Technische Universität Chemnitz-Zwickau
Sonderforschungsbereich 393
Numerische Simulation auf massiv parallelen Rechnern

Uwe Reichel

Partitionierung von Finite-Elemente-Netzen

Preprint SFB393/96_18

Abstrakt: In dieser Arbeit werden spektrale Methoden zur Partitionierung von Finite-Elemente-Netzen vorgestellt und untersucht. Die Auswirkungen verschiedener Methoden der Lastbalancierung auf die Kommunikationszeiten werden speziell anhand des FE-Programmsystems *SPC-PMPo 3D* dargestellt. Abschließend wird aufgrund der Tests die für diese Anwendung beste Methode ermittelt.

Das Hauptaugenmerk dieser Arbeit liegt dabei klar bei der Untersuchung des praktischen Nutzens der vorgestellten Algorithmen. Auf eine detaillierte Beschreibung der einzelnen Algorithmen wird dabei im wesentlichen verzichtet und stattdessen auf die entsprechenden Literaturstellen verwiesen.

Preprint-Reihe des Chemnitzer SFB 393

SFB393/96_18 **Dezember 1996**

Anschrift des Autors:

Uwe Reichel
Irkutsker Str. 255
09119 Chemnitz

email: reichel@mathematik.tu-chemnitz.de

Inhaltsverzeichnis

Danksagung **2**

Notation **2**

1 Einführung **3**
1.1 Allgemeines 3
1.2 Zum Inhalt der Arbeit 4

2 Theorie spektraler Algorithmen **7**
2.1 Einleitung 7
2.2 Über Graphen 7
2.2.1 Begriffe 7
2.2.2 Das Spektrum der Laplace-Matrix 8
2.2.3 Graphenpartitionierung 8
2.3 Spektrale Bisektion 8
2.3.1 Der Algorithmus 8
2.3.2 Berechnung des Fiedlervektors 10
2.4 Partitionierung in mehr als zwei Teile 10
2.5 Praktische Tests 12
2.5.1 Die Testumgebung 12
2.5.2 Die Metriken 13
2.5.3 Zufällige Verteilung, lineare Verteilung und RSB 14
2.5.4 Lokale Nachbesserung mittels KL 15
2.5.5 Quadri- und Oktasektion 16
2.5.6 Weitere Optimierungen im Postprocessing 18

3 Terminal Propagation **23**
3.1 Motivation 23
3.2 Die Idee 24
3.3 Erweiterung des Graphenmodells 25
3.4 Spektrale Bisektion mit Terminal Propagation 26
3.5 Praktische Tests 28
3.5.1 Rekursive Spektralbisektion mit Terminal Propagation 28
3.5.2 Variation der Cut zu Hop Kosten 28

4 Zusammenfassung **31**

A Zugehörige Programme **33**
A.1 *std2graph* 33
A.2 *plotass* 33

B Die FE-Netze **35**

Literaturverzeichnis **37**

Danksagung

An dieser Stelle möchte ich mich recht herzlich bei Prof. Bruce Hendrickson von den Sandia National Laboratories für die kostenlose Überlassung des Programms **Chaco** bedanken. **Chaco** von Bruce Hendrickson und Robert Leland ist eine Implementation aller hier behandelten Algorithmen und wurde für die praktischen Tests genutzt.

Außerdem gilt mein Dank Dr. Thomas Apel, der diese Arbeit betreute und mir stets mit wertvollen Ratschlägen und Hinweisen zu helfen wußte.

Notation

An dieser Stelle folgt eine Liste aller im Text verwendeten Symbole zusammen mit einer kurzen Erläuterung dieser.

$\lvert . \rvert$	: Anzahl der Elemente einer Menge
e	: Vektor, dessen Komponenten alle 1 sind
d	: Dimension
e	: Kante eines Graphs
v	: Ecke eines Graphen
$\mathcal{G}$	: Graph
$\mathcal{V}, \mathcal{V}_i$	: Eckenmenge von Graphen
$\mathcal{E}, \mathcal{E}_i$	: Kantenmenge von Graphen
$\underline{\mathcal{G}}$	: Untergraph (bei Terminal Propagation)
$\underline{\mathcal{V}}, \underline{\mathcal{V}}_i$	: Eckenmenge von Untergraphen
$\underline{\mathcal{E}}, \underline{\mathcal{E}}_i$	: Kantenmenge von Untergraphen
A	: Adjazenzmatrix eines Graphen
L	: Laplace-Matrix eines Graphen
I	: Einheitsmatrix
D	: Diagonalmatrix
$\mathrm{diag}(t_i)$	: Diagonalmatrix, deren Hauptdiagonalelemente die t_i sind
$\deg(v)$	: Grad der Ecke v

In dieser Arbeit wird der *Slanted Stil* genutzt, um tatsächlich existierende Pfade und Dateinamen hervorzuheben. Der `Typewriter Stil` soll spezielle Variablennamen kennzeichnen.

1 Einführung

1.1 Allgemeines

Moderne Anwendungen aus Wissenschaft und Technik erfordern immer höhere Rechenleistungen und Speicherkapazitäten. Um diesen Anforderungen gerecht zu werden, nimmt die Integrationsdichte mikroelektronischer Bauelemente immer weiter zu. Leider ist dies nicht unbegrenzt möglich, und trotz ständiger Erweiterung der technischen Grenzen ist da auch noch eine physikalische Grenze, die nicht mehr zu überwinden ist.

Aus diesem Grund wird seit etwa 15 Jahren ein weiteres Konzept zur Erhöhung der Rechen- und Speicherkapazitäten verfolgt. Dabei gilt es, nicht die Leistung eines einzelnen Chips zu erhöhen, vielmehr wird die Leistungssteigerung durch den Einsatz mehrerer Prozessoren erreicht. Dieses Konzept führte zur Entwicklung der Parallelrechner und ist heute sehr erfolgreich.

Leider ist die Sache nicht ganz so einfach, wie es zunächst scheint. Die Annahme, daß n Prozessoren die n-fache Leistung eines einzelnen Prozessors bringen, stellt sich als Trugschluß heraus (Amdahls Gesetz). Bis auf wenige ideal parallelisierbare Anwendungen ist dieser zu erwartende Speedup nicht zu beobachten. Diese Tatsache resultiert daraus, daß die einzelnen Prozessoren fast nie ganz unabhängig voneinander arbeiten können. Vielmehr ist an gewissen Stellen in einem Programm ein Datenaustausch zwischen den Prozessoren notwendig, den wir im weiteren als *Kommunikation* bezeichnen wollen.[1]

Diese Kommunikation unterliegt nun einigen Regeln:

1. Der sendende Prozessor muß warten, bis der Empfangende bereit ist.
2. Es können nur je zwei Prozessoren gleichzeitig miteinander kommunizieren.
3. Der Datenaustausch ist unidirektional.

Punkt 1 heißt, daß alle Prozessoren zwecks Kommunikation synchronisiert werden müssen. Um dabei unnötige Wartezeiten und damit Performanceverluste zu reduzieren, sollte also der Arbeitsaufwand möglichst gleichmäßig über die Prozessoren verteilt werden, damit alle etwa in der gleichen Zeit ihre Arbeit erledigen und zu Beginn des Datenaustausches auch alle zu diesem bereit sind. Dies wollen wir im weiteren als *Lastbalancierung* bezeichnen.

Punkt 2 und 3 beziehen sich im Gegensatz zu Punkt 1 direkt auf die Form der Kommunikation. Voraussetzung für eine Kommunikation zwischen den Prozessoren ist ein Netzwerk, über das die einzelnen Prozessoren verbunden sind. Da es nun eine Menge möglicher und auch praktisch vorkommender Netzwerktopologien gibt, wollen wir uns auf ein Modell beschränken. Wir nehmen im weiteren an, daß dieses Netzwerk in Form eines *Hypercubes*, also eines n-dimensionalen Würfels, der Dimension $n = \log_2 p$ vorliegt (p ist die Anzahl der Prozessoren – zweckmäßigerweise eine Zweierpotenz). Dies stellt keine große Einschränkung dar, da diese Topologie zumindest virtuell auf allen modernen Parallelrechnern realisiert ist und programmiertechnisch recht günstig zu handhaben ist.

Die Prozessornummern werden im Hypercubemodell so vergeben, daß ihre binäre Darstellung die Position im Hypercube repräsentiert, siehe dazu als Beispiel Abbildung 1. Zwischen Prozessor 0 und den Prozessoren 1, 2 und 4 kann direkt kommuniziert werden – 1 Bit Unterschied. Zwischen Prozessor 0 und den Prozessoren 3, 5 und 6 muß über zwei Links kommuniziert werden – 2 Bit Differenz. Von Prozessor 0 zu Prozessor 7 geht die Kommunikation über drei Links – 3 Bit Differenz.

[1] Die betrachtete Rechnerarchitektur ist ein *MIMD*-Parallelrechner mit *distributed memory* und *synchroner Kommunikation*.

Diese Form des Netzwerkes bringt eine weitere Einschränkung mit sich: Es ist nicht zwischen allen Prozessoren eine direkte Kommunikation möglich. Lediglich Prozessoren, deren Nummern sich in nur einem Bit unterscheiden, können über einen gemeinsamen *Link* kommunizieren. Alle anderen müssen über die Anzahl von Links kommunizieren, um die sie sich in Bits unterscheiden. Bei 128 Prozessoren, also einer Hypercubedimension 7, können das bis zu sieben Links sein!

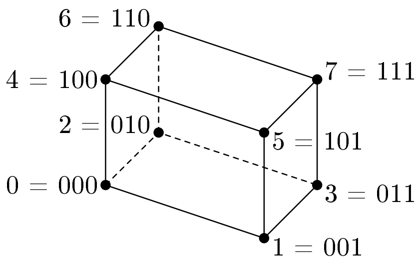


Abbildung 1: *Prozessornumerierung und Bitrepräsentation der Hypercubetopologie in 3D.*

Die Zahl der Links zwischen zwei Prozessoren werden wir im weiteren auch häufig als *Hypercube Hops* oder kurz *Hops* bezeichnen.

Aus der geringen Zahl direkter Verbindungen zwischen Prozessoren und aus der Tatsache, daß der Datendurchsatz (die Bandbreite) des Netzwerkes begrenzt ist, ergeben sich folgende Forderungen an eine möglichst optimale Kommunikation:

1. Die Datenmenge ist so gering wie möglich zu halten.
2. Die Kommunikationswege sind kurz zu halten (wenige Hypercube-Hops).
3. Möglichst viel Kommunikation gleichzeitig durchführen.

Wie man diese Forderungen in der Praxis umsetzt, werden wir im weiteren für die Finite-Elemente-Methode, kurz FEM[2], untersuchen, wobei sich Abschnitt 2 speziell Punkt 1 zuwendet während die Punkte 2 und 3 Inhalt von Abschnitt 3 sind. Die dabei besprochenen Algorithmen sind aber recht allgemeiner Natur und sind durchaus auf andere Anwendungsgebiete übertragbar.

1.2 Zum Inhalt der Arbeit

Wie im letzten Abschnitt erwähnt, wollen wir in dieser Arbeit die Lastbalancierung sowie die Minimierung und Optimierung der Interprozessorkommunikation speziell anhand der FEM[2] untersuchen.

Unser Weg zur Parallelisierung der FEM[2] geht über die Verteilung des verwendeten Nutzernetzes auf die vorhandenen Prozessoren. Auf diese Art und Weise können Schritte wie die Nutzernetzverfeinerung und Assemblierung vollständig parallel durchgeführt werden. Lediglich bei der Auflösung des linearen Gleichungssystems mittels PPCG ist Kommunikation erforderlich und das in jedem Iterationsschritt des CG.

Dabei wird über die sogenannten *Koppelkanten und -flächen* kommuniziert. Der erste Schritt zur Lastbalancierung und Verringerung des Kommunikationsaufwandes wird also eine Partitionierung in etwa gleichgroße Teile bei gleichzeitiger Minimierung der Koppelflächen bei der Verteilung des Nutzernetzes sein.

Da dieses Optimierungsproblem in die Klasse der NP-vollständigen Probleme fällt, ist es notwendig, sich mit Näherungsverfahren zu begnügen. Die in den letzten Jahren entwickelten und getesteten Verfahren besitzen alle heuristischen Charakter, es gibt also keinen genauen Nachweis für die Güte ihrer Ergebnisse, sie haben sich aber in der Praxis recht gut bewährt. Als die wichtigsten Verfahren seien hier der Algorithmus von Kernighan-Lin [15],

[2]Das für die Tests benutzte FEM-Programm löst stationäre 3D-Aufgaben ohne adaptive Netzverfeinerung mit vorkonditioniertem parallelen CG (PPCG) als Löser.

die spektralen Algorithmen und der Multilevel-KL [13] genannt.

In **Abschnitt 2** wollen wir uns mit den spektralen Methoden der Netzpartitionierung beschäftigen. Diese Form der Partitionierung erfordert zwar den höchsten Rechenaufwand liefert aber auch die besten Ergebnisse. Aufgrund der Tatsache, daß wir die Partitionierung als Preprocessing betreiben wollen und unsere Problemgrößen recht klein sind (≈ 1000 Nutzernetzelemente), spielt der Rechenaufwand für uns keine größere Rolle.

Dies ändert sich drastisch bei sehr großen Problemen ($\gg 1000$ Elemente) oder bei dynamischer Anwendung während der Laufzeit. Das soll aber nicht Inhalt dieser Arbeit sein, so daß wir uns auf die spektralen Algorithmen beschränken können.

Am Ende dieses Abschnittes werden wir uns dann noch mit diversen Postprocessingtechniken zur weiteren Optimierung der Partitionierung beschäftigen.

In **Abschnitt 3** werden wir uns dann mit einer Erweiterung des Graphenmodells aus Abschnitt 2 beschäftigen, die eine Kombination von Partitionierung und Abbildung auf die Hypercubetopologie erlaubt. Diese Möglichkeit zur weiteren Optimierung des Kommunikationsverhaltens wird *Terminal Propagation* genannt. Dabei werden während der rekursiven Anwendung der Spektralbisektion neben der Anzahl der Koppelflächen auch die Kommunikationswege minimiert. Ziel ist es dabei, die Lokalität der Daten zu erhöhen, um die Belastung des Netzwerkes bei der Kommunikation zu reduzieren.

Abschnitt 4 soll die erzielten Ergebnisse dann mit einer detailierten Auswertung und Gegenüberstellung der praktischen Tests aus den Abschnitten 2.5 und 3.5 zusammenfassen. Abschließend wird versucht, eine Empfehlung für eine optimale Parameterwahl im verwendeten Partitionierungsprogramm **Chaco** zu geben.

2 Theorie spektraler Algorithmen

2.1 Einleitung

Die spektralen Algorithmen basieren auf Ergebnissen der Graphentheorie. Sie partitionieren Graphen, die wir als zusammenhängend annehmen wollen. Doch zunächst haben wir nur unser FE-Nutzernetz. Wie erhalten wir nun einen geeigneten Graphen?

Der von uns gesuchte Graph ist der sogenannte *Dualgraph* unseres Nutzernetzes, das wir auch als Graphen betrachten wollen. Den Dualgraphen erhalten wir, indem wir jedem Element in unserem Nutzernetz einen Knoten bzw. eine *Ecke* zuordnen und zwischen je zwei, zu benachbarten Elementen gehörenden, Ecken eine *Kante* legen.

Einige grundlegende und im weiteren häufig benutzte Begriffe der Graphentheorie werden in Abschnitt 2.2 definiert. In Abschnitt 2.3 wird die spektrale Bisektion erläutert und in Abschnitt 2.4 werden Algorithmen zur Partitionierung in mehr als zwei Teil vorgestellt. Letztlich wird in Abschnitt 2.5 der praktischen Nutzen der vorgestellten Algorithmen untersucht.

Bemerkung: Die Darstellung der theoretischen Sachverhalte in diesem Abschnitt entspricht im wesentlichen der in [4]. Die Definitionen in Abschnitt 2.2.1 wurden mit Ausnahme der Notation der Vorlesung "Graphentheorie" von Prof. R. Diestel entnommen.

