Berechnungen mit großen Datenmengen

In gleichem Maße wie Computer immer leistungsfähiger werden, rücken neue Anwendungen ins Blickfeld, die immer größere Anforderungen stellen. Hier soll es vor allem um Probleme gehen, die mit sehr großen Datenmengen umgehen:

In *Data Warehouses* tragen große Firmen sämtliche ihrer Geschäftsvorgänge zusammen. Will nun eine bundesweit vertretene Supermarktkette herausfinden wie hoch ihr Absatz von Lyonerwürstchen im Saarland nach 18:00 im Spätherbst ist, müssen riesige Datenmengen durchsucht werden.

Geografische Informationssysteme ersetzen zunehmend die traditionellen Landkarten. Die digitale Speicherung erlaubt eine flexible Aufbereitung vielfältiger Informationen und eine direkte Weiterverarbeitung geografischer Daten. Fast jeder kennt elektronische Routenplaner, die einen großen Teil des deutschen Straßennetzes auf einer CD unterbringen. Aber wie findet so ein Gerät die schnellste Route zwischen zwei Adressen ohne minutenlanges Rattern des CD-Laufwerks. Was, wenn Wissenschaftler anhand eines europaweiten Katasterplanes mögliche Routen für ein Schnellbahnnetz darauf abklopfen wollen, wieviele bebaute Grundstücke mit Lärmbelästigung zu rechnen hätten?

Inzwischen lassen sich komplette Bibliotheken oder gar ein Filmarchiv kostengünstig digital speichern. Eine solche Bibliothek sollte hunderten oder tausenden von Benutzern gleichzeitig den Zugriff auf die Bestände ermöglichen. Wie geht das?

Archive für Satellitenbilder, Klimasimulation oder Experimente der Elementarteilchenphysik rechnen schon heute mit *Petabytes* ($\approx 10^{15}$ Bytes zu je acht Ja/Nein-Informationen). Ein Petabyte Daten als Text auf DIN A4 Blätter ausdgedruckt, ergäbe Papierstapel, mit denen man die chinesische Mauer nachbauen könnte.

Für den Umgang mit großen Datenmengen genügt es nicht, bekannte Verfahren für kleine Datenmengen zu übernehmen und ansonsten auf die ständig wachsende Leistung der Computer zu setzen. Um das einzusehen, muss man überhaupt nicht über Computer reden. Betrachten wir ein (künstliches) Beispiel aus den klassischen Geisteswissenschaften: Nehmen wir an, eine Gelehrte namens Eva arbeite an einem umfassenden Aufsatz über den Europagedanken in der Lyrik des zweiten Jahrtausends, der pünktlich zum Jahrtausendwechsel fertig werden soll. Eva kann einige wenige der gerade bearbeiteten Verse im Kurzzeitgedächtnis behalten und innerhalb von Sekundenbruchteilen Querverbindungen zwischen einzelnen Wörtern ziehen. Lassen wir ihr Langzeitgedächtnis unberücksichtigt, so ist sie auf Bücher und Notizen auf ihrem Schreibtisch angewiesen, um mit den gerade bearbeiteten Gedichten zurecht zu kommen. Eva benötigt einige Sekunden, um ein aufgeschlagenes Buch zu greifen und die richtige Zeile zu finden. In ihrem Arbeitszimmer hat Eva hunderte von Büchern zusammengetragen, die einen großen Teil der benötigten Information abdecken. Das Auffinden des richtigen Buchs im Bücherschrank und das Aufschlagen der richtigen Seite kann allerdings Minuten in Anspruch nehmen. Um das auszugleichen, nimmt sie die Bücher meist aus dem Bücherregal und legt sie aufgeschlagen auf den Schreibtisch. Erst wenn die Bücher sich zu hoch stapeln, stellt sie nicht mehr benötigte Bände zurück. Um ein dringend benötigtes Buch aus der Universitätsbibliothek zu bekommen, benötigt sie eine Stunde. Deshalb kopiert sie oft ganze Kapitel aus mehreren Büchern, um die aufwendigen Bibliotheksbesuche in Grenzen zu halten. Ist das gesuchte Buch selbst dort nicht verfügbar, muß Eva wochenlang auf eine Fernleihbestellung warten. Wenn sie erst dort weitere Hinweise auf andere seltene Bücher bekommt, kann es schwer werden, den Aufsatz rechtzeitig fertigzustellen.

