Informatik Spektrum manuscript No.
(will be inserted by the editor)

Algorithm Engineering

Peter Sanders · Dorothea Wagner

Received: date / Accepted: date

Zusammenfassung bleibt leer

Schlüsselwörter keine keywords

1 Einleitung

Effiziente Algorithmen und Datenstrukturen sind Grundvoraussetzung für anspruchsvolle Computeranwendungen. Algorithmik – die systematische Ent-

Teilweise unterstützt durch DFG SPP 1307 Algorithm Engineering SA 933/4-3

Karlsruher Institut für Technologie
Tel.: +49-721-608-47580
Fax: +49-721-608-43088
E-Mail: {sanders,dorothea.wagner}@kit.com

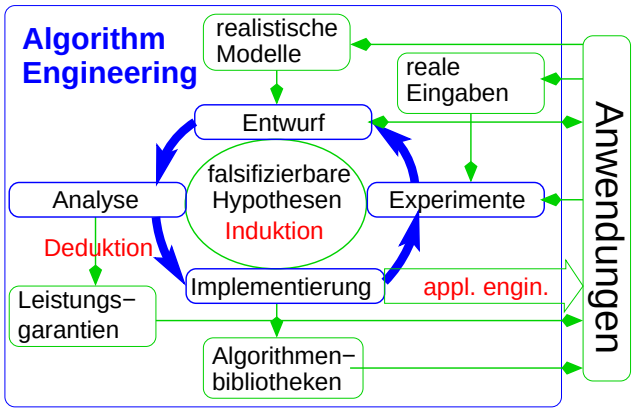

Abb. 1 Algorithm engineering als Kreislauf aus Entwurf, Analyse, Implementierung und experimenteller Analyse.

wicklung effizienter Algorithmen – ist deshalb entscheidend für die Umsetzung technologischer Möglichkeiten in Anwendungen mit großer Bedeutung für Technik, Wirtschaft, Wissenschaft und unser tägliches Leben.

Traditionell hat sich die Algorithmik der Methodik der *Algorithmentheorie* bedient, die aus der Mathematik stammt: Algorithmen werden für einfache und abstrakte Problem- und Maschinenmodelle entworfen. Hauptergebnis sind dann beweisbare Leistungsgarantien für alle möglichen Eingaben. Dieser Ansatz führt in vielen Fällen zu eleganten, zeitlosen Lösungen, die sich an viele Anwendungen anpassen lassen. Die harten Leistungsgarantien ergeben zuverlässig hohe Effizienz auch für zur Implementierungszeit unbekannte Typen von Eingaben. Aufgreifen und Implementieren eines Algorithmus ist aus Sicht der Algorithmentheorie Teil der Anwendungsentwicklung. Nach allgemeiner Beobachtung ist diese Art des Ergebnistransfers aber ein sehr langsamer Prozess. Bei wachsenden Anforderungen an innovative Algorithmen ergeben sich daraus wachsende Lücken zwischen Theorie und Praxis: Reale Hardware entwickelt sich durch Parallelismus, Pipelining, Speicherhierarchien u.s.w. immer weiter weg von einfachen Maschinenmodellen. Anwendungen werden immer komplexer. Gleichzeitig entwickelt die Algorithmentheorie immer ausgeklügeltere Algorithmen, die zwar wichtige Ideen enthalten aber oft kaum implementierbar sind. Außerdem haben reale Eingaben oft wenig mit den worst-case Szenarien der theoretischen Analyse zu tun. Im Extremfall werden viel versprechende algorithmische Ansätze vernachlässigt, weil eine vollständige Analyse mathematisch zu schwierig wäre.

Algorithm Engineering (AE) ist eine Methodik für die Entwicklung und Erforschung von Algorithmen, die hilft die oben beschriebenen Lücken zwischen Theorie und Praxis zu überwinden. AE vereint theoretische und experimentelle Ansätze und erlaubt eine enge Kopplung an Anwendungen. Kern des AE ist ein Kreislauf aus Entwurf, Analyse, Implementierung und experimenteller Bewertung von Algorithmen. Hinzu kommt eine stärkere Einbindung von Anwendungen. Implementierungen können direkt oder indirekt (über wiederverwendbare Algorithmenbibliotheken) in Anwendungen einfließen. Umgekehrt liefern Anwendungen verfeinerte Modelle für den Algorithmenentwurf sowie Szenarien für realistische Experimente. Abbildung 1 zeigt ein Schema dieses Ansatzes. Die Einzelnen Aspekte werden in den folgenden Abschnitten diskutiert.

