.* Von Ann Chervenak, Ian Foster,
Carl Kesselman, Charles Salisbury and Steven Tuecke, 1999.

[161] *The DOE Science Grid.* Herausgegeben von William E. Johnston,
Office of Science - US Department of Energy, 2003.

[162] *The Globus Alliance Homepage.*
Internet, www.globus.org, Stand vom15.7.2005.

[163] *The Globus Data Grid Effort.*
Internet, www.globus.org/toolkit/docs/2.4/datagrid/, Stand vom 14.7.2005.

[164] *The Globus Project: A Status Report.*
Von Ian Foster und Carl Kesselman, Globus Paper bei www.globus.org, 1998.

[165] *The Grid: Blueprint for a New Computing Infrastructure.*
Von Ian Foster und Carl Kesselman (Editors),
Morgan Kaufmann Publishers, 1999.

[166] *The I-WAY and the Future of Distributed Supercomputing.* Artikel in Parallel Computing Research, www.softlib.rice.edu/CRPC/newsletters/fal95/index.html, Stand vom 23.9.2003.

[167] *The I-WAY Network - Überersichtsartikel.* Internet, www.anl.gov/OPA/frontiers96/iway.htm, Stand vom 23.9.2003.

[168] *The Legion Grid Portal.* Von Anand Natrajan, Anh Nguyen-Tuong, Marty A. Humphrey, Andrew S. Grimshaw, Internet, 30.6.2001.

[169] *The MPICH-G2 Build and Installation Instructions.* Internet, www3.niu.edu/mpi/, Stand vom 23.9.2003.

[170] *The OpenLDAP Project Homepage.* Internet, www.openldap.org, Stand vom 22.6.2005.

[171] *The Physiology of the Grid.* Von I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, Draft Version from 22.6.2002.

[172] *The TeraGrid: A Primer.* Von Charlie Catlett, September 2002.

[173] *UNICORE Forum.* Internet, www.unicore.org/forum.htm, Stand vom 22.6.2005.

[174] *UNICORE.* Internet, www.unicore.sourceforge.net, Stand vom 22.6.2005.

[175] *Verifying the Goldbach Conjecture up to 4*10^14.* Von: Jörg Richstein, Mathematics of Computation 70, 2001.

[176] *What is Condor?.* Internet, www.cs.wisc.edu/condor/description.htmStand vom 22.6.2005.

[177] *What is the Grid? A Three Point Checklist.* Von Ian Foster, Artikel bei www.gridtoday.com, 22.7.2002.