2.2 Über Graphen

2.2.1 Begriffe

Graph Paar $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ disjunkter Mengen, bei dem die Elemente von $\mathcal{E}$ 2-elementige Teilmengen von $\mathcal{V}$ sind und $\mathcal{V} \neq \emptyset$.

$\boldsymbol{\mathcal{V}}$ Eckenmenge eines Graphen; $|\mathcal{V}|$ Eckenzahl eines Graphen.

$\boldsymbol{\mathcal{E}}$ Kantenmenge eines Graphen; $|\mathcal{E}|$ Kantenzahl eines Graphen (auch geschrieben als $\|\mathcal{G}\|$).

Teilgraph Gilt $\mathcal{V}' \subseteq \mathcal{V}$ und $\mathcal{E}' \subseteq \mathcal{E}$, so heißt $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ *Teilgraph* von $\mathcal{G}$.

induziert Ein Teilgraph heißt *induziert* (von $\mathcal{V}'$ in $\mathcal{G}$), wenn er *alle* Kanten $xy \in \mathcal{E}$ mit $x, y \in \mathcal{V}'$ enthält.

Untergraph Induzierter Teilgraph. Bezeichnung: $\underline{\mathcal{G}} = (\underline{\mathcal{V}}, \underline{\mathcal{E}})$.

Weg Ein *Weg* P ist ein Graph der Form

$$\mathcal{V}(P) = \{x_0, x_1, \ldots, x_k\} \quad \mathcal{E}(P) = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\},$$

wobei die x_i paarweise verschieden sind.

Grad Eckengrad, Anzahl der von v ausgehenden Kanten. Bezeichnung: $\deg(v)$.

zusammenhängender Graph Ein Graph heißt *zusammenhängend*, wenn es zwischen je zwei seiner Ecken x, y einen Weg in $\mathcal{G}$ gibt.

Komponente Inklusionsmaximal zusammenhängender Teilgraph von $\mathcal{G}$.

Adjazenzmatrix Matrix $A = (a_{ij})_{n \times n}$ eines Graphen $\mathcal{G}$, die durch

$$a_{ij} := \begin{cases} 1 & \text{falls } v_iv_j \in \mathcal{E} \\ 0 & \text{sonst} \end{cases}$$

definiert ist. Die Elemente der Hauptdiagonale sind alle 0 und die Spalten- und Zeilensumme der i-ten Spalte bzw. Zeile entspricht den Grad der Ecke v_i von $\mathcal{G}$.

Laplace-Matrix Sei A die Adjazenzmatrix eines Graphen $\mathcal{G}$, und sei D eine Diagonalmatrix mit

$$d_{ii} = \deg(v_i) = \sum_{j=1}^{n} a_{ij} \quad \text{mit} \quad i = 1, \ldots, n.$$

Dann nennt man die Matrix $L = D - A$ *Laplace-Matrix* bzw. *Laplacian* von $\mathcal{G}$.

2.2.2 Das Spektrum der Laplace-Matrix

Die Laplace-Matrix ist symmetrisch und positiv semidefinit. Ihre Eigenwerte und Eigenvektoren enthalten wertvolle Informationen über den zugehörigen Graph.

Die Vielfachheit des kleinsten Eigenwertes 0 ist gleich der Anzahl Komponenten des Graphen, wenn 0 ein einfacher Eigenwert ist, dann ist $\mathcal{G}$ zusammenhängend. Miroslav Fiedler nannte den zweitkleinsten Eigenwert des Laplacians die *Algebraische Zusammenhängigkeit* des Graphen und erkannte einige Verbindungen zum Ecken- und Kantenzusammenhang [7]. So spiegelt seine Vielfachheit die Anzahl der Symmetrieachsen des Graphen wieder. Seine Vielfachheit ist also maximal zwei für 2D- bzw. drei für 3D-Probleme.

Der zugehörige Eigenvektor spielt eine wichtige Rolle in der spektralen Bisektion und wird gewöhnlich in Anerkennung Fiedlers Arbeit [8] als *Fiedlervektor* bezeichnet.

2.2.3 Graphenpartitionierung

Die Graphenpartitionierung ist, wie bereits erwähnt, ein NP-vollständiges Optimierungsproblem.

Zunächst ist der Graph $\mathcal{G}$ mit der Eckenmenge $\mathcal{V}$ zusammen mit P natürlichen Zahlen $n_0, \ldots, n_{P-1}$, deren Summe gleich $|\mathcal{V}|$ ist, gegeben[3]. Das Problem besteht dann darin, die Eckenmenge $\mathcal{V}$ so in P disjunkte Teilmengen $\mathcal{V}_0, \ldots, \mathcal{V}_{P-1}$, mit $|\mathcal{V}_j| = n_j$ für $j = 0, \ldots, P-1$ aufzuteilen, daß die Anzahl der Kanten zwischen den Teilmengen minimal wird. Wir sagen, daß eine Kante zwischen zwei Teilmengen durch die Partitionierung *geschnitten* wird.

Die Anzahl der geschnittenen Kanten werden wir im weiteren als *Edge Cuts* bezeichnen. Die Teilmengen $\mathcal{V}_i$ werden wir als *Partitionen* und deren Gesamtheit als *Partitionierung* bezeichnen.

2.3 Spektrale Bisektion

2.3.1 Der Algorithmus

Pothen, Simon und Liou [16] schlugen eine Zweiteilung der Eckenmenge eines Graphen vor, indem die kleineren Komponenten des Fiedlervektors der ersten und die größeren Komponenten des Fiedlervektors der zweiten Partition zugeteilt werden.

Es ist zunächst nicht offensichtlich, daß diese Bisektion nur wenige Kanten schneidet. Die folgende Herleitung des Algorithmus wird uns aber zeigen, daß dies tatsächlich der Fall ist. Wir nehmen an, daß ein Graph $\mathcal{G}$ mit n Ecken (n gerade) in zwei Teilgraphen mit $n/2$ Ecken geteilt werden soll.

[3] Im allgemeinen können sowohl Ecken als auch Kanten des Graphen noch mit Gewichten w_v und w_e versehen sein. Auf Probleme dieser Form werden wir in Abschnitt 3 noch näher eingehen.

Eine Zweiteilung seiner Eckenmenge $\mathcal{V}$ in Teilmengen $\mathcal{V}_0$ und $\mathcal{V}_1$ kann durch einen Vektor der Länge n und den Komponenten

$$x_i = \begin{cases} -1 & \text{für } v_i \in \mathcal{V}_0, \\ 1 & \text{für } v_i \in \mathcal{V}_1, \end{cases} \qquad i = 1, \ldots, n.$$

repräsentiert werden. Dabei ist zu bemerken, daß $\mathcal{V}_0$ und $\mathcal{V}_1$ genau dann beide die Größe $n/2$ haben, wenn $\sum_{i=1}^n x_i = 0$. Weiterhin ist zu bemerken, daß $(x_i - x_j)^2/4$ gleich null ist falls die Ecken v_i und v_j zur selben Teilmenge gehören und gleich eins falls sie zu verschiedenen Teilmengen gehören. Deshalb sind die *Kosten* der Bisektion, repräsentiert durch x, gleich

$$\frac{1}{4} \sum_{e_{ij} \in \mathcal{E}} (x_i - x_j)^2 = \frac{1}{4}(2|\mathcal{E}| - x^T A x), \tag{1}$$

wenn A die Adjazenzmatrix von $\mathcal{G}$ ist.

Es wird sich als vorteilhaft erweisen, noch den Term $\sum_{i=1}^n t_i(x_i^2 - 1)/4$ zu ergänzen. Da $x_i = \pm 1$ ist, ist dieser Term gleich null, aber später werden wir die Forderung $x_i = \pm 1$ etwas auflockern, und der Term bekommt Bedeutung. Sei nun $r = \sum_{i=1}^n t_i$, $D = \operatorname{diag}(t_i)$ und $B = D - A$, dann ergibt sich als neue Gewichtsfunktion

$$\frac{1}{4}(2|\mathcal{E}| - r) + \frac{1}{4} x^T B x \tag{2}$$

Wir wählen die Werte der t_i so, daß alle Zeilensummen in B null sind, das heißt $t_i = \deg(v_i)$. Mit dieser Wahl verschwindet der erste Term in (2), da $t_i = \deg(v_i) \quad r = 2|\mathcal{E}|$ impliziert. Mehrnoch, B ist die Laplace-Matrix L des Graphen. Wenn nun e ein Vektor mit lauter Einsen ist, dann erhalten wir folgendes diskretes Optimierungsproblem:

$$\text{Minimiere } \frac{1}{4} x^T L x \text{ unter der Bedingung, daß } x \in \{-1, 1\}^n \text{ und } \mathrm{e}^T x = 0. \tag{3}$$

Dieses Minimierungsproblem ist, wie bereits erwähnt, NP-vollständig. Dennoch deutet es eine mögliche Approximation an, die zu einem wesentlich einfacheren Problem führt. Anstatt auf der Bedingung, daß alle Komponenten von x exakt ± 1 sein sollen, zu bestehen, erlauben wir ihnen, einen beliebigen Wert anzunehmen, und ersetzen die Bedingung $x \in \{-1, 1\}^n$ durch eine Normbedingung an den Vektor x. Auf diese Art und Weise approximieren wir unser diskretes Problem durch das folgende stetige Optimierungsproblem

$$\text{Minimiere } \frac{1}{4} x^T L x \text{ unter der Bedingung, daß } x^T x = n \text{ und } \mathrm{e}^T x = 0. \tag{4}$$

Auch dieses Problem wird in der praktischen Umsetzung nicht so gelöst, sondern zunächst weiter umgeformt. Auf diese praktischen Details wollen wir hier aber nicht näher eingehen und deshalb sei hier für weitere Informationen auf z.B. [14] verwiesen.

Als Lösung des eigentlichen diskreten Minimierungsproblems nehmen wir den Vektor aus $\{-1, 1\}^n$, der am nächsten an der Lösung des stetigen Problems liegt. Diesen Vektor erhalten wir, indem wir den $n/2$ größten Komponenten von x eine 1 und den $n/2$ kleinsten eine -1 zuordnen.

Im allgemeinen es ist nicht garantiert, daß mit dieser Strategie eine optimale Lösung gefunden wird. In [16] wird das Spektrum regulärer Graphen untersucht und gezeigt, daß für diese Graphen der Fiedlervektor eine optimale Bisektion gewährleistet. Dies gibt Anlaß zu der Vermutung, daß im Fall allgemeiner Graphen letztlich fast optimale Ergebnisse erzielt werden können. Tatsächlich hat sich diese Methode auch in der Praxis gut bewährt.

(a) (b)

Abbildung 2: *(a) Optimale Partitionierung des Netzes in vier Teile. (b) Partitionierung mit rekursiver Spektralbisektion. Das Netz wird zunächst vertikal in der Mitte geteilt, was das Finden einer optimalen Vierteilung verhindert.*

2.3.2 Berechnung des Fiedlervektors

Für die spektrale Bisektion wird der zweitkleinste Eigenwert der Laplace-Matrix und der dazu korrespondierende Eigenvektor benötigt. Da die Laplace-Matrix großdimensioniert, dünn besetzt und symmetrisch ist, bietet sich besonders der *Lanczos*-Algorithmus (siehe z.B. [9]) zur Lösung des Problems an. Dieser Algorithmus basiert im wesentlichen auf Matrix-Vektor-Multiplikationen und profitiert so von der schwachen Besetztheit der Matrix. Außerdem konvergiert er typischerweise in $\mathcal{O}(\sqrt{n})$ Schritten gegen die extremen Eigenwerte einer $n \times n$ Matrix.

Für sehr große Graphen ist der Lanczos-Algorithmus dennoch zu aufwendig in Bezug auf Rechenzeit und Speicherbedarf, so daß dafür Multilevelmethoden angebracht sind, siehe z.B. [3], doch damit wollen wir uns hier nicht beschäftigen, da es für unsere Problemgrößen nicht von Bedeutung ist.

2.4 Partitionierung in mehr als zwei Teile

Die spektrale Bisektion, wie jede andere Bisektion, kann rekursiv angewendet werden, um den Graphen in mehr als zwei Partitionen zu teilen. Doch diese Vorgehensweise hat den Nachteil, daß die Bisektion in keiner früheren Phase einen etwas schlechteren Schnitt machen kann, der später Vorteile bringen würde, im allgemeinen wird jede Bisektion für sich durchgeführt ohne Vorausschau auf noch folgende Bisektionen.

Abbildung 2 zeigt das Problem. Abbildung 2-(a) zeigt die optimale Partitionierung eines T-förmigen Netzes mit 400 quadratischen Elementen in vier Teile. Die rekursive Spektralbisektion liefert die Partitionierung in Abbildung 2-(b). Dieser Algorithmus teilt das Netz zunächt vertikal in der Mitte, was tatsächlich die optimale Partitionierung wäre, wenn unser Ziel nur eine Bisektion wäre. Aber wenn das Netz in vier Teile zu partitionieren ist, verhindert diese erste Zweiteilung das Auffinden einer Quadrisektion nahe dem Optimum. Für die tatsächlich erreichte Partitionierung ist die Länge der Schnittkanten 50, obwohl sie im optimalen Fall 30 wäre.

Deshalb wäre es von Interesse, einen spektralen Algorithmus zu finden, der einen Graphen in mehr als zwei Teile in einem Schritt partitioniert. Solche Algorithmen wurden in verschiedenen Form entwickelt.

F. Rendl und H. Wolkowicz [17] beschreiben einen spektralen Algorithmus, der einen Graphen in P Teile partitioniert. Doch dieser Algorithmus erfordert die Berechnung von P Eigenvektoren, was im allgemeinen unakzeptabel teuer ist.

B. Hendrickson und R. Leland [12] entwickelten einen Algorithmus, der Graphen in 2^d Teile unter Ausnutzung von nur d Eigenvektoren der Laplace-Matrix. Ihr Algorithmus minimiert nicht die Edge Cuts, sondern versucht vielmehr die Partitionierung im Sinne der *Hypercube Hop* bzw. *Manhattan* Metrik zu minimieren. Das heißt, daß für jedes Paar benachbarter Ecken, die verschiedenen Teilgraphen zugeordnet werden, die Wichtung[4] der Verbindungskante mit der Anzahl der Bits, um die sich die Nummern der Teilgraphen in binärer Darstellung unterscheiden, multipliziert wird. Empirische Studien [10] haben gezeigt, daß dies ein aussagekräftiges Maß für die Modellierung der Performance auf Hypercubearchitekturen ist, da die Minimierung dieser Metrik mit den Forderungen der Punkte 1 und 2 aus Abschnitt 1.1 für diese Netzwerktopologie korrespondiert, siehe auch Abbildung 3.

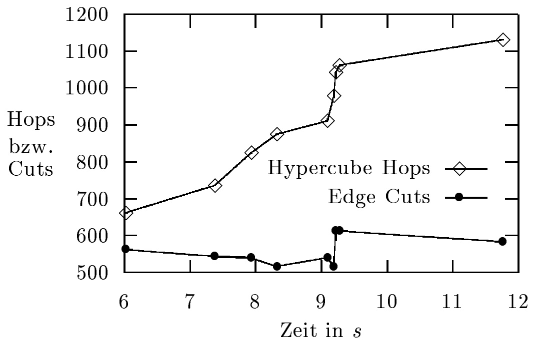


Abbildung 3: *Edge Cuts bzw. Hypercube Hops im Verhältnis zur Rechenzeit für verschiedene Verteilungen von cube768.std auf 64 Prozessoren. Offenbar existiert ein funktioneller Zusammenhang zwischen der Zeit und den Hypercube Hops, was bei den Edge Cuts nicht der Fall ist.*

Die Vorgehensweise von Hendrickson und Leland zur Erarbeitung ihres Algorithmus ist ähnlich der Herleitung des Bisektionsalgorithmus in Abschnitt 2.3. Das Partitionierungsproblem wird zunächst mathematisch als NP-vollständiges diskretes Optimierungsproblem formuliert, welches dann unter Lockerung einiger Bedingungen auf ein handlicheres stetiges Problem reduziert wird. Die Lösung des stetigen Problems wird dann auf die naheliegendste diskrete Lösung abgebildet.