Computer arbeiten ungleich primitiver als Menschen. Trotzdem geht die Analogie zu unserem konstruierten Beispiel sehr weit.

Eine Abfolge von Datenspeichern mit steigender Kapazität und abnehmender Zugriffsgeschwindigkeit bezeichnet man in der Informatik als Speicherhierarchie. Erhöhte Zugriffszeiten werden durch die blockweise Übertragung nebeneinanderstehender Information teilweise ausgeglichen. Diese einfachen Prinzipien helfen, Ordnung in die Vielfalt von Speichertechnologien zu bringen, die in der Computertechnik eine Rolle spielen.

Abbildung 1 zeigt die Zugriffszeiten, Kapazitäten und typische Blockgrößen für die Speicherhierarchie eines heutigen Computers. Evas Kurzzeitgedachtnis entsprechen einige Dutzend *Register*, die je ein *Maschinenwort* speichern können, das jeweils einen von 2^{64} verschiedenen Werten speichern kann. Statt des Schreibtischs in unserem Beispiel gibt es zwei oder drei Ebenen von Zwischenspeichern (engl. Caches) – auch Eva kann auf direkt vor ihr liegende Bücher durch bloße Augenbewegung zugreifen, während sie andere Büchern auf dem Schreibtisch erst zu sich hinziehen muß. Zwischenspeicher sind in den letzten zehn Jahren immer wichtiger geworden, da die Geschwindigkeit preiswerter Halbleiterspeicher mit hoher Kapazität (DRAMs) viel langsamer zugenommen hat, als die der Prozessoren. Zwar lassen sich Speicherbausteine parallel schalten und der Abtransport nebeneinanderstehender Informationen aus einem Chip kann ebenfalls beschleunigt werden, aber diese Maßnahmen sind nur sinnvoll, wenn Speicherblöcke von 16 bis 256 Bytes gemeinsam in den Zwischenspeicher gebracht werden. Insgesamt sind Register etwa hundert mal schneller als Hauptspeicher. Dieser wiederum ist etwa zehntausend mal schneller als Magnetplatten, die mechanische Komponenten enthalten. Das ist etwa so, als arbeite Eva auf einer einsamen Insel von wo sie nur einmal pro Woche die Fähre zur Universtätsbibliothek nehmen kann. So ein Aufwand lohnt sich nur, wenn sehr viele Daten auf einmal übertragen werden. Schließlich lassen sich wirklich große Datenmengen nur mit auswechselbare Medien wie Magnetbändern oder optischen Platten (z.B. CDs) preiswert archivieren. Selbst ein roboterbetriebenes Bandarchiv benötigt allerdings Minuten für das Einlegen und Vorspulen eines Bandes.

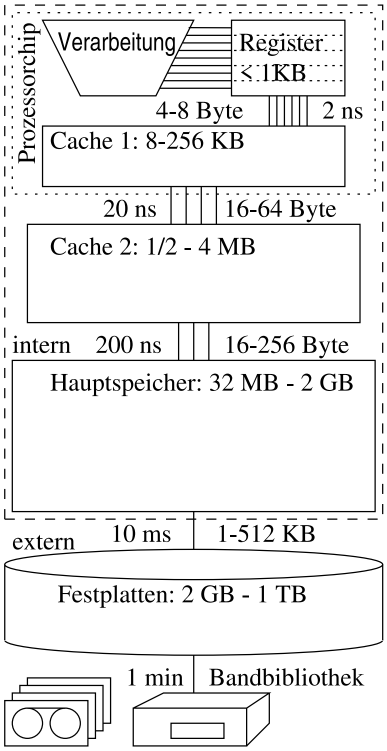

Abbildung 1: Typische Speicherhierarchie eines heutigen Computers.

Bemerkenswert ist, dass sich vom Hochleistungsrechner bis herab zum Heimcomputer Aufbau und Geschwindigkeit der Hierarchieebenenen heute kaum unterscheiden. Hochleistungsrechner haben lediglich größere Speicher, können größere Blöcke in der gleichen Zeit übertragen und bestehen oft aus vielen parallel betriebenen Berechnungseinheiten, Speichermodulen und Magnetplattenlaufwerken. Das Bild war in der Vergangenheit weniger einheitlich und mag sich auch wieder ändern – genau wie viele andere technologische Details. Das Prinzip, dass größere Speicher längere Zugriffszeiten erfordern, ist jedoch allgegenwärtig. Die Endlichkeit der Lichtgeschwindigkeit ist einer der Gründe dafür und wird sich wohl kaum überwinden lassen. Die blockweise Übertragung von Daten ist die einfachste Gegenmaßnahme und wird sich deshalb ebenfalls behaupten.