2 Realistische Modelle

Basis jedes systematischen Algorithmenentwurfs sind mathematische Modelle, die das zu lösende Problem und die ausführende Maschine beschreiben. Entscheidend ist hier die richtige Balance zwischen Einfachheit und Genauigkeit. Besonders deutlich wird das bei den Maschinenmodellen. Während die Algorithmentheorie weitgehend mit dem von Neumann-Modell arbeitet, bei dem ein einfach aufgebauter Prozessor wahlfreien Zugriff auf einen homogen strukturierten Speicher hat, bestehen moderne Computer aus vielen Prozes-

sorkernen und Speichermodulen (Caches, Hauptspeicher, Platten,...), die eine komplexes hierarchisches Netzwerk bilden. Selbst die einzelnen Kerne haben durch verschiedene Formen von Befehlsparallelismus (Pipelining, Superskalarität, Vektorbefehle, spekulative Ausführung,...) ein sehr komplexes Verhalten. Eine genaue Modellierung der Ausführungszeit eines Programms ist deshalb kaum möglich und wäre auch nicht zielführend, weil das Ergebnis weder verständlich noch auf andere Architekturen übertragbar wäre. Das AE setzt deshalb auf Abstraktionen, die einzelne besonders wichtige Aspekte des Maschinenmodells erfassen. Erfolgreich ist zum Beispiel das I/O-Modell, das sich auf zwei Speicherhierarchieebenen beschränkt – einen schnellen Speicher mit beschränkter Kapazität M und einen großen Speicher auf den in Blöcken der Größe B zugegriffen wird [5]. Ein nützliches Modell für Parallelverarbeitung betrachtet p gleichartige Rechnerknoten, die über ein Netzwerk verbunden sind. Knoten interagieren durch Nachrichtenaustausch. Jeder Prozessor kann zu einem Zeitpunkt nur eine Nachricht senden und/oder empfangen und der Zeitbedarf für eine Nachricht der Länge n ist $\alpha n + \beta$ für geeignete Parameter α und β. Dieses Modell passt sehr gut zu Cluster-Computern und führt auch bei many-core Rechnern mit gemeinsamem (NUMA-)Speicher zu effizienten Algorithmen.

Das Abstraktionsniveau für Anwendungsprobleme ist naturgemäß sehr problemabhängig. Bei einem relationalen Datenbanksystem können wir zum Beispiel sehr genau spezifizieren was das Ergebnis einer SQL-Anfrage sein soll. Bei einem Routenplaner für Straßennetzwerke ist das richtige Modell dagegen viel weniger klar. Eine einfache Abstraktion ist die Modellierung durch einen Graphen bei dem Kanten Straßenabschnitte darstellen und Kantengewichte die Fahrzeit oder ein anderes Kostenmaß angeben [3]. Die genaue Modellierung fortgeschrittener Aspekte wie tageszeitabhängige Verkehrsdichte, Stauumfahrungen oder Ampelwartezeiten ist dagegen extrem schwierig und scheitert schon an der Verfügbarkeit geeigneter Daten. Trotzdem können wir hoffen durch geeignete Modelle verbesserte Ergebnisse zu erzielen. Zum Beispiel lassen sich schnelle Routenplanungstechniken für das einfache Modell auf ein zeitabhängiges Modell mit stückweise linearen Fahrzeitfunktionen übertragen [2].

3 Entwurf

Beim Algorithmenentwurf sucht die Algorithmentheorie nach asymptotisch effizienten Algorithmen für den schlechtesten Fall. Im AE interessieren wir uns zusätzlich für Implementierbarkeit. Die Algorithmen sollten also einfach sein und wenn möglich bereits verfügbare Komponenten wiederverwenden. Bei der Effizienz und Lösungsqualität interessiert uns nicht nur der schlechtetste Fall, sondern mehr noch das Verhalten für "typische" Eingaben, das deutlich anders sein kann. Natürlich sind konstante Faktoren in der Ausführungszeit keineswegs zu vernachlässigen. Zum Beispiel ist unser Algorithmus zur Bestimmung minimaler Spannbäume im I/O-Modell theoretisch um einen logarithmischen

Faktor schlechter als als das beste bekannte Verfahren. Für realistische Hardware wird aus diesem Faktor aber ein Konstante und verglichen zu den theoretisch besten Verfahren ergibt sich eine mindestens dreimal schnellerere und sehr einfach zu implementierende Methode [4].

4 Analyse

Die Algorithmenanalyse bleibt im AE wichtig, weil sie in kompakter Form Vorraussagen über das Verhalten des Algorithmus in einer Vielzahl von Situationen erlaubt. Hier entstehen auch neue theoretische Herausforderungen, z.B. bei der Entwicklung von Analysetechniken jenseits der worst-case Analyse oder bei der Analyse von Algorithmen, die schon lange im praktischen Gebrauch sind, sich einer einfachen Analyse aber entziehen [1].