Für das Quadrisektionsproblem ($d = 2$) erhielten sie eine 1-dimensionale Mannigfaltigkeit von Lösungen des stetigen Problems: jedes Paar möglicher normalisierter orthogonaler Linearkombinationen aus dem zweiten und dritten Eigenvektor der Laplace-Matrix ist eine Lösung. Hendickson und Leland schlugen vor, die Lösung so zu wählen, daß der Abstand zur naheliegendsten diskreten Lösung minimal ist.

Im Fall der Oktasektion ($d = 3$) erhielten sie eine 2-dimensionale Mannigfaltigkeit von Lösungen, bestehend aus Tripeln orthogonaler Linearkombinationen aus dem zweiten bis vierten Eigenvektor. In diesem Fall werden die Freiheitsgrade ebenso durch Wahl der Lösung mit der minimalen Entfernung zur naheliegendsten diskreten Lösung eliminiert.

Theoretisch kann diese Prozedur auch für $d > 3$ angewandt werden, einige Implementationsdetails werden aber komplizierter. Für $d = 4$ gibt es 6 Freiheitsgrade im Lösungsunterraum und 5 Balancebedingungen. Diese Bedingungen ergeben sich aus drei Gleichungen dritten Grades und einer vierten Grades, so daß der rechnerische Aufwand sehr hoch wäre. Wenn $d > 4$ ist, übersteigt die Zahl der Bedingungen die der Freiheitsgrade, so ist es im allgemeinen nicht mehr möglich eine ausbalancierte Partitionierung aus den Eigenvektoren zu den $d + 1$ kleinsten Eigenwerten der Laplace-Matrix zu konstruieren. Für eine detailliertere Beschreibung siehe [12].

Hendrickson und Leland zeigten empirisch, daß die mit ihrem Quadri- und Oktasektionsalgorithmus erzielten Partitionierungen besser sind als die mittels rekursiver Spektralbisektion erhaltenen. Und dies speziell dann, wenn die Kosten in der Hypercube Hop Metrik gemessen werden.

Eine Verallgemeinerung der Quadrisektion von Hendrickson und Leland kann in [4] gefunden werden.

[4]Das Graphenmodell von Hendrickson und Leland erlaubt eine Wichtung von Ecken und Kanten zur Steuerung der Partitionierung, was wir aber nicht weiter betrachten wollen. In unserem Fall sind alle Ecken und Kanten mit 1 gewichtet.

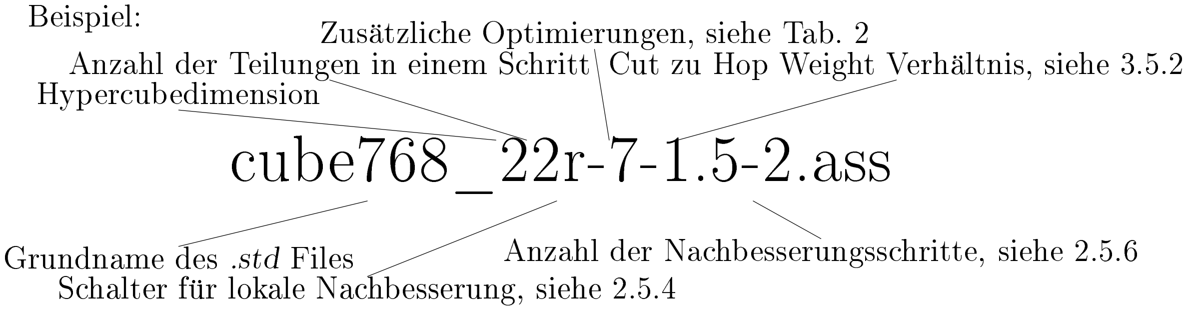


Abbildung 4: *Namenskonvention für das* **Chaco**-*Ausgabefile*

2.5 Praktische Tests

2.5.1 Die Testumgebung

Wie bereits erwähnt, wurden alle Partitionierungen für die Tests mit **Chaco** von Hendrickson und Leland berechnet [11]. Das Programm liegt in ausführbarer Form im Verzeichnis

∼reichel/Chaco-2.0/exec

in der Fakultät für Mathematik als *chaco.sun* für Sun4 Workstation vor. Es ist für jedermann ausführbar und sucht das Konfigurationsfile *User_Params* immer im aktuellen Verzeichnis. Eine Dokumentation und weitere Beschreibungen zu den Algorithmen liegen in

∼reichel/Chaco-2.0/doc

in gepackter Form vor. Sie sind frei kopierbar. Ein Kopieren des Programms oder der Quellfiles ist leider nicht möglich, da dies der Lizenzvereinbarung mit den Sandia National Laboratories widersprechen würde.

Die Erzeugung des Eingabefiles für **Chaco** ist mittels *std2graph* möglich, das sich im Verzeichnis

∼reichel/Chaco-2.0/tools

befindet. Es erzeugt das Eingabefile für **Chaco** aus dem *.std* Eingabefile für *SPC-PM Po 3D*, siehe auch Abschnitt A.1.

Als permanente Parameter für **Chaco** waren gesetzt:

```
OUTPUT_ASSIGN = TRUE
LANCZOS_TYPE = 1
```

was die Ausgabe der Partitionierung in ein File veranlaßt und im Lanczos-Algorithmus für vollständige Reorthogonalisierung sorgt. Das von **Chaco** ausgegebene File unterliegt der Namenskonvention nach Abbildung 4.

Der im Beispiel gezeigte Dateiname ist also folgendermaßen zu entschlüsseln: Das Partitionierungsfile gehört zu *cube768.std* und ist für 4 Prozessoren. Es ist mittels Spektralbisektion mit lokaler Nachbesserung durch KL entstanden. Dabei waren die Parameter `REFINE_MAP`, `TERM_PROP` und `INTERNAL_VERTICES` aktiv. Außerdem stand der Parameter `CUT_TO_HOP_COST` auf 1.5 und es wurden 2 Schritte globale Nachbesserung mittels KL als Postprocessing durchgeführt.

Alle Tests wurden mit dem FE-Programmpaket *SPC-PM Po 3D* V2.31 des Sonderforschungsbereichs 393 der TU Chemnitz-Zwickau durchgeführt, siehe [1, 2]. Dazu wurde das

Bit	Bedeutung
1.	`REFINE_MAP = TRUE` – Anpassung der Partitionierung an die Hypercubetopologie, siehe 2.5.6
2.	`TERM_PROP = TRUE` – Terminal Propagation, siehe Abschnitt 3 (Achtung! – nur aktiv bei rekursiver Spektralbisektion)
3.	`INTERNAL_VERTICES = TRUE` – Erhöhung der Zahl innerer Ecken, siehe 2.5.6

Tabelle 2: *Binäre Kodierung der möglichen Optimierungen (Bit=1 heißt: Optimierung angewandt)*

Programm um die Verteilungsvariante 3 erweitert, die Partitionierungen aus den Verzeichnissen *ass3* bzw. *ass4* einliest. Diese Verzeichnisse enthalten alle getesteten Partitionierungen.

Getestet wurde mit *cube768.std* als regelmäßiges 3D-Tetraedernetz aus 768 Elementen und *spc3-123.std* als unregelmäßiges 3D-Tetraedernetz aus 1398 Elementen. Das Programm wurde mit *bsp.xy* gelinkt und BPX Vorkonditionierung benutzt. Es kam die Kommunikationsvariante 3 zum Einsatz, der das Hypercubemodell zugrunde liegt.

Die Rechnungen wurden auf dem Parsytec GCPowerPlus-128 an der Technischen Universität Chemnitz-Zwickau durchgeführt, wobei stets mit zwei Prozessoren pro Knoten gerechnet wurde.

2.5.2 Die Metriken

Das Partitionierungsprogramm **Chaco** bietet die Möglichkeit der Bewertung der erhaltenen Partitionierung durch 7 Metriken:

Es sei: $\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j} \subset \mathcal{E} \quad : \quad \{e_{\ell m} : v_\ell \in \mathcal{V}_i, v_m \in \mathcal{V}_j\}$.

Set Size: Summe über die Gewichte der Ecken in einer Partition (da in unserem Fall alle Ecken die Wichtung 1 haben, entspricht dies der Anzahl von Ecken in einer Partition).**Chaco** gibt neben der Gesamtzahl der Ecken des Graphen auch die maximale und minimale Anzahl von Ecken pro Partition aus.

$$\min_i |\mathcal{V}_i|, \quad \max_i |\mathcal{V}_i|, \quad \sum_i |\mathcal{V}_i|.$$

Dies liefert ein Maß für die Ausgewogenheit der Partitionierung.

Edge Cuts: Summe über die Gewichte der Schnittkanten (in unserem Fall Anzahl der Schnittkanten, da alle Gewichte 1 sind).

$$\min_{i\neq j} |\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j}|, \quad \max_{i\neq j} |\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j}|, \quad \sum_{i\neq j} |\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j}|.$$

Dies ergibt ein aussagekräftiges Maß für die Güte der Partitionierung.

Hypercube Hops: Ein Maß, bei dem jede Schnittkante mit der Anzahl der Links zwischen ihren Endecken multipliziert wird.

$$\min_{i\neq j} h_{ij}|\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j}|, \quad \max_{i\neq j} h_{ij}|\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j}|, \quad \sum_{i\neq j} h_{ij}|\mathcal{E}_{\mathcal{V}_i\mathcal{V}_j}|,$$

mit h_{ij} Anzahl der Bits, um die sich i und j in Dualdarstellung unterscheiden. Diese Metrik modelliert den Kommunikationsaufwand oft besser als die Edge Cuts, da sie das Netzwerk berücksichtigt. Diese Metrik ist also ein Maß für die Güte der Partitionierung in Verbindung mit der Zuordnung auf die Prozessoren.

Boundary Vertices: Summe über die Gewichte der Ecken mit Schnittkanten. Beispiel: Wenn eine ungewichtete Ecke in Partition 1 drei Verbindungskanten zu Partition 4 hat ist ihr Beitrag zur Boundary-Vertice-Metrik 1. Hat sie außerdem eine Kante zu Partition 7, dann ist ihr Beitrag 2. Dies ist nützlich bei der Modellierung von Anwendungen wie der parallele Matrix-Vektor-Multiplikation, bei denen der Zahlenwert, symbolisiert durch die Ecke, nur einmal an eine andere Partition zu übertragen ist aber doch mehrfach genutzt wird.

Boundary Vertex Hops: Grenzecken gewichtet durch die Anzahl der Links über die ein Datum zwischen korrespondierenden Prozessoren "wandern" muß. Dies erweitert die Boundary-Vertex-Metrik so, daß der Aufwand berücksichtigt wird.

Adjacent Sets: Die Ecken in einer speziellen Partition haben im allgemeinen Kanten, die sie mit anderen Partitionen verbinden. Diese Metrik zählt die Anzahl solch anderer Partitionen. Dieser Wert entspricht der Anzahl der send-Prozeduren, die der Prozessor ausführen muß, unter der Voraussetzung, daß direkt gesendet wird. Die Metrik wird von **Chaco** als Maximum, Minimum und Summe über alle Partitionen ausgegeben.

Internal Vertices: Die Summe über die Gewichte aller Ecken einer Partition, die keine Kanten zu anderen Partitionen haben (in unserem Fall der ungewichteten Ecken die Anzahl innerer Ecken).

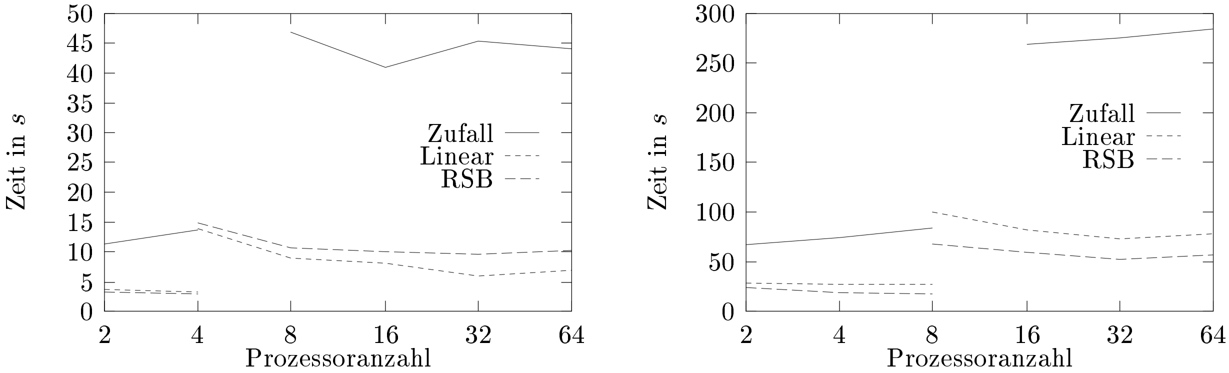
$$\min_i |\{v_\ell \in \mathcal{V}_i : \forall e_{\ell m} \quad \text{ist} \quad v_m \in \mathcal{V}_i\}|,$$
$$\max_i |\{v_\ell \in \mathcal{V}_i : \forall e_{\ell m} \quad \text{ist} \quad v_m \in \mathcal{V}_i\}|,$$
$$\sum_i |\{v_\ell \in \mathcal{V}_i : \forall e_{\ell m} \quad \text{ist} \quad v_m \in \mathcal{V}_i\}|.$$

Wie noch in Abschnitt 2.5.6 näher ausgeführt wird, kann die Existenz solcher Ecken eine Überlappung von Kommunikation und Rechnung ermöglichen.

2.5.3 Zufällige Verteilung, lineare Verteilung und Rekursive Spektralbisektion

Als erstes wollen wir zunächst die zufällige Verteilung als schlechtesten Fall und die lineare Verteilung als einfachsten Fall der rekursiven Spektralbisektion gegenüberstellen. Die erzielten Rechenzeiten zeigt die Abbildung 5. In den Tabellen 3 werden die zugehörigen Metriken für den Fall $d = 6$ angegeben.

Wie erwartet führt eine zufällige Verteilung für beide Netze zu extremen Rechenzeiten. Bei der linearen Verteilung zeigt sich aber ein überraschendes Bild. Abbildung 5 links zeigt, daß die lineare Verteilung zu deutlich besseren Rechenzeiten führt als die rekursive Spektralbisektion. Daß dieses Ergebnis netzspezifisch und keinesfalls allgemeingültig ist, zeigt Abbildung 5 rechts eindrucksvoll. Das sehr gute Ergebnis der linearen Verteilung bei der Partitionierung von *cube768.std* resultiert zum einen aus der durch 64 teilbaren Elementanzahl und zum anderen wohl hauptsächlich aus der Regelmäßigkeit des Netzes und dessen günstiger Elementnumerierung. Bei einem sehr unregelmäßigen Netz, wie *spc3-123.std* erweist sich die rekursive Spektralbisektion als überlegen, da sie unabhängig von der Form und der Numerierung des Netzes arbeitet.