Der Forschungsschwerpunkt unserer Arbeitsgruppe ist Entwurf und Analyse effizienter Algorithmen (also Berechnungsvorschriften für Computer). Traditionell herrscht in diesem Gebiet ein Modell vor, das sich mindestens bis zu einem Aufsatz von John von Neumann aus dem Jahr 1945 ("First Draft of a Report on the EDVAC") zurückverfolgen läßt. Dieses *Von-Neumann-Modell* kennt keine Speicherhierarchie sondern nur eine Verarbeitungseinheit und einen großen Hauptspeicher. Jeder Zugriff auf eine einzelne Speicherstelle dauert gleich lange. In unserem Lyrikbeispiel würde das bedeuten, dass Eva auf Fernleihe und Ablagefläche auf dem Schreibtisch verzichtet, ihre Wohnung bis unter die Decke mit Büchern aus der Universitätsbibliothek vollstopft und ständig zwischen ihrem Schreibtisch und den Bücherregalen hin- und herspringt. Da sie dabei jeweils nur ein Wort nachschlägt wird ihr Aufsatz trotzdem nie fertig.

Die zunehmende Vielfalt von Anwendungen mit großen Datenmengen und die sich weiter öffnende Schere zwischen Prozessorgeschwindigkeit und Speicherzugriffsverzögerung hat in den letzten Jahren dazu geführt, das gesamte Spektrum algorithmischer Problemstellungen unter dem Blickwinkel von Speicherhierarchien zu betrachten. Leider ist es leichter, das Von-Neumann-Modell zu kritisieren, als eine sinnvolle Alternative anzugeben. Schon seit den fünfziger Jahren wurden ausgewählte wichtige Probleme wie Sortieren für Bandlaufwerke oder Magnetplatten in der Praxis gelöst und auch theoretisch untersucht. Detaillierte Modelle einer Technologie sind aber für den allgemeinen Gebrauch zu kompliziert und in zu großer Gefahr, von der technologischen Entwicklung überholt zu werden. Deshalb wird neben dem Von-Neumann-Modell heute vor allem auf das folgende einfache *Externspeicher-Modell* (engl. external memory) zurückgegriffen: Ein Von-Neumann-Rechner mit *begrenztem* internen Speicher für M Maschinenworte ist mit einem externen Speicher verbunden. In einer Ein/Ausgabezeiteinheit kann ein Block mit B Maschinenwörtern zwischen internem und externem Speicher bewegt werden. Wir werden Algorithmen für das Von-Neumann-Modell von nun an als *interne* Algorithmen bezeichnen. Wegen des großen Geschwindigkeitsunterschieds zwischen Hauptspeicher und Plattenspeicher spielen meist diese beiden Ebenen die Rollen von internem und externem Speicher. Zur Algorithmenanalyse werden neben dem internen Aufwand vor allem die Anzahl der Ein/Ausgabeoperationen gezählt. Manchmal ist der interne Aufwand sogar vernachlässigbar.

Im Folgenden konzentrieren wir uns auf ein Beispiel — Sortieren und verwandte Probleme — um grundlegende Techniken für externe Algorithmen einzuführen. Anschließend geben wir einen Überblick über weitere Fragestellungen und die Arbeiten unserer Gruppe dazu.

Sortieren & Co.

Zunächst ist klar, dass ein schlechter interner Algorithmus im allgemeinen auch ein schlechter externer Algorithmus ist. Der erste Schritt beim Entwurf eines externen Algorithmus ist deshalb eine Sichtung der existierenden Verfahren für das Von-Neumann-Modell. Nehmen wir als Beispiel das Sortierproblem: Gegeben N Elemente (z.B. Zahlen, Namen,...), die in beliebiger Reihenfolge hintereinander im externen Speicher stehen. Die Elemente sollen so umgeordnet werden, dass sie in aufsteigender Reihenfolge — wie im Telefonbuch — sortiert sind. Es gibt mehrere interne Sortieralgorithmen deren Ausführungszeit proportional zu $N \log N$ ist. Unter bestimmten plausiblen Annahmen kann man zeigen, dass es keine wesentlich schnelleren Sortieralgorithmen geben kann.