5 Implementierung

Die effiziente Implementierung von Algorithmen ist eine große Herausforderung, weil es große semantische Lücken zwischen der abstrakten Beschreibung eines Algorithmus, höheren Programmiersprachen und der ausführenden Hardware gibt. Diese Lücken werden durch die immer komplexere Hardware eher größer als kleiner.

6 Experimente

Aussagekräftige Experimente sind der Schlüssel zum Schließen des AE-Entwicklungszyklus und erfordern eine sorgfältige Planung, oft hohen Rechenaufwand und eine noch sorgfältigere Interpretation der Ergebnisse. Die experimentelle Methodik kann von der Popper's wissenschaftlicher Methode lernen: Experimente sind getrieben von falsifizierbaren Hypothesen über das Verhalten der untersuchten Algorithmen. Die Hypothesen stammen aus Entwurf, Analyse, Implementierung oder aus vorhergehenden Experimenten. Die Ergebnisse bestätigen, widerlegen oder verfeinern die Hypothesen. Dies liefert dann Ideen für den Entwurf verbesserter Algorithmen, genauere Analyse, oder effizientere Implementierung.

Wichtig ist die Reproduzierbarkeit von Experimenten wichtig. Das bedeutet, dass die involvierten Programme, Eingabedaten und sonstige Ablaufparameter sorgfältig dokumentiert werden müssen. Diese Informationen müssen nach allgemein anerkannten Richtlinien mindestens 10 Jahre aufgehoben werden. Wissenschaftler, die Experimente überprüfen oder unter veränderten Bedingungen wiederholen möchten, sollten Zugang zu diesen Informationen bekommen können. Eine exakte Wiederholung von Experimenten nach mehreren Jahren ist dagegen oft nicht möglich, da die benutzen Rechner nicht mehr existieren und alte Softwareversionen auf neuer Hardware vielleicht nicht mehr

laufen. Deshalb ist es ratsam nicht nur Laufzeiten zu messen sondern auch weniger harwareabhängige Größen. Zum Beispiel kann man bei einem Algrithmus im I/O-Modell das I/O-Volumen dokumentieren.

7 Instanzen und Benchmarks

Sammlungen realistischer Eingaben für ein algorithmisches Problem sind entscheidend für Fortschritte beim AE, weil sie einen einfachen und objektiven Vergleich verschiedener Ansätze erlauben. Zum Beispiel hat das Verfügbarwerden kontinentgroßer Straßengraphen um 2005 zu einer regelrechten Explosion der Leistung von Routenplanungsalgorithmen geführt [3].

Obwohl synthetische Instanzen weniger realistisch sind haben auch diese Vorteile, da sie sich in beliebiger Größe erzeugen lassen, erleichtern sie Skalierbarkeitseexperimente und können als Stresstests dienen.

8 Algorithmenbibliotheken

Algorithmenbibliotheken sind die Schnittstelle zwischen AE und Software Engineering. Die Bibliotheken sollten effizient, leicht benutzbar, portabel und gut dokumentiert sein. Algorithmenbibliotheken erleichtern den Transfer von AE know-how in Anwendungen. Innerhalb des AE vereinfachen sie den Vergleich zwischen Algorithmen und die Konstruktion von darauf aufbauender Funktionalität (siehe auch das Beispiel in Abschnitt 3). Softwareengineering für Algorithmenbibliotheken ist eine besondere Herausforderung, weil es oft einen natürlichen Konflikt zwischen einfacher Benutzbarkeit, Allgemeinheit und Effizienz der Implementierungen gibt. Dazu kommt, dass zur Erstellungszeit der Bibliothek gar nicht alle Anwendungen bekannt sind. Besonders hoch sind auch die Anforderungen an die Zuverlässigkeit, weil ein Benutzer einer Algorithmenbibliothek schnell entnervt aufgibt wenn eine Bibliothek (wirklich oder scheinbar) nicht funktioniert (bekanntlich schreiben viele Programmierer/Arbeitsgruppen ihre Codes lieber selbst).

All diese Schwierigkeiten machen deutlich, dass eine gute Algorithmenbibliothek viel schwieriger zu erstellen ist als die prototypischen Implementierungen, die sonst den AE-Zyklus vorantreiben. Nur ein Bruchteil der implementierten Verfahren wird also seinen Weg in Bibliotheken finden.

Literatur

1. D. Arthur, B. Manthey, and H. Röglin. Smoothed analysis of the k-means method. *J. ACM*, 58(5):19, 2011.
2. G. Batz, D. Delling, P. Sanders, and C. Vetter. Time-dependent contraction hierarchies. In *10th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 97–105, 2009.
3. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS State-of-the-Art Survey*, pages 117–139. Springer, 2009.

4. R. Dementiev, P. Sanders, D. Schultes, and J. Sibeyn. Engineering an external memory minimum spanning tree algorithm. In *IFIP TCS*, pages 195–208, Toulouse, 2004.
5. U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*. Springer, 2003.