(*cube768.std*) (*spc3-123.std*)

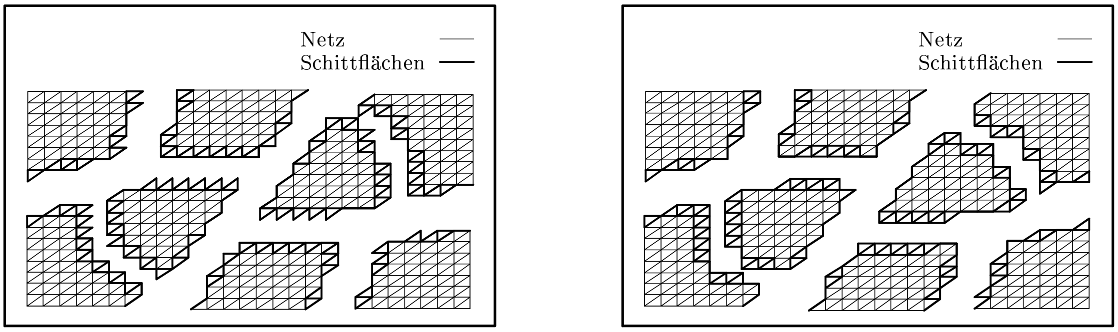
Abbildung 5: *Rechenzeiten verschiedener Verteilungen für Hypercubedimension* $d = 1 \ldots 6$.

cube768.std				*spc3-123.std*			
	Zufall	Linear	RSB		Zufall	Linear	RSB
Set Size (max./min.)	12/12	12/12	12/12	Set Size (max./min.)	22/21	22/21	22/21
Edge Cuts	1390	544	584	Edge Cuts	2232	655	526
Hypercube Hops	4181	736	1130	Hypercube Hops	6778	1834	926
Boundary Vertices	2700	1072	1019	Boundary Vertices	4300	1240	862
Boundary Vertex Hops	8127	1456	2016	Boundary Vertex Hops	13300	3536	1548
Adjacent Sets	1964	320	500	Adjacent Sets	2748	354	256
Internal Vertices	0	68	118	Internal Vertices	0	470	682

Tabelle 3: *Metriken der zufälligen und linearen Verteilung sowie der rekursiven Spektralbisektion für eine Verteilung auf 64 Prozessoren.*

2.5.4 Lokale Nachbesserung mittels KL

Die Heuristik von Kernighan und Lin (im weiteren kurz KL) ist ein iterativer Algorithmus, der versucht, eine gegebene Partitionierung durch fortgesetztes Austauschen von Randelementen zwischen benachbarten Gebieten zu verbessern. Ausgehend von einer zufälligen Verteilung des Netzes auf die Prozessoren, liefert dies aufgrund der lokalen Natur des Algorithmus meist unbefriedigende Ergebnisse. Andererseits erzeugt die rekursive Spektralbisektion meist global gute Partitionierungen, die aber im Detail nicht optimal sind. Daher ist es vorteilhaft, zunächst mit rekursiver Spektralbisektion eine Ausgangsverteilung zu berechnen und diese dann mittels KL nachzubessern.



(a) (b)

Abbildung 6: *Tetraedernetz mit 1800 Elementen mittels rekursiver Spektralbisektion in 8 Partitionen geteilt. (a) normale RSB. (b) RSB mit Nachbesserung mittels Kernighan-Lin.*

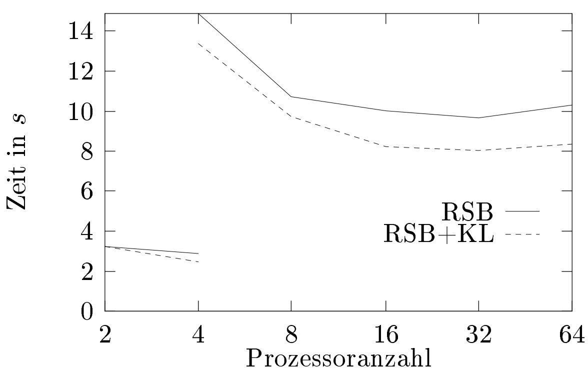
Als Beispiel betrachten wir Abbildung 6. Während im linken Teil der Abbildung das Ergebnis der reinen rekursiven Spektralbisektion mit sehr rauhen Grenzen zu sehen ist,

ist die rechte Partitionierung mit KL nachbearbeitet worden, was offensichtlich zu deutlich glatteren Schnittflächen führt. Diese Beobachtung wird auch durch die Werte in Tabelle 4 belegt. Die Anzahl der Edge Cuts sinkt durch die Nachbesserung von 165 auf 142 und die Zahl der Hypercube Hops von 244 auf 217.

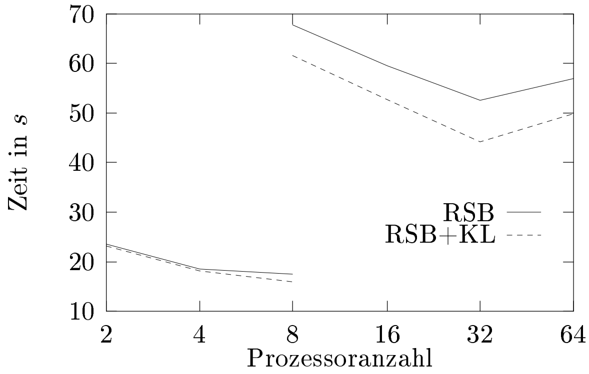
	RSB	RSB+KL
Edge Cuts	165	142
Hypercube Hops	244	217
Boundary Vertices	267	271
Boundary Vertex Hops	401	420
Adjacent Sets	30	30

Tabelle 4: *Metriken für 3D Tetraedernetz mit 1800 Elementen bezogen auf den Dualgraphen.*

Auch in unseren beiden Testbeispielen erzielen wir ein deutliche Verbesserung der Partitionierung. Abbildung 7 zeigt die Verbesserung der Rechenzeiten durch die lokale Nachbesserung und die Tabellen 5 enthalten die zugehörigen Metriken.



(*cube768.std*)



(*spc3-123.std*)

Abbildung 7: *Rechenzeiten der rekursiven Spektralbisektion mit und ohne lokaler Nachbesserung mittels KL für Hypercubedimension* $d = 1 \ldots 6$.

2.5.5 Quadri- und Oktasektion

Wie bereits in Abschnitt 2.4 besprochen, scheint es aus verschiedenen Gründen vorteilhaft das Problem in mehr als zwei Teile pro Rekursionsschritt zu teilen. Deshalb wollen wir nun den praktischen Nutzen der Quadri- und Oktasektionsalgorithmen von Hendrickson und Leland [12] untersuchen. Wir werden die erzielten Ergebnisse mit dem bislang Besten, der

cube768.std		
	RSB	RSB+KL
Edge Cuts	584	516
Hypercube Hops	1130	978
Boundary Vertices	1019	984
Boundary Vertex Hops	2016	1892
Adjacent Sets	500	468
Internal Vertices	118	115

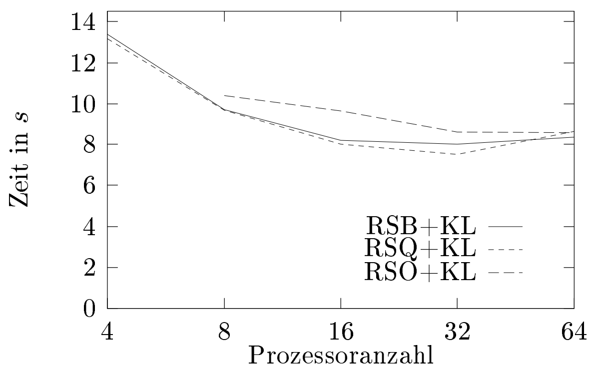
spc3-123.std		
	RSB	RSB+KL
Edge Cuts	526	458
Hypercube Hops	926	708
Boundary Vertices	862	854
Boundary Vertex Hops	1548	1335
Adjacent Sets	256	250
Internal Vertices	682	659

Tabelle 5: *Metriken der rekursiven Spektralbisektion mit und ohne lokale Nachbesserung für eine Verteilung auf 64 Prozessoren.*

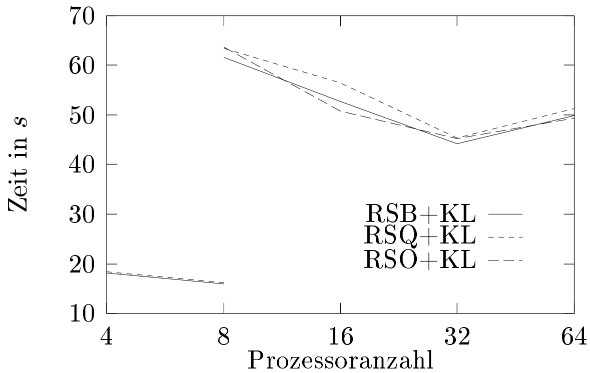
cube768.std			
	RSB+KL	RSQ+KL	RSO+KL
Edge Cuts	516	540	613
Hypercube Hops	978	911	1061
Boundary Vertices	984	1025	1156
Boundary Vertex Hops	1892	1756	2030
Adjacent Sets	468	436	546
Internal Vertices	115	106	79
spc3-123.std			
	RSB+KL	RSQ+KL	RSO+KL
Edge Cuts	458	469	504
Hypercube Hops	708	689	749
Boundary Vertices	854	872	949
Boundary Vertex Hops	1335	1296	1415
Adjacent Sets	250	236	284
Internal Vertices	659	659	616

Tabelle 6: *Metriken der rekursiven spektralen Bi-, Quadri- und Oktasektion mit lokaler Nachbesserung für eine Verteilung auf 64 Prozessoren.*

rekursiven Spektralbisektion mit lokaler Nachbesserung durch KL, vergleichen. Dabei ist zu beachten, daß eine Quadrisektion natürlich erst ab eine Hypercubedimension von $d = 2$ möglich ist, und im weiteren auch nur bei einer Vervierfachung der Prozessorzahl. Bei ungerader Hypercubedimension wird im letzten Schritt bisektioniert. Ähnliches gilt auch für die Oktasektion, die erst ab $d = 3$ möglich ist, und im weiteren bei Verachtfachung der Prozessorzahl. Ist dies nicht der Fall, wird im letzten Schritt bi- oder quadrisektioniert. Da es sich als Vorteil erwies, lokal nachzubessern, nutzen wir auch hier diese Möglichkeit. Die erreichten Rechenzeiten sind in Abbildung 8 für unsere beiden Testbeispiele dargestellt. Die



(*cube768.std*)



(*spc3-123.std*)

Abbildung 8: *Rechenzeiten der rekursiven spektralen Bi-, Quadri- und Oktasektion jeweils mit lokaler Nachbesserung mittels KL für Hypercubedimension* $d = 2 \ldots 6$.

Metriken der Verteilungen, wieder für 64 Prozessoren als Beispiel, sind in den Tabellen 6 zusammengestellt. Die Ergebnisse zeigen leider nicht die gewünschte Verbesserung der Partitionierungen. Speziell die rekursive Oktasektion bringt nicht den gewünschten Erfolg. Am recht zwiespältigen Abschneiden dieser Algorithmen, die teilweise eine leichte Verbesserung aber auch zum Teil eine leichte Verschlechterung gegenüber der Bisektion erbrachten, ist hier sicherlich am besten der heuristische Aspekt dieser Verfahren zu erkennen. Offenbar reagiert die Approximation des Optimierungsproblems, die bei der Quadri- und Oktasek-

cube768.std						
Proz.	RSB+KL	RSB+KL+RM	RSQ+KL	RSQ+KL+RM	RSO+KL	RSO+KL+RM
4	76	76	64	64	—	—
8	180	150	188	132	289	289
16	384	317	338	338	491	485
32	660	561	597	544	759	735
64	978	875	911	824	1061	1043
spc3-123.std						
Proz.	RSB+KL	RSB+KL+RM	RSQ+KL	RSQ+KL+RM	RSO+KL	RSO+KL+RM
4	52	52	56	56	—	—-
8	93	93	115	103	125	125
16	206	177	238	201	212	204
32	395	331	449	393	459	432
64	708	615	689	689	749	733

Tabelle 7: *Hypercube-Hop-Metriken der rekursiven spektralen Bi-, Quadri- und Oktasektion mit lokaler Nachbesserung und mit und ohne Anpassung der Verteilung an das Hypercubemodell.*

tion noch wesentlich weitreichender als bei der Standardbisektion ist [12], doch sehr empfindlich auf wechselnde Bedingungen. Trotzdem werden wir die Quadri- und Oktasektion noch nicht beiseite lassen und im nächsten Abschnitt auch diese Algorithmen mit weiteren Postprocessing-Routinen kombinieren und sehen, ob sie eventuell dann Vorteile erbringen.

2.5.6 Weitere Optimierungen im Postprocessing

Anpassung der Partitionierung an das Hypercubemodell Bei dieser Form des Postprocessings wird versucht, die Partitionen so umzunumerieren, daß eine optimale Abbildung der Partitionierung auf die Hypercubearchitektur erreicht wird. Die Partitionierung selbst ist davon aber nicht betroffen, dies wird erst durch Terminal Propagation möglich, siehe Abschnitt 3.

Um die Anpassung zu erreichen, untersucht der Algorithmus, wie sich die Abbildung ändert, wenn zwei miteinander über Links verbundene Partitionen getauscht werden. Der Tausch, der zur maximalen Verbesserung der Abbildung führt, wird vorgenommen, und der Algorithmus arbeitet weiter, bis keine Verbesserung mehr zu erreichen ist.

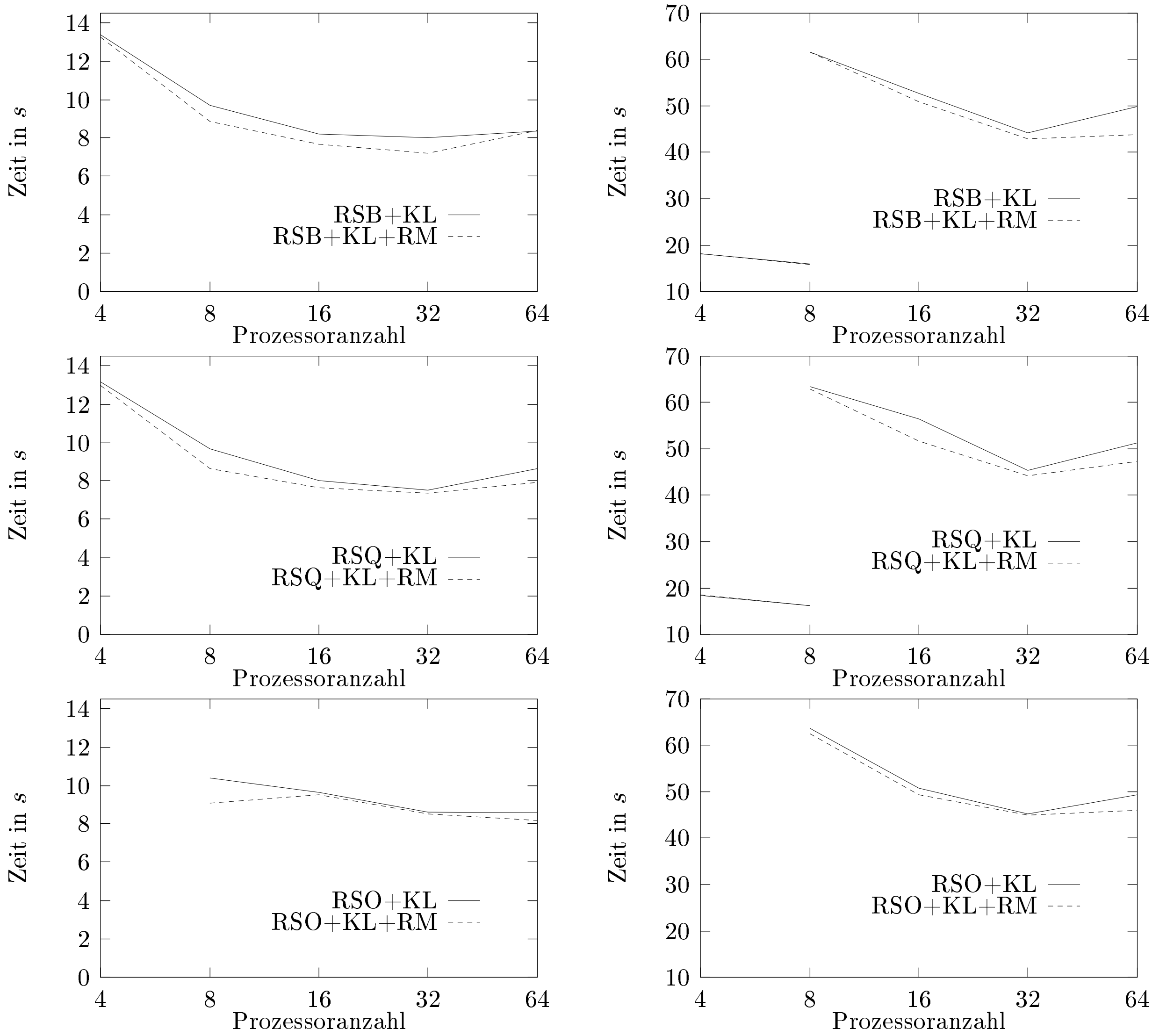
Zur Aktivierung dieser Postprocessing-Prozedur ist der Parameter `REFINE_MAP = TRUE` im File *User_Params* zu setzen. Im Dateinamen des Partitionierungsfiles ist er laut Abbildung 4 und Tabelle 2 verschlüsselt. Für weitere Informationen siehe [11].