Der in mehrfacher Hinsicht beste dieser schnellen Algorithmen heißt *Heapsort*. Setzt man Heapsort in einen externen Algorithmus um, benötigt man jedoch sehr viele Ein/Ausgabeoperationen und müsste bei heutiger Technologie mit einer mehrtausendfachen Verlangsamung gegenüber internem Speicher rechnen. Heapsort hat das Problem, dass die Speicherstellen auf die nacheinander zugegriffen wird, nicht nebeneinander stehen. Deshalb gehen wir nicht näher auf dieses Verfahren ein und betrachten *Sortieren durch Mischen*:

Das Problem, eine Folge von N Zahlen zu sortieren wird zunächst vereinfacht, indem die Folge in zwei Hälften geteilt wird. Die Hälften werden dann getrennt sortiert. Die kleineren Folgen werden mittels *Rekursion* sortiert, also durch Anwendung des gleichen Verfahrens. Dadurch werden die Folgen solange weiter aufgeteilt bis sie so kurz sind, dass sie mittels eines einfachen Verfahrens sortiert werden können (z.B. sind Folgen der Länge eins immer sortiert). Schließlich müssen die beiden sortierten Teilfolgen nur noch zu einer einzigen sortierten Folge zusammengemischt werden. Dazu genügt es, jeweils die ersten Elemente der beiden Folgen zu vergleichen, das kleinere zu entfernen und an die Ergebnisliste anzuhängen. Mischen läßt sich sehr gut in einen externen Algorithmus umsetzen. Es genügt jeweils einen Block der Eingabelisten und der Ausgabelisten im internen Speicher zu behalten. Dann lassen sich die kleinsten Elemente auswählen und im Ausgabeblock zwischenspeichern. Ist der Ausgabeblock voll, wird er ausgegeben und im internen Speicher ist wieder Platz für die nächsten Elemente. Ist einer der Eingabeblöcke erschöpft ist, wird der nächste Block der zugehörigen Liste eingelesen. So lassen sich zwei sortierte Listen der Gesamtlänge K mit je K/B Ein- und Ausgabeoperationen zu einer einzigen sortierten Liste zusammenfügen (wenn B Elemente in einen Speicherblock passen). Im Verlauf des gesamten Sortieralgorithmus nimmt jedes Element an $\log_2 N$ Mischoperationen teil, und es ergibt sich ein Gesamtaufwand von etwa $2(N/B)\log_2 N$ Ein/Ausgabeoperationen.

Läßt sich der Entwurf externer Algorithmen also auf die Anpassung interner Vorbilder mit hoher Zugriffslokalität reduzieren? Wenn das so wäre, gäbe es hier wenig zu berichten. Interne Algorithmen mit hoher Lokalität sind jedoch nicht immer bekannt. Außerdem gibt es mindestens noch ein weiteres Entwurfsprinzip für externe Algorithmen: Ist ein Datum einmal im internen Speicher, sollte möglichst viel damit geschehen bevor es wieder ausgelagert wird. In diesem Sinne ist Sortieren durch Mischen noch nicht optimal — jedes gelesene Element wird mit nur einem anderen Element verglichen bevor es wieder geschrieben wird. Sortieren durch Mischen läßt sich jedoch weiter verfeinern: Eine zu sortierende Folge wird nicht nur in zwei sondern in k Folgen aufgespalten. Gelingt es, diese Folgen durch *k-Wege-Mischen* effizient zu einer sortierten Folge zusammenzumischen, so müssen die Datenelemente viel seltener — nämlich etwa $\log_k N$ mal angefaßt werden. Dies ist tatsächlich möglich, wenn neben dem Ausgabepuffer im internen Speicher Platz für k Eingabepuffer ist. Wir können also $k \approx M/B$ setzen. Für realistische Eingabegrößen bedeuted dies, dass die Daten nur zweimal bis viermal angefaßt werden müssen. Das ist oft zehnmal seltener als beim Zwei-Wege-Mischen. Es läßt sich außerdem zeigen, dass eine weitere deutliche Verbesserung, z.B. auf einen Faktor, der überhaupt nicht mehr von N abhängt, nicht möglich ist.