Das praktische Ergebnis ist in Abbildung 9 dargestellt. Bei gleicher Anzahl von Edge Cuts sinkt durch dieses Postprocessing die Zahl der Hypercube Hops, siehe Tabelle 7. Dies ergibt dann die geringere Rechenzeit, verglichen mit Abbildung 8.

Wie sowohl aus Abbildung 9 als auch aus Tabelle 7 hervorgeht, ist diese Form des Postprocessing in jedem Fall zu empfehlen. Voraussetzung dafür ist natürlich, daß den Kommunikationsalgorithmen das Hypercubemodell zugrundeliegt. Egal ob bei der Bi-, Quadri- oder Oktasektion, in jedem Fall ergibt sich eine gut sichtbare Verringerung der Rechenzeit.

Im Vergleich der Bi-, Quadri- und Oktasektion, den wir in Abschnitt 2.5.5 begonnen haben, ergibt sich nichts grundlegend Neues, wie aus Abbildung 10 ersichtlich.

Erhöhung der Anzahl innerer Ecken Die Erhöhung der Anzahl innerer Ecken kann aus zwei Gründen nützlich sein: Über innere Ecken ist nicht zu kommunizieren, und ihre Bearbeitung erfordert nur lokale Daten. Dies verringert zum einen das Kommunikationsaufkommen und ermöglicht außerdem die Überlappung von Kommunikation und Rechnung,



(*cube768.std*) (*spc3-123.std*)

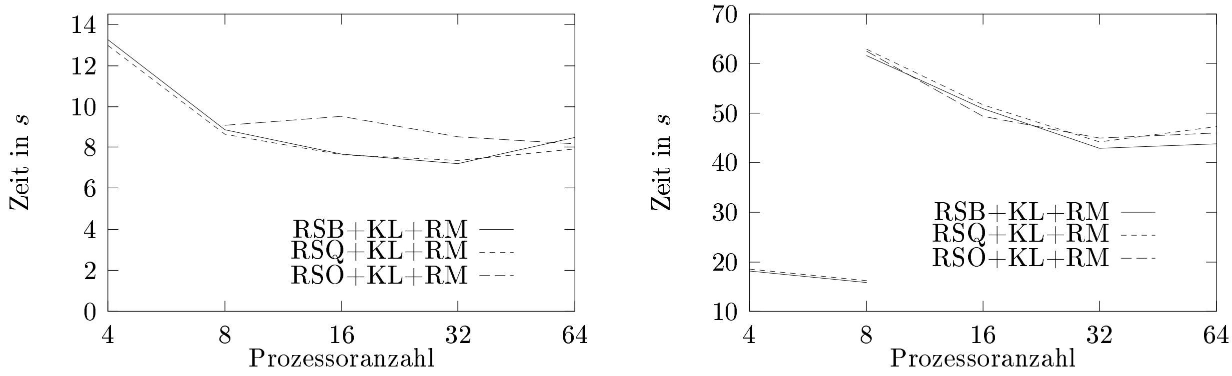
Abbildung 9: *Rechenzeiten der lokal KL-verfeinerten rekursiven spektralen Bi-, Quadri- und Oktasektion mit und ohne spezieller Anpassung der Verteilung an das Hypercubemodell für Hypercubedimension* $d = 2 \ldots 6$.

da innere Ecken aufgrund ihrer Datenlokalität bearbeitet werden können, während für die Bearbeitung von Grenzecken auf Daten anderer Prozessoren zu warten ist.

Um diese Vermehrung innerer Ecken zu erreichen, bestimmt der Algorithmus zunächst die Anzahl innerer Ecken in jeder Partition. Dann bekommt die Partition mit den wenigsten inneren Ecken Ecken von anderen Partitionen, um einige ihrer Ecken zu inneren zu machen, und gibt andere Ecken ab, um die Lastbalance zu wahren.

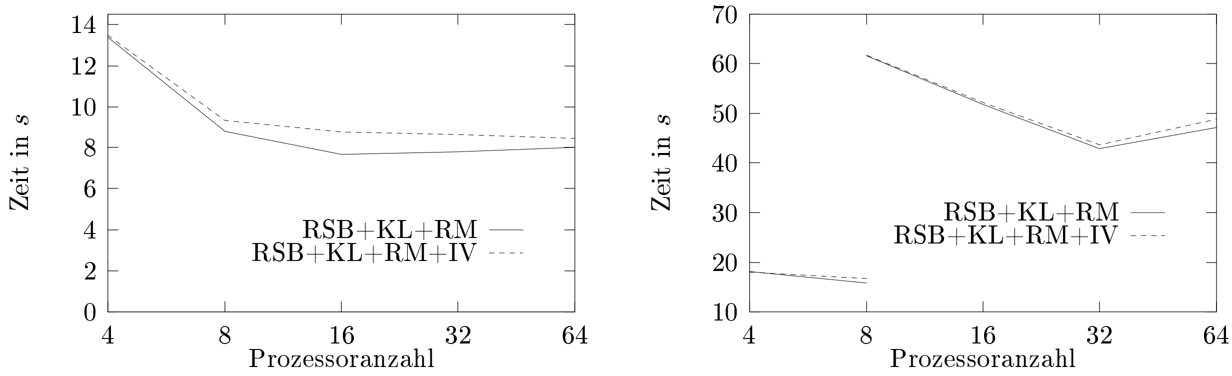
Zur Aktivierung dieser Prozedur ist der Parameter `INTERNAL_VERTICES = TRUE` im File *User_Params* zu setzen. Im Dateinamen des Partitionierungsfiles ist er laut Abbildung 4 und Tabelle 2 verschlüsselt. Für weitere Informationen siehe [11].

Die damit erzielten Ergebnisse sind in Abbildung 11 dargestellt. Die Änderung der Zahl innerer Ecken geht aus Tabelle 8 hervor. Der Erfolg dieser Form des Postprocessings ist sicherlich stark von der Implementierung der Kommunikationsroutinen abhängig und es wird offensichtlich, daß bei unserer Testanwendung *SPC-PMPo 3D* dieser Ansatz keinesfalls eine Verbesserung erbringt. Es ist aber durchaus vorstellbar, daß andere Anwendungen sich in dieser Hinsicht positiv verhalten. Eine solche Anwendung müßte in der Lage sein, alternativ



(*cube768.std*) (*spc3-123.std*)

Abbildung 10: *Vergleich der Rechenzeiten der lokal KL-verfeinerten rekursiven spektralen Bi-, Quadri- und Oktasektion mit spezieller Anpassung der Verteilung an das Hypercubemodell für Hypercubedimension $d = 2 \dots 6$.*



cube768.std *spc3-123.std*

Abbildung 11: *Rechenzeiten für die lokal KL-verfeinerte rekursive Spektralbisektion mit und ohne die Postprocessing-Prozedur zur Erhöhung der Anzahl innerer Ecken.*

zu entscheiden, ob kommuniziert wird, weil gerade der Zielprozessor bereit ist und die Links frei sind, oder besser an inneren Knoten gerechnet wird, für deren Bearbeitung keine Daten von anderen Prozessoren benötigt werden. Auf diese Art und Weise könnten Wartezeiten sinnvoll genutzt werden, um die Gesamtperformance zu steigern.

Globale Nachbesserung der Partitionierung Während der rekursiven Teilung geht von Level zu Level Information verloren. Beispielsweise wird eine lokale Nachbesserung nur auf einem Bruchteil der verbundenen Partitionen ausgeführt. Wenn gewünscht, kann **Chaco** aber auch eine globale Nachbesserung zwischen *allen* Paaren von Partitionen ausführen.

Dazu wird zunächst das Gewicht der Schnittkanten zwischen jedem Partitionspaar bestimmt. Dann wird zwischen jedem Paar Kernighan-Lin Nachbesserung durchgeführt, in der Reihenfolge von dem Paar mit der größten Grenze zu dem mit der Kleinsten. Das Ergebnis einer solchen globalen Nachbesserung soll Abbildung 12 verdeutlichen.

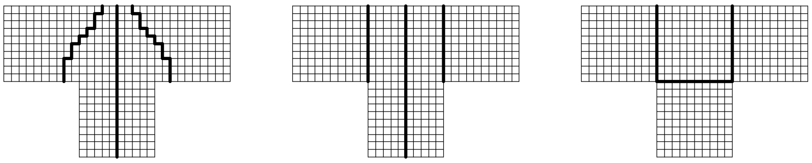
Die Anzahl der Iterationsschritte dieses Algorithmus wird durch den Wert des Parameter `REFINE_PARTITION` im File *User_ Params* bestimmt und ist im Dateinamen des Partitionierungsfiles nach Abbildung 4 vermerkt. Für weitere Informationen siehe [11].

Die Effizienz dieses Postprocessings in der Praxis ist in Abbildung 13 dargestellt. Wie aus der Abbildung ersichtlich wird, ist der Erfolg eher gering. Die Verbesserung der Rechenzeit ist kaum nachvollziehbar. Da aber der Rechenaufwand für die Postprocessing-Prozedur, gemessen am Gesamtaufwand, recht gering ist und keinesfalls mit einer Verschlechterung der

cube768.std			*spc3-123.std*		
Proz.	RSB+KL+RM	RSB+KL+RM+IV	Proz.	RSB+KL+RM	RSB+KL+RM+IV
4	628	631	4	1303	1305
8	522	525	8	1236	1238
16	342	360	16	1100	1098
32	213	245	32	903	908
64	115	148	64	659	679

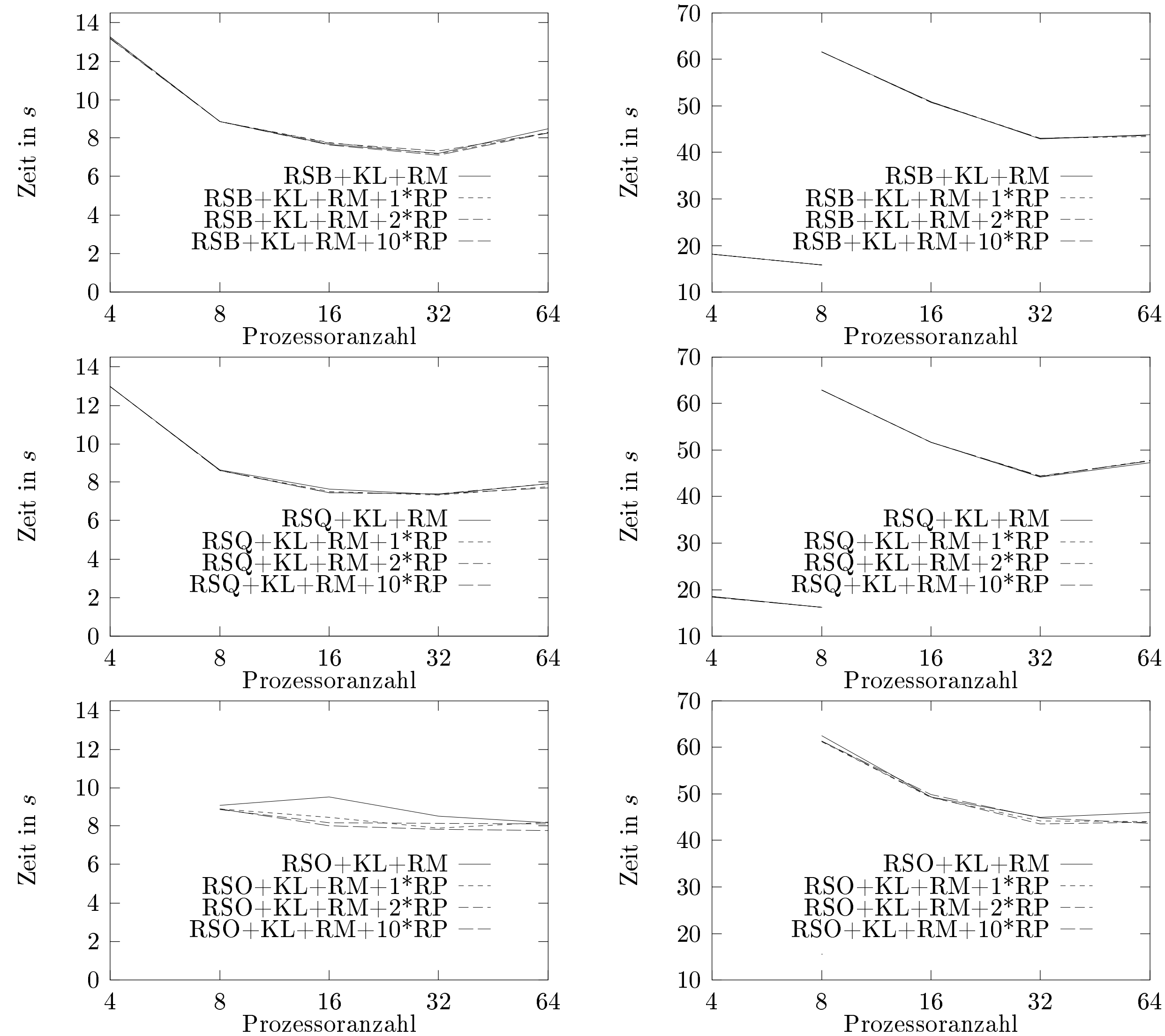
Tabelle 8: *Anzahl der inneren Ecken bei Partitionierung mit lokal KL-verfeinerter rekursiver Spektralbisektion, angepaßt an das Hypercubemodell, mit und ohne den Versuch im Postprocessing die Anzahl der inneren Ecken zu erhöhen.*

Partitionierung zu rechnen ist, ist es dennoch sinnvoll, sie zu nutzen. Die Anzahl der Iterationsschritte sollte dabei abhängig von der Anzahl der Netzelemente gewählt werden. Bei bis zu 1000 Elementen genügen sicher 2–3 Iterationen, bei mehr Elementen kann eine Erhöhung der Iterationszahl noch Verbesserungen bringen. Es ist aber fraglich, ob Iterationszahlen größer als 10 sinnvoll sind.



(a) (b) (c)

Abbildung 12: *Effizienz der Anwendung globaler KL-Nachbesserung. (a) wie bereits in Abbildung 2-(a) gezeigt, ergibt die rekursive Spektralbisektion kein optimales Ergebnis. Die Zahl der Edge Cuts ist hier 50. (b) Rekursive Spektralbisektion mit einem Iterationsschritt globaler Nachbesserung. Das Ergebnis ist mit 40 Edge Cuts deutlich besser. (c) Rekursive Spektralbisektion mit zwei Iterationsschritten globaler Nachbesserung. Hier wird die optimale Partitionierung mit 30 Edge Cuts erreicht.*



(*cube768.std*) (*spc3-123.std*)

Abbildung 13: *Rechenzeiten der lokal KL-verfeinerten rekursiven spektralen Bi-, Quadri- und Oktasektion mit und ohne spezielle Anpassung der Verteilung an das Hypercubemodell für Hypercubedimension* $d = 2 \dots 6$.

3 Terminal Propagation

Bemerkung: Die in diesem Abschnitt dargelegten theoretischen Sachverhalte sind im wesentlichen [14] und [4] entnommen.

3.1 Motivation

Wie wir bereits in Abschnitt 2 gesehen haben, liefert die rekursive Spektralbisektion recht gute Ergebnisse bei der Partitionierung, die durch diverse Postprocessingtechniken noch verbessert werden können. Dennoch ist es bisher nicht möglich gewesen, die Partitionierung und die Zuordnung auf die Prozessoren gemeinsam durchzuführen, was für eine optimale Anpassung der Verteilung an die Rechnerarchitektur wünschenswert wäre. Abbildung 14 soll dies verdeutlichen.

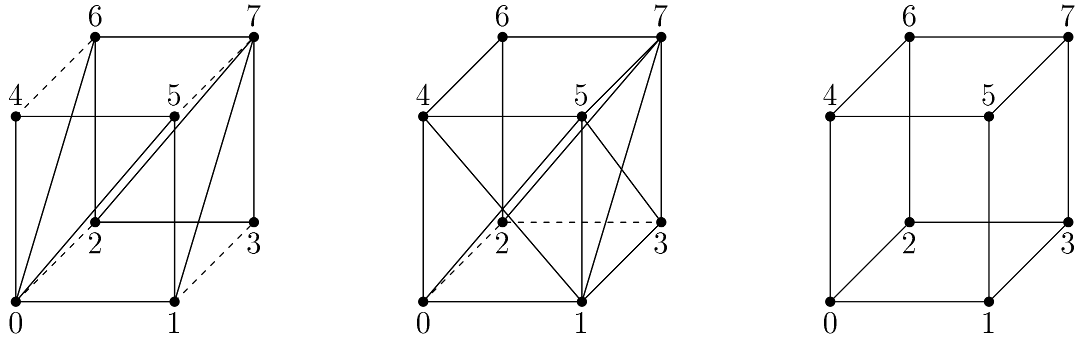


Abbildung 14: *Interprozessorzusammenhänge bei Verteilung von* cube442.std *auf 8 Prozessoren. Links: reine RSB, die vorhandenen Links werden nur schlecht genutzt. Mitte: RSB mit KL Nachbesserung und Topologieanpassung im Postprocessing, die Links werden deutlich besser genutzt, doch ein optimales Ergebnis ist im Postprocessing nicht mehr zu erreichen. Rechts: RSB mit Terminal Propagation, die Verteilung wurde optimal an die Topologie angepaßt.*

Die rekursive Spektralbisektion (links) erreicht keine optimale Partitionierung und die Beziehungen zwischen den Partitionen passen schlecht zur Prozessortopologie. Selbst bei einer möglichen optimalen Partitionierung mit rekursiver Spektralbisektion und lokaler Nachbesserung durch KL (Mitte) ist im Postprocessing keine optimale Anpassung an die Topologie mehr möglich. Erst die Kombination von Bisektion und Mapping bei der Terminal Propagation erreicht das Optimum.

Die erzielten Qualitäten der Verteilungen spiegeln sich auch in den gemessenen Metriken wieder, siehe Tabelle 9.

	RSB	RSB+KL+RM	RSB+TP
Set Size	32	32	32
Edge Cuts	40	32	32
Hypercube Hops	56	38	32
Boundary Vertices	64	64	64
Boundary Vertex Hops	88	76	64
Adjacent Sets	24	30	20

Tabelle 9: *Gemessene Metriken für verschiedene Verteilungen von* cube442.std *auf 8 Prozessoren. Die Werte beziehen sich auf den Dualgraphen.*

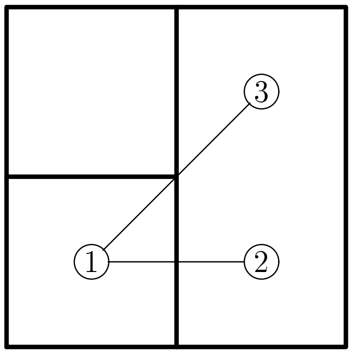


Abbildung 15: *Grundlegende Motivation für die Terminal Propagation im Kontext des Schaltungslayouts. Wir sind dabei, die rechte Hälfte des Chips zu partitionieren und unter dem Gesichtspunkt der Minimierung der Leitungslänge wäre es wünschenswert das Ende der Kante ausgehend von Punkt 1 in der unteren Hälfte an Punkt 2 statt in der oberen Hälfte an Punkt 3 anzusiedeln. Gewöhnliche Partitionierungsschemata können diese beiden Fälle nicht unterscheiden.*

Wie klar ersichtlich wird, liefert die rekursive Spektralbisektion mit Terminal Propagation sehr gute Ergebnisse. Daher werden wir uns im weiteren etwas intensiver damit beschäftigen. Nachdem wir uns in Abschnitt 3.2 mit der zugrundeliegenden Idee befassen, werden wir im Abschnitt 3.3 das Graphenmodell aus 2.2 erweitern und dann in Abschnitt 3.4 die Besonderheiten der Lösung des erweiterten Problems erörtern. Abschließend werden im Abschnitt 3.5 Ergebnisse praxisnaher Tests angegeben.

3.2 Die Idee

Die meisten heute verwendeten Algorithmen zur Graphenpartitionierung wurden ursprünglich von Forschern aus der Sparte des Schaltungslayouts entwickelt. Bei der Platzierung von Schaltungselementen auf einem Chip ist es wichtig, die Länge der Leiterbahnen so kurz wie möglich zu halten. Dies spart Platz auf dem Chip und hält die Übermittelungsverzögerungen gering. Eine wichtige Methode zur Platzierung von Schaltungselementen ist verbunden mit der Partitionierung von Graphen, die die Schaltung beschreiben. Typischerweise wird die Schaltung zunächst in zwei etwa gleichgroße Teile mit ein paar Leitungsbahnen dazwischen geteilt. Die Chipfläche wird auf die gleiche Weise geteilt und die beiden Schaltungshälften werden auf den beiden Chiphälften plaziert. Dies wird nun rekursiv wiederholt bei jedem Teilproblem. Da nur wenige Leitungen zwischen den Hälften liegen, werden die meisten Verbindungen lokal und damit kurz gehalten.

Dieser simple Ansatz hat eine wichtige Folgerung. Da die beiden Hälften vollständig voneinander getrennt werden, gibt es im weiteren Verlauf keinen Mechanismus mehr, um die Länge der Leiterbahnen zwischen ihnen zu minimieren. Als Beispiel dafür sei auf Abbildung 15 verwiesen. Im ersten Schritt teilen wir die Schaltung in zwei Hälften und weisen eine Hälfte dem linken Teil des Chips und die andere dem rechten Teil des Chips zu. Anschließend teilen wir die linke Hälfte und weisen die resultierenden Teile dem oberen und unteren Quadranten der linken Hälfte zu. Es sei nun eine Leitung bei der ersten Teilung geschnitten worden und sei nun ihr linker Endpunkt im linken unteren Teil (z.B. Punkt 1). Es ist nun klar, daß es im Rahmen einer Minimierung der Leitungslänge besser wäre, wenn der rechte Endpunkt unserer Leitung den rechten unteren Teil (Punkt 2) statt dem oberen rechten Teil (Punkt 3) zugewiesen würde, doch einfache Partitionierungsalgorithmen sind nicht in der Lage, dies zu beachten.

Diesem Umstand ist es zuzuschreiben, daß Dunlop und Kernighan in [6] das Konzept der Terminal Propagation erarbeiteten. Ihr Ansatz ist unmittelbar mit dem populären Partitionierungsalgorithmus von Kernighan und Lin [15] verbunden, aber wir wollen die zugrundeliegende Idee hier in einer etwas allgemeineren Form betrachten, die uns auch ihre Anwendung in Verbindung mir anderen rekursiven Algorithmen erlaubt.

Die grundlegende Idee der Terminal Propagation ist es, jeder Ecke des zu teilenden Untergraphen eine Zahl zuzuordnen, die die *Bevorzugung* des oberen statt des unteren Quadranten widerspiegelt. Zu beachten ist, daß diese *Vorzugszahl* nur eine Funktion der Kanten ist, die die Ecke mit Ecken verbinden die nicht zum aktuell zuteilenden Untergraphen gehören. Der Name *Terminal Propagation* kommt vom Schaltungslayout, wo zusätzliche Bedingungen dieser Art aus Verbindungsleitungen zwischen Chip und Umgebung erwachsen. Diese *Terminals* implizieren Vorzüge gewisser Partitionierungen und die Vorzüge werden im Laufe der Rekursion *propagiert.*

Ein analoges Problem ergibt sich beim parallelen Rechnen. Der Graph beschreibe nun unser FEM-Netz, das wir auf die Prozessoren verteilen wollen. Der normale Weg wäre es, diesen Graphen zu zweiteilen und die Teilgraphen jeweils einer Hälfte der Prozessoren zuzuordnen. Diesen Vorgang können wir nun rekursiv anwenden, bis jedem Prozessor ein spezieller Teil des Graphen zugeordnet ist. Bedauerlicherweise beachtet dieser Ansatz keinerlei architektonisch bedingte Entfernung zwischen den Prozessoren. Da die Kanten zwischen den Teilproblemen im Laufe der Rekursion ignoriert werden, müssen Daten möglicherweise über viele Links wandern. Die Zuordnung auf die Prozessoren ist also, wie im letzten Abschnitt bereits bemängelt, unabhängig von der Partitionierung. Dies ist im wesentlichen dasselbe Problem, das beim Schaltungslayout auftritt, welches zur Anwendung der Terminal Propagation motivierte.

3.3 Erweiterung des Graphenmodells

Im Kontext des parallelen Rechnens brauchen wir eine geringfügig andere Interpretation der Terminal Propagation, die in Abbildung 16 dargestellt ist.

Abbildung 16: *Die Idee der Terminal Propagation im Kontext des parallelen Rechnens. Hier definieren wir die Teilung zwischen den Prozessoren 10 und 11. Die linke Zuordnung ist zu bevorzugen, da sie Kommunikationskosten von 4 Hops bringt, wären die rechte Zuordnung 5 Hops aufweist.*

Die Quadranten repräsentieren hier Prozessoren oder Prozessorgruppen eines (als Beispiel) Hypercubes oder einer 2D-Netz Architektur. Die (Gruppen von) Prozessoren können durch einen 2 Bit-Code identifiziert werden und die Anzahl der Links zwischen zwei Prozessoren ergibt sich aus der Zahl der Bits um die sich die Nummern unterscheiden. Der Graph sei nun bereits einmal geteilt, die Teilgraphen der linken und rechten Hälfte der Rechners zugeordnet und der linke Teilgraph ist nochmals geteilt und dem linken oberen und unteren Quadranten zugeordnet. Wenn wir den rechten Teilgraphen auf die Prozessoren 10 und 11 verteilen, wollen wir erreichen, daß Daten möglichst kurze Wege haben. Die Zuordnung in der linken Abbildung ist besser, da die totale Datenweglänge $2 \times 1 + 2 = 4$ ist, während die Zuordnung der rechten Abbildung Kosten von $2 \times 2 + 1 = 5$ bedeutet.

Diese Argumentation wollen wir nun in einer Bevorzugungszahl zum Ausdruck bringen. Die uns interessierende Ecke hat zwei Nachbarn im unteren linken Quadranten und einen im oberen linken Quadranten. Daher ergibt sich ein Vorzug für den unteren rechten Quadranten von 1. Wenn die Kanten zu diesen externen Ecken nun noch Gewichte haben, dann ist diese *Vorzugszahl* noch entsprechend zu skalieren. Statt nun wie bisher nur die Anzahl der Kanten zwischen oberen und unteren rechten Quadranten bei der Partitionierung zu minimieren, minimieren wir jetzt die Summe über die geschnittenen Kanten und die nichterfüllten Bevorzugungen.

Diese Zielfunktion können wir nun auch algebraisch formulieren. Wenn wir an einem Subproblem arbeiten, konstruieren wir die Vorzugszahl für jede Ecke auf der Basis der Kanten, die diese Ecke mit anderen Subproblemen verbinden. Wir wollen nun entscheiden, ob wir die Ecke v_i der einen oder der anderen Partition zuordnen, wobei v_i mit v_j verbunden sei und v_j nicht zum aktuellen Subproblem gehöre. Die Kante e_{ij} trägt einen Wert von $w_e(e_{ij})(D_1 - D_0)$ zur Vorzugszahl bei, wobei $w_e(e_{ij})$ das Gewicht der Kante ist, D_0 die architektonische Entfernung zwischen v_i und v_j, falls $v_i \in \mathcal{V}_0$ und D_1 die architektonische Entfernung zwischen v_i und v_j, wenn $v_i \in \mathcal{V}_1$. Es ist nun noch möglich den Vektor der Vorzugszahlen so zu skalieren, daß in unsere Metrik auch eine Abschätzung der Wichtigkeit von Datenlokalität gegenüber dem Kommunikationsvolumen eingeht, siehe dazu Abschnitt 3.5.2.

Mit dieser Vereinbarung können wir nun das Problem formal Aufschreiben: Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein Graph mit Ecken $v \in \mathcal{V}$ und Kanten $e_{ij} \in \mathcal{E}$. Wir lassen sowohl für Ecken als auch Kanten positive Gewichte $w_v(v_i)$ und $w_e(e_{ij})$ zu. Wir werden n (und m) benutzen, um die Anzahl der Ecken (und Kanten) des Graphen anzugeben. Wir wollen nun $\mathcal{V}$ in zwei Teilgraphen $\mathcal{V}_0$ und $\mathcal{V}_1$ partitionieren, und wir haben einen Bevorzugungsvektor p dafür, daß die Ecken von $\mathcal{V} \quad \mathcal{V}_0$ zugeordnet werden. Die mit einer Partitionierung verbundenen Kosten haben nun zwei Komponenten. Erstens, jede Kante e_{ij} zwischen $\mathcal{V}_0$ und $\mathcal{V}_1$ trägt ihr Gewicht $w_e(e_{ij})$ bei. Zweitens, für jede Ecke in $\mathcal{V}_0$ mit einem negativen Vorzug addieren wir den Betrag dieses Vorzuges zu den Kosten und das gleiche tun wir für jede Ecke in $\mathcal{V}_1$ mit einem positiven Vorzug. Unser Ziel ist es, eine Partitionierung zu finden, mit $|\mathcal{V}_0| \approx |\mathcal{V}_1|$ unter Minimierung der Kostenfunktion.

Leider ist dies wieder ein NP-vollständiges Problem, so daß ein allgemeingültiger und effizienter Algorithmus auszuschließen ist. Im nächsten Abschnitt werden wir nun beschreiben, wie dieses Problem mittels rekursiver Spektralbisektion behandelt werden kann.

3.4 Spektrale Bisektion mit Terminal Propagation

Im Gegensatz zu Abschnitt 2.3 wollen wir hier ein allgemeineres Graphenmodell zugrundelegen, bei dem wir sowohl für Ecken als auch Kanten positive reelle Gewichte zulassen. Dies findet seinen Niederschlag in einer etwas anderen Definition der Laplace-Matrix L. Diese sei im folgenden definiert durch

$$L(i,j) = \begin{cases} \sum_{e_{ik} \in \mathcal{E}} w_e(e_{ik}) & \text{falls} \quad i = j, \\ -w_e(e_{ij}) & \text{falls} \quad e_{ij} \in \mathcal{E}, \\ 0 & \text{sonst.} \end{cases}$$

Falls alle w_e gleich 1 sind, ist dies mit der vorherigen Definition äquivalent. Auch an den Eigenschaften der Laplace-Matrix ändert sich nichts. Wie in Abschnitt 2.3 werden wir nun ein diskretes Optimierungsproblem konstruieren und dann versuchen, dies mittels Approximation durch ein stetiges Problem zu lösen.