Unsere Arbeitsgruppe hat mehrere Arbeiten über eine Verallgemeinerung des Sortierproblems geschrieben: Eine *Prioritätsliste* verwaltet eine Menge von Elementen, in die man jederzeit neue Elemente einfügen sowie das jeweils kleinste Element entfernen kann. Diese Datenstruktur ist das Herzstück vieler Algorithmen, für Optimierung oder Simulation. Sortieren ergibt sich als Spezialfall, indem zu-

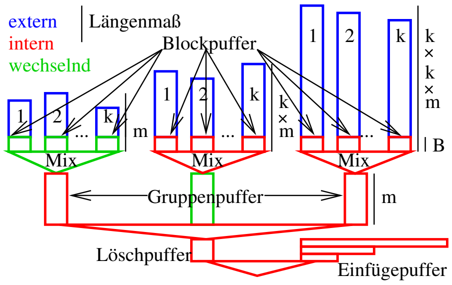

Abbildung 2: Prioritätslistendatenstruktur

nächst alle Elemente eingefügt werden und die Liste dann geleert wird. Dabei ergeben sich die ursprünglichen Elemente in sortierter Reihenfolge. Einer unserer Ansätze wird in Abbildung 2 skizziert. Die Grundidee ist, die Listenelemente in mehreren sortierten Listen zu verwalten und die kleinsten Elemente durch Mehr-Wege-Mischen auszuwählen. Die Schwierigkeit dabei ist, dass es einerseits zu teuer ist, einzelne neue Elemente in existierende Listen einzufügen, andererseits nicht beliebig viele verschiedene Listen gleichzeitig im Auge behalten werden können. Deshalb werden neue Element zunächst in einem Einfügepuffer gesammelt, der als interne Prioritätsliste organisiert ist. Damit können auch neu eingefügte Elemente beim Auffinden des kleinsten Elements berücksichtigt werden. Erst wenn der Einfügepuffer voll ist, wird er sortiert und als neue sortierte Liste in den externen Speicher geschrieben. Sammeln sich zuviele dieser Listen an, werden sie zu einer einzigen langen sortierten Liste zusammengemischt. Gibt es zu viele lange Listen, werden diese wiederum zusammengemischt etc. Dabei stellt sich heraus, dass für optimale Leistung doch etwas mehr Listen benötigt werden als man gleichzeitig zusammenmischen möchte. Deshalb werden die Listen nach ihrer Maximallänge gruppiert und für jede Gruppe ein eigener Puffer verwaltet, in dem die kleinsten Elemente der Gruppe zwischengelagert werden. Die global kleinsten Elemente werden immer aus diesen Gruppenpuffern geholt. Eine sorgfältige Analyse zeigt, dass nur ungefähr $2(N/B)\log_{M/B} N$ Ein/Ausgabeoperationen benötigt werden, um eine beliebige Folge von Einfüge- und Entferne-Minimum-Operationen mit bis zu N Einfügeoperationen durchzuführen. Die zusätzliche Flexibilität gegenüber einfachem Sortieren muß also nicht mit zusätzlichem Zeitaufwand bezahlt werden.

In vielen Anwendungen sind Prioritäten Zahlen mit begrenztem Wertebereich zwischen 1 und U und ergeben sich aus dem zuletzt entfernten Element durch Addieren eines Wertes. In diesem Fall gibt es einen besseren internen Algorithmus, der nur $\log\log U$ Operationen benötigt (*radix heaps*). Das Grundprinzip dieses Algorithmus besteht darin, die möglichen Schlüsselwerte geschickt in Intervalle einzuteilen und die Listenelemente für jeden Wertintervall getrennt abzuspeichern. Die Grundoperation ist dann einen Schlüsselwert zu kategorisieren und das Element in die richtige Teilliste zu schreiben. Dieser Ansatz – ein Eingabestrom und viele Ausgabeströme – ist komplementär zu dem beim Mehr-Wege-Mischen beobachteten Ansatz, viele Eingabeströme zu vereinigen. Beide Schemata finden sich in vielen externen Algorithmen und sind oft ein Zeichen, dass es gelungen ist, mit einem einzigen Durchlauf möglichst viel nützliche Arbeit zu verrichten.

Weitere Themen

Mindestens so viel wie über Sortieren ließe sich hier über andere fundamentale Fragestellungen sagen: Sogenannte *Graphenalgorithmen* stehen hinter fast jeder Anwendung, die etwas mit Querverweisen zwischen Daten zu tun hat. Da ein Verweis etwas inhärent nichtlokales ist, ergeben sich große Schwierigkeiten beim Entwurf externer Graphenalgorithmen. Immerhin ist es uns gelungen, das Umwandeln von Listen in Tabellen (*list ranking*) und das Bestimmen von Zusamenhangskomponenten extern zu realieren. Wir haben auch Fortschritte beim Suchen kürzester Pfade gemacht.