In der Standardherleitung begannen wir mit der algebraischen Formulierung (1) des exakten Partitionierungsproblems. Nun müssen wir diese Zielfunktion erweitern, um die Terminal Propagation einzuarbeiten. Unter dem Gesichtspunkt der Terminal Propagation müssen wir nun unterscheiden zwischen dem aktuell zu partitionierenden Untergraphen und dem Rest des Graphen, der mögliche Bedingungen an unsere Partitionierung stellt. Wenn p_i die Vorzugszahl der Ecke v_i ist und wir im Fall $v_i \in \mathcal{V}_0$ $p_i = +1$ setzen und sonst -1, dann sieht unser neues zu lösendes Problem wie folgt aus:

$$\begin{array}{ll} \text{Minimiere } \frac{1}{4} \sum\limits_{e_{ij} \in \underline{\mathcal{E}}} w_e(e_{ij})(x_i - x_j)^2 - \frac{1}{2} \sum\limits_{i \in \underline{\mathcal{V}}} p_i x_i & \\ \text{unter den Bedingungen, daß} & \\ \text{(a)} \qquad \sum\limits_{i \in \underline{\mathcal{V}}} w_v(v_i) x_i \approx 0 & \\ \text{(b)} \qquad x_i = \pm 1. & \end{array} \tag{5}$$

Mit der leicht nachzuprüfenden Beziehung

$$\sum_{e_{ij} \in \mathcal{E}} w_e(e_{ij})(x_i - x_j)^2 = x^T L x$$

und derselben Approximation wie im Abschnitt 2.3 erhalten wir

$$\begin{array}{l} \text{Minimiere } \frac{1}{4} x^T L x - \frac{1}{2} p^T x \\ \text{unter den Bedingungen, daß} \\ \text{(a)} \qquad w_v^T x = 0 \\ \text{(b)} \qquad x^T x = n. \end{array} \tag{6}$$

Mit einer Variablentransformation können wir (6) in ein äquivalentes System überführen, das letztlich auf ein 'erweitertes' Eigenwertproblem führen wird. Zunächst definieren wir $s_i = \sqrt{w_v(v_i)}$ und $t_i = 1/s_i$. Sei nun $y = \text{diag}(s_i)x$ und sei $B = \text{diag}(t_i)^T L \text{diag}(t_i)$. Der neue Vektor s übernimmt die Rolle von w_v, ist aber gleichzeitig ein Eigenvektor von B. Da die Komponenten von x Approximationen an ± 1 sind, ergibt sich die Normierung von y aus $y^T y = \sum_i w_v(v_i)$, die wir mit ω_v bezeichnen. Außerdem sei $h = \text{diag}(t_i)p$ und wir multiplizieren die Zielfunktion mit 4. Damit ergibt sich

$$\begin{array}{l} \text{Minimiere } y^T B y - 2h^T y \\ \text{unter den Bedingungen, daß} \\ \text{(a)} \qquad s^T y = 0 \\ \text{(b)} \qquad y^T y = \omega_v. \end{array} \tag{7}$$

Wir wenden die Lagrangemethode an. Dazu führen wir die Lagrangeschen Multiplikatoren η und μ ein und suchen nach stationären Punkten der Funktion

$$F(y, \eta, \mu) = y^T B y - 2h^T y + \eta(s^T y) + \mu(\omega_v - y^T y). \tag{8}$$

Die partiellen Ableitungen von F nach η und μ ergeben die beiden Zwangsbedingungen. Für die Ableitung nach den Komponenten von y erhalten wir

$$2By - 2h + \eta s - 2\mu y = 0. \tag{9}$$

Wir können η berechnen, indem wir (9) mit s^T von links durchmultiplizieren. Da s orthogonal zu y ist und s ein Eigenvektor von B zum Eigenwert 0 ist, erhalten wir $\eta = 2s^T h/\omega_v$. Wir definieren nun

$$g = h - \frac{s^T h}{\omega_v} s, \tag{10}$$

was es uns erlaubt, (9) zu schreiben als

$$By = \mu y + g. \tag{11}$$

Dieses 'erweiterte' Eigenwertproblem muß nun unter Beachtung der Zwangsbedingungen aus (7) gelöst werden. Im allgemeinen hat dieses Problem mehrere Lösungen, aber Van Driessche und Roose haben in [5] gezeigt, daß die Lösung, welche die Zielfunktion minimiert, immer der Vektor y in Verbindung mit dem kleinstmöglichen Wert von μ ist. Ist die Lösung von (11) einmal bestimmt, wird sie zurücktransformiert zu einer Lösung von (6) von der aus dann eine naheliegende diskrete Lösung gefunden werden kann.

Auf die Lösung des 'erweiterten' Eigenwertproblems wollen wir hier nicht weiter eingehen. Dazu sei hier auf die weitergehenden Ausführungen in [14] oder [5] verwiesen.

3.5 Praktische Tests

3.5.1 Rekursive Spektralbisektion mit Terminal Propagation

Wir wollen nun untersuchen, ob sich die Erweiterung unserer Theorie auch in der Praxis als gewinnbringend erweist. Dazu vergleichen wir die rekursive Spektralbisektion mit Terminal Propagation mit dem bisher besten Verfahren. Das Ergebnis ist in Abbildung 17 dargestellt.

Wie zu erkennen ist, bringt die bloße Anwendung der Terminal Propagation bessere Ergebnisse als alle bisher betrachteten Algorithmen und Postprocessing-Routinen. Dieses Ergebnis wird auch durch die gemessenen Metriken in Tabelle 10 verdeutlicht.

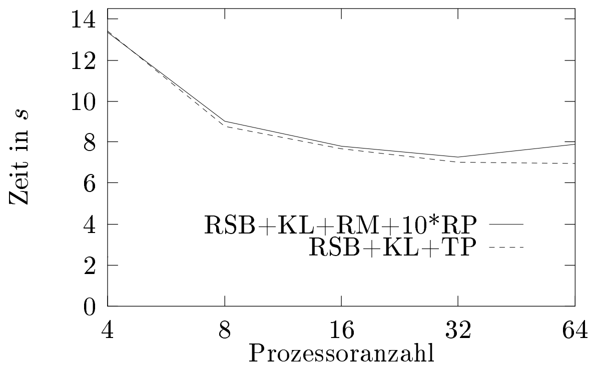
cube768.std		
	RSB+KL+RM+10*RP	RSB+KL+TP
Edge Cuts	501	555
Hypercube Hops	873	661
Boundary Vertices	966	1048
Boundary Vertex Hops	1695	1259
Adjacent Sets	488	390
spc3-123.std		
	RSB+KL+RM+10*RP	RSB+KL+TP
Edge Cuts	454	508
Hypercube Hops	625	544
Boundary Vertices	849	943
Boundary Vertex Hops	1174	1015
Adjacent Sets	246	214

Tabelle 10: *Vergleich der Metriken der rekursiven Spektralbisektion mit lokaler KL-Nachbesserung,* `REFINE_MAP` *und* `REFINE_PARTITION=10` *bzw. Terminal Propagation für eine Verteilung auf 64 Prozessoren.*

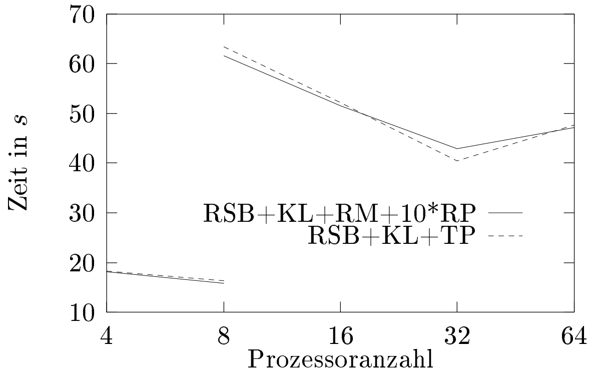
Im nächsten Abschnitt wollen wir jetzt noch untersuchen, ob sich dieses Ergebnis durch eine geschickte Skalierung des Vorzugszahlenvektors weiter verbessern läßt.

3.5.2 Variation der Cut zu Hop Kosten

Wie bereits in Abschnitt 3.3 erwähnt, bietet sich die Möglichkeit an, den Vektor der Vorzugszahlen zu skalieren, um auf die Partitionierung einzuwirken. Diese Skalierung wird in **Chaco** durch den Parameter `CUT_TO_HOP_COST` vorgenommen. Standardmäßig ist dieser



(*cube768.std*)

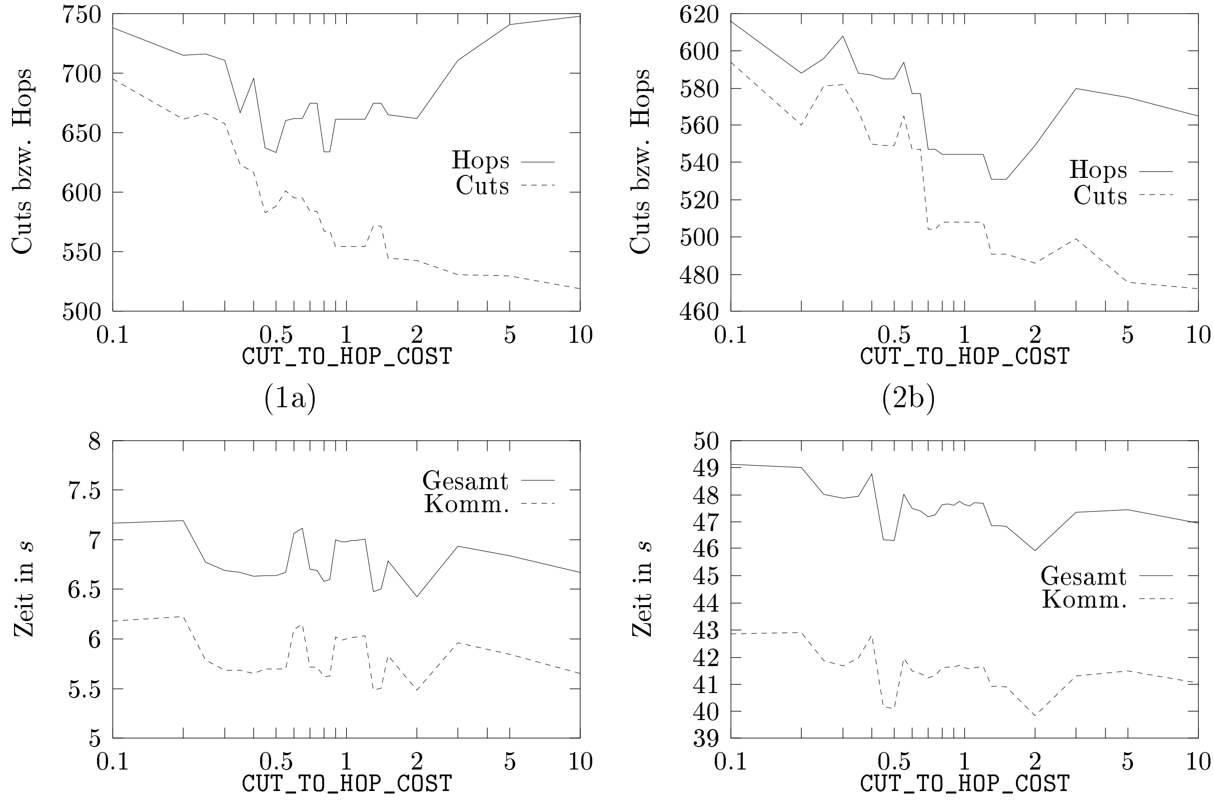


(*spc3-123.std*)

Abbildung 17: *Vergleich der Rechenzeiten der Verteilung mittels rekursiver Spektralbisektion und Terminal Propagation und der mittels rekursiver Spektralbisektion* `REFINE_MAP` *und* `REFINE_PARTITION` *mit 10 Iterationen.*

Parameter 1, er kann aber auch mit einer reellen Zahl belegt werden, um die Partitionierung zugunsten der Edge Cuts oder der Hypercube Hops zu beeinflussen. Abbildung 18 zeigt die Ergebnisse.

Es ist sehr gut zu sehen, daß lokale Minima in der Hypercube-Hop-Kurve auch zu lokalen Minima in der Rechenzeit führen. Es wird jedoch klar, daß diese Form des Tunings sehr empfindlich ist und der optimale Parameter offenbar netzspezifisch ist. Das heißt, es müßte für jedes Netz eine Testserie gerechnet werden, um das beste `CUT_TO_HOP_COST`-Verhältnis zu bestimmen. Es ist jedoch fraglich, ob dieser Aufwand, bei der doch recht geringen Verbesserung gegenüber dem Defaultwert 1, gerechtfertigt wäre.



(2a) (2b)

Abbildung 18: *Auswirkungen des Parameters* `CUT_TO_HOP_COST`*. Die Bilder 1a und 1b zeigen die Edge Cuts und Hypercube Hops für (a) cube768.std und (b) spc3-123.spc als Funktion des Parameters (Verteilung auf 64 Prozessoren). Die Bilder 2a und 2b zeigen die resultierenden Kommunikations- und Rechenzeiten.*

4 Zusammenfassung

Bevor wir nun die Ergebnisse dieser Arbeit zusammenfassen, sind noch zwei wichtige Dinge anzumerken.

Die gemessenen Zeiten können allenfalls als Tendenz bzw. Zirkawert der Rechenzeit angesehen werden. Eine Messung der Rechenzeit bis auf die 100stel Sekunde genau ist illusorisch und in der Praxis nicht sinnvoll. Messungen mit ein und derselben Verteilung können um mehrere zehntel Sekunden variieren, da die Anwendung nicht als Stand-Alone-Version arbeitet, sondern auf dem Parix-System aufsetzt, welches selbst mehr oder weniger Rechenzeit benötigt. Deshalb sind sehr gering Unterschiede von weniger als 1% zwischen verschiedenen Verteilungen in diesem Kontext zu vernachlässigen.

Alle von mir getesteten Verteilungen wurden auf meinem eigenen Computer, basierend auf einer MC68040 CPU, berechnet. Wie aber Vergleiche mit anderen Systemen, wie Sun, HP oder Pentium-PCs gezeigt haben, sind die von **Chaco** gelieferten Ergebnisse vom System abhängig. Eine Rückfrage bei Bruce Hendrickson bestätigte, daß die Ergebnisse von **Chaco** durch maschinenabhängige Werte, wie dem Maschinen-eps, beeinflußt werden. Eine weitere wichtige Rolle in den Algorithmen von **Chaco** spielt der Zufallszahlengenerator, der auch maschinenabhängig arbeitet. Aufgrund dieser Tatsachen ist es nicht ausgeschlossen, daß einige der hier vorgestellten und getesteten Algorithmen auf anderen Maschinen abweichende Ergebnisse erzielen. Diese mögen besser, aber auch schlechter sein. Ein individueller Test auf zumindest zwei verschiedenen Maschinen und Vergleich der Ergebnisse anhand der resultierenden Metriken mag daher durchaus sinnvoll sein.

Ungeachtet dieser Einschränkungen lassen sich doch ein paar Aussagen aus den Tests ableiten. Zunächst läßt sich zweifelsfrei festhalten, daß die betrachteten spektralen Algorithmen im allgemeinen deutlich bessere Partitionierungen versprechen, als zufällige oder lineare Verteilungen (↗ 2.5.3). Desweiteren wurde deutlich, daß sich die lokale Nachbesserung mittels Kernighan-Lin Algorithmus in jedem Fall rentiert (↗ 2.5.4). Die dadurch erreichte Verkürzung der Rechenzeit lag durchschnittlich bei ca. 18%.

Die Weiterentwicklungen der spektralen Bisektion, die spektrale Quadri- und Oktasektion konnten den Erwartungen in den Tests nicht gerecht werden und stellen hier keine Verbesserung der Bisektion dar (↗ 2.5.5). Die rekursive Spektralquadrisektion konnte nur eine sehr geringe Verbesserung von maximal 6.5% erbringen und war teilweise sogar etwas schlechter als die Spektralbisektion. Die spektrale Oktasektion brachte im Test nahezu immer schlechtere Ergebnisse als die spektrale Bisektion. Leider konnte ich nicht feststellen, ob dies mit unseren Testnetzen zusammenhängt oder ob es sich dabei um ein generelles Problem des Algorithmus handelt. Dazu ist sicher noch anzumerken, daß beim Vergleich von spektraler Bi-, Quadri- und Oktasektion prinzipiell nach jeder Rekursion lokal KL-nachverbessert wurde, und das möglicherweise dadurch Vorteile der spektralen Quadri- und Oktasektion gegenüber der Spektralbisektion eliminiert wurden.

Als sehr nützlich erwies sich im Test auch die Anpassung der Verteilung an das Hypercubemodell (↗ 2.5.6). Dadurch konnte eine weitere durchschnittliche Verringerung der Rechenzeit von ca. 5–7% erreicht werden.

Die Anpassung erübrigt sich aber, wenn man die Terminal Propagation anwendet. Diese Erweiterung der herkömmlichen Spektralbisektion erbrachte durchweg die besten Ergebnisse (↗ 3.5.1). Gegenüber der Standardbisektion mit lokaler KL-Nachbesserung ließ sich der Rechenzeitbedarf um ca. 8–15% senken. Ein weiteres Tuning dieses Algorithmus durch die Variation des Parameters `CUT_TO_HOP_COST` ist aufwendig und vermutlich nicht den Aufwand wert. Dennoch konnte im Test dadurch eine weitere Verbesserung von bis zu 7.8% erzielt werden (↗ 3.5.2).

Die globale Nachbesserung der Partitionierung im Postprocessing kann auch zu einer

Verbesserung der Verteilung beitragen (↗ 2.5.6). Die Tests erbrachten eine Verkürzung der Rechenzeit um ca. 2%, was aber schon knapp an der Grenze der Meßgenauigkeit liegt. Da das Verfahren nicht sehr aufwendig ist, können einige Iterationsschritte dieses Postprocessings aber nie schaden.

Die Erhöhung der Anzahl der inneren Ecken hat sich im Test als eher kontraproduktiv erwiesen (↗ 2.5.6). Aber wie bereits ausgeführt, muß dies bei einem anderen Anwendungsprogramm nicht der Fall sein.

Um die Ergebnisse dieser Arbeit auf den Punkt zu bringen, läßt sich für die Berechnung kommender Partitionierungen folgende generelle Empfehlung geben: Als günstigstes Verfahren erweist sich die spektrale Bisektion mit Terminal Propagation und lokaler KL-Nachbesserung. Dazu sind 2–3 Iterationen globaler KL-Nachbesserung im Postprocessing sinnvoll. Ein dementsprechendes File *User_Params* sieht wie folgt aus:

```
OUTPUT_ASSIGN = TRUE
LANCZOS_TYPE = 1
TERM_PROP = TRUE
REFINE_PARTITION = 2
```

Dies sollte im allgemeinen sehr gute Ergebnisse liefern. Verbesserungen, die sich aus anderen Algorithmen oder weiteren Postprocessings ergeben, sind sicher netzspezifisch, so daß hier keine weitergehende generelle Empfehlung gegeben werden kann.

Die in dieser Arbeit getesteten spektralen Algorithmen stellen die Ergebnisse der heutigen Forschung auf dem Gebiet der Partitionierung dar. Der einzige, hier nicht getestete, moderne Algorithmus, ist die spektrale Quadrisektion mit Terminal Propagation, die in [4] vorgestellt und getestet wird. Diese verspricht nochmals eine Verbesserung der Partitionierung. Da dieser Algorithmus aber nicht in **Chaco** implementiert ist, konnte er hier auch nicht getestet werden. Die in den Tests erzielten Ergebnisse sind aber schon so gut, daß nur noch geringe Verbesserung zu erwarten sind.

A Zugehörige Programme

A.1 *std2graph*

Das Programm *std2graph* ist notwendig, um die FE-Netzinformationen aus den Standard-Eingabefiles von *SPC-PMPo3D* (beschrieben in [1]) für **Chaco** aufzubereiten. Es wird ein File *.graph* erzeugt, das den Eingabespezifikationen von **Chaco**, siehe [11], entspricht.

Es handelt sich um ein einfaches C-Programm, das das Eingabe- und Ausgabefile als Kommandozeilenparameter erwartet.

```
std2graph [-q] <infile> <outfile>
```

Ein normaler Lauf des Programmes sieht folgendermaßen aus:

```
bash$ std2graph amw22d.std amw22d.graph
--- This is std2graph C-Version V1.2  10.6.96 ---
Read data (96 solids with 232 faces)...
Read in ready! Scanning neighbourhood relationships...
Conversion finished! Writing file...
Ready!
bash$
```

Optional kann direkt nach dem Kommandonamen der Parameter **-q** angegeben werden, der bewirkt, daß das Programm im Quietmode arbeitet und alle Ausgaben unterdrückt werden.

Die aktuelle Version des Programmes ist V1.2, die demnächts um die Einleseroutine für Standardfiles von Dag Lohse erweitert wird.

Das Programm befindet sich stets in der aktuellsten offiziellen Version im Verzeichnis

~reichel/Chaco-2.0/tools.

A.2 *plotass*

Das Programm *plotass* soll zur Visualisierung der von **Chaco** berechneten Partitionen dienen. Dabei wurde der Einfachheit halber zur grafischen Ausgabe auf das Programm *gnuplot* zurückgegriffen. *plotass* erzeugt ein File *.plt*, das ein Skript für *gnuplot* ist, und zwei Datenfiles *.dat*.

Zum Start wird *plotass* einfach ohne weitere Parameter in einer Shell aufgerufen. Danach wird die Angabe des Standardfiles ohne Endung *.std* und des Verteilungsfiles ohne Endung *.ass* erwartet. Abschließend benötigt das Programm noch den Grundnamen für die Ausgabefiles. Nachdem das Programm die Daten bearbeitet hat, bietet es die Möglichkeit, zur besseren Erkennbarkeit die einzelnen Partitionen um einen einzugebenden Faktor nach außen zuverschieben. Ein typischer Programmlauf sieht wie folgt aus:

```
bash$ plotass
-- plotass C Version 2.01  20.3.96 --

infile1 - Standardfile, Grundname (*.std): cube768
infile2 - aus Chaco, Grundname (*.ass): cube768_32r-1--10
outfiles - fuer GNUPlot, Grundname: plot
205 Ecken werden eingelesen...
1100 Kanten werden eingelesen...
1664 Seitenflaechen werden eingelesen...
```

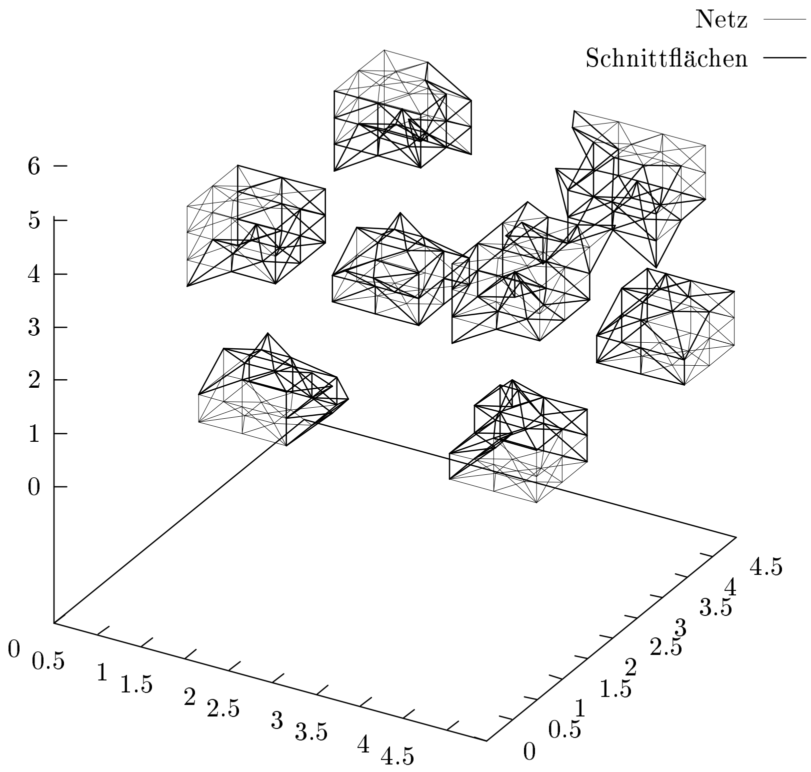


Abbildung 19: *Visualisierung von* cube768_32r-1--10.ass *mittels* plotass *und* gnuplot.

```
768 Elemente werden eingelesen...
Fertig! Jetzt Assignmentdaten einlesen und verarbeiten...
8 Partitionen erkannt!
Partitionen verschieben - Faktor (1 = keine Verschiebung) ? 3
Fertig! Schreibe files...
Fertig !
bash$
```

Die aus diesem Lauf resultierende Grafik zeigt Abbildung 19.

Die aktuelle Version des Programmes ist V2.01. Wie schon *std2graph* wird auch *plotass* demnächts um die Einleseroutine für Standardfiles von Dag Lohse erweitert.

Das Programm befindet sich stets in der aktuellsten offiziellen Version im Verzeichnis

~reichel/Chaco-2.0/tools.

B Die FE-Netze

Die Abbildungen zeigen die FE-Netze, mit denen alle Tests gerechnet wurden. Das in Abbildung 20 dargestellte Netz ist recht regelmäßig und hat die wichtige Eigenschaft, daß die Elementanzahl durch 64 teilbar ist. Dadurch ist stets eine ausgewogene Partitionierung zu erreichen. Abbildung 21 zeigt hingegen ein sehr unregelmäßiges Netz, das mit 1398 Elemente lediglich durch 2 teilbar ist, und damit große Anforderungen an den Partitionierungsalgorithmus stellt, um eine gleichmäßige Verteilung zu erzielen.

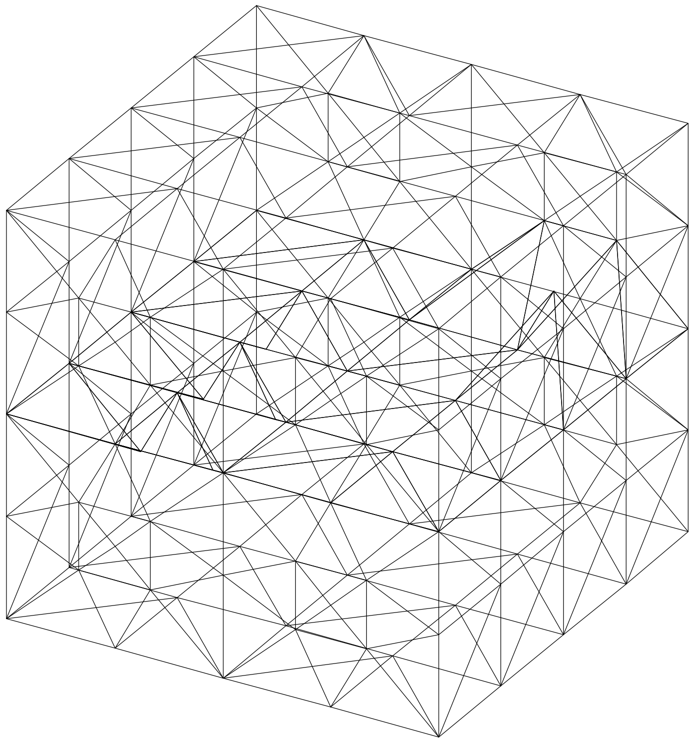


Abbildung 20: *FE-Netz cube768.std mit 768 Tetraederelementen.*

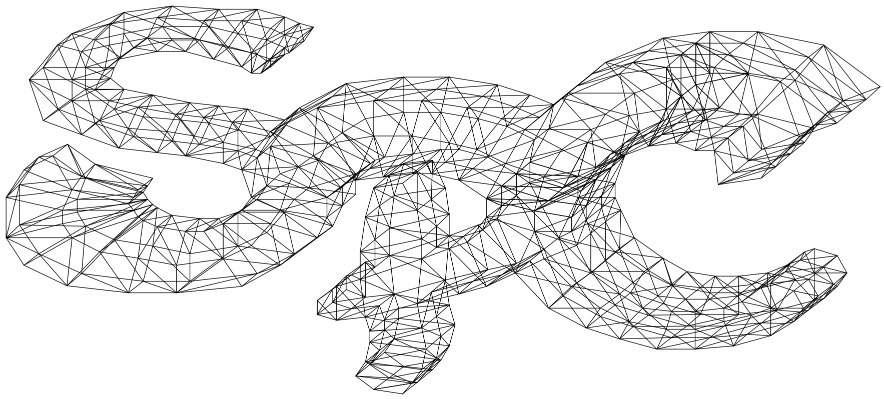


Abbildung 21: *FE-Netz spc3-123.std mit 1398 Tetraederelementen.*

Literatur

[1] Th. Apel. *SPC-PMPo 3D* — User's Manual. Preprint SPC95_33, TU Chemnitz-Zwickau, 1995.

[2] Th. Apel, F. Milde, and M. Thess. *SPC-PMPo 3D* — Programmer's Manual. Preprint SPC95_34, TU Chemnitz-Zwickau, 1995.

[3] S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, April 1994.

[4] R. Van Driessche. *Algorithms for Static and Dynamic Load Balancing on Parallel Computers.* PhD thesis, Dept. of Computer Science, Katholieke Universiteit Leuven, Belgium, November 1995. ISBN 90-5682-007-9.

[5] R. Van Driessche and D. Roose. A spectral algorithm for constrained graph partitioning I: The bisection case. TW Report 216, Dept. of Computer Science, Katholieke Universiteit Leuven, Belgium, October 1994.

[6] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Trans.*, CAD(CAD-4):92–98, 1985.

[7] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[8] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(100):619–633, 1975.

[9] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore and London, second edition, 1989.

[10] S. Hammond. *Mapping unstructured grid computations to massively parallel computers.* PhD thesis, Rensselaer Polytechnic Institute, Dept. of Computer Sience, Rensselaer, NY, 1992.

[11] B. Hendrickson and R. Leland. *The Chaco User's Guide – Version 2.0.* Sandia National Laboratories, Albuquerque, NM. SAND94-2692.

[12] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. Technical Report SAND92-1460, Sandia National Laboratories, Albuquerque, NM, September 1992.

[13] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM, October 1993.

[14] B. Hendrickson, R. Leland, and R. Van Driessche. Enhancing data locality by using terminal propagation. To be presented at the minitrack on Partitioning and Scheduling at the HICSS-29 Conference, Maui, Hawaii, January 3–6, 1996.

[15] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–308, 1970.

[16] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparce matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[17] F. Rendl and H. Wolkowicz. A projection technique for partitioning the nodes of a graph. Technical Report CORR 90-20, University of Waterloo, Faculty of Mathematics, Waterloo, Ontario, November 1990.

Other titles in the SFB393 series:

96-01 V. Mehrmann, H. Xu. Chosing poles so that the single-input pole placement problem is well-conditioned. Januar 1996.

96-02 T. Penzl. Numerical solution of generalized Lyapunov equations. January 1996.

96-03 M. Scherzer, A. Meyer. Zur Berechnung von Spannungs- und Deformationsfeldern an Interface-Ecken im nichtlinearen Deformationsbereich auf Parallelrechnern. March 1996.

96-04 Th. Frank, E. Wassen. Parallel Solution Algorithms for Lagrangian Simulation of Disperse Multiphase Flows. Proc. of 2nd Int. Symposium on Numerical Methods for Multiphase Flows, ASME Fluids Engineering Division Summer Meeting, July 7-11, 1996, San Diego, CA, USA. June 1996.

96-05 P. Benner, V. Mehrmann, H. Xu. A numerically stable, structure preserving method for computing the eigenvalues of real Hamiltonian or symplectic pencils. April 1996.

96-06 P. Benner, R. Byers, E. Barth. HAMEV and SQRED: Fortran 77 Subroutines for Computing the Eigenvalues of Hamiltonian Matrices Using Van Loans's Square Reduced Method. May 1996.

96-07 W. Rehm (Ed.). Portierbare numerische Simulation auf parallelen Architekturen. April 1996.

96-08 J. Weickert. Navier-Stokes equations as a differential-algebraic system. August 1996.

96-09 R. Byers, C. He, V. Mehrmann. Where is the nearest non-regular pencil? August 1996.

96-10 Th. Apel. A note on anisotropic interpolation error estimates for isoparametric quadrilateral finite elements. October 1996.

96-11 Th. Apel, G. Lube. Anisotropic mesh refinement for singularly perturbed reaction diffusion problems. August 1996.

96-12 B. Heise, M. Jung. Scalability, efficiency, and robustness of parallel multilevel solvers for nonlinear equations. September 1996.

96-13 F. Milde, R. A. Römer, M. Schreiber. Multifractal analysis of the metal-insulator transition in anisotropic systems. October 1996.

96-14 R. Schneider, P. L. Levin, M. Spasojević. Multiscale compression of BEM equations for electrostatic systems. October 1996.

96-15 M. Spasojević, R. Schneider, P. L. Levin. On the creation of sparse Boundary Element matrices for two dimensional electrostatics problems using the orthogonal Haar wavelet. October 1996.

96-16 S. Dahlke, W. Dahmen, R. Hochmuth, R. Schneider. Stable multiscale bases and local error estimation for elliptic problems. October 1996.

96-17 B. H. Kleemann, A. Rathsfeld, R. Schneider. Multiscale methods for Boundary Integral Equations and their application to boundary value problems in scattering theory and geodesy. October 1996.

The complete list of current and former preprints is available via
`http://www.tu-chemnitz.de/~pester/sfb/sfb96pr.html`.