Geometrische Fragestellungen sind wichtig bei Anwendungen wie geografischen Informationssystemen, Computergrafik, Chipentwurf oder Computer Aided Design. Verglichen mit Sortieren ist Geometrie eine größere wissenschaftliche Herausforderung, weil es nicht nur größer und kleiner sondern (mindestens) links und rechts sowie über und unter gibt. Wir haben eine Technik von Mulmuley, die ursprünglich für parallele Algorithmen entwickelt wurde, für externe Maschinen angepasst. Diese Technik ist vielseitig anwendbar und liefert insbesondere das beste bekannte Verfahren zum Bestimmen der Schnittpunkte einer Menge von Strecken. Leider können wir auf die Details hier nicht eingehen, wollen aber erwähnen, dass die Technik ein Beispiel für zwei weitere allgemeine Prinzipien hinter externen Algorithmen ist. Zunächst läßt sich zeigen, dass beliebige Algorithmen in einem bestimmten parallelen Modell (genannt **B**ulk **S**ynchronous **P**rocessing) sich rein mechanisch in externe Algorithmen umsetzen lassen. Da Parallelverarbeitung von vielen Forschern betrachtet wurde, ist diese Umsetzung eine wichtige Quelle von externen Algorithmen.

Mulmuleys Algorithmus funktioniert nur dann gut, wenn die Strecken in einer „günstigen“ Reihenfolge betrachtet werden. Leider ist kein effizientes systematisches Verfahren bekannt, das eine günstige Reihenfolge bestimmt! Interessanterweise genügt es aber, die Reihenfolge *zufällig* zu wählen, um im Mittel eine schnelle Verarbeitung zu gewährleisten. Ein Zufallselement spielt bei vielen Algorithmen eine Rolle und ermöglicht oft bessere oder einfachere Lösungen als deterministische Verfahren.

Bei alle bisher erwähnten Beispielen ging es um maßgeschneiderte externe Algorithmen. Diese sind in der Praxis aber noch die Ausnahme. Vielmehr werden Zwischenspeicher für Daten aus dem Hauptspeicher (*Caches*) oder von Platten (*virtueller Speicher*) weitgehend automatisch verwaltet. Der Benutzer startet einen internen Algorithmus und die benötigten Speicherblöcke werden vom Betriebssystem bzw. der Prozessorhardware in einen Zwischenspeicher gelesen und sind dort solange verfügbar, bis sie durch andere (hoffentlich wichtigere) Datenblöcke verdrängt werden. Wir haben mehrere Arbeiten über Strategien für virtuellen Speicher geschrieben. Für Prozessorcaches können wir z.B. zeigen, unter welchen Umständen typische externe Algorithmen (serieller Zugriff auf viele Ein/Ausgabefolgen) durch die sehr einfachen Ersetzungregeln der Hardware effizient unterstützt werden. Nebenbei ist dies das erste theoretische Ergebnis, das die in der Praxis beobachteten Unterschiede zwischen den üblichen Ersetzungsstrategien befriedigend erklärt.

Wir entwerfen nicht nur externe Algorithmen mit Bleistift und Papier sondern implementieren diese auch und setzen sie gezielt in in reale Anwendungen um. Wir arbeiten an einer Erweiterung unserer Algorithmenbibliothek LEDA (**L**ibrary of **E**fficient **D**atatypes and **A**lgorithms) um externe Algorithmen und Datenstrukturen – LEDA-SM (**S**econdary **M**emory). Für das Krebsforschungszentrum Heidelberg entwickeln wir eine Suchmaschine für Gendatenbanken, die biologisch ähnliche Gene zu einem gegebenen Gen auffindet.

Dem aufmerksamen Leser wird aufgefallen sein, dass das einfache zweistufiges Externspeichermodell der realen mehrstufigen Hierarchie nicht gerecht wird. Die meisten Verfahren lassen sich jedoch relativ einfach verallgemeinern und in vielen Anwendungen sind nur zwei der Ebenen wirklich entscheidend. Trotzdem beschäftigen wir uns auch mit allgemeineren Modellen. Insbesondere Hochleistungsrechner mit vielen Magnetplatten und mehreren Prozessoren werfen interessante Fragen auf.

(P. Sanders, S. Albers, S. Burkhart, A. Crauser, K. Mehlhorn, U. Meyer, E. Ramos, J. Sibeyn, u.v.a.m.)