Konvergente lokale Lastverteilungsverfahren und ihre Modellierung durch Zellularautomaten

Peter Sanders, Thomas Worsch
Fakultät für Informatik, LIIN
Universität Karlsruhe, 76128 Karlsruhe
E-mail: {sanders,worsch}@ira.uka.de

Zusammenfassung

Es wird eine Familie von einfachen lokal arbeitenden Lastverteilungsverfahren für d-dimensionale Gitter vorgestellt, die sich im wesentlichen als Zellularautomatenalgorithmen auffassen lassen. Die Algorithmen haben asymptotisch optimale Laufzeit und sind deutlich effizienter als die in der Literatur bisher vorherrschenden auf dem Diffusionsprinzip basierenden Algorithmen. Eine Variante ist das erste effiziente lokale Verfahren, das auch im mehrdimensionalen Fall vollständigen Lastausgleich ermöglicht.

1 Einführung

Wir stellen hier eine Familie von einfachen Lastverteilungsalgorithmen für d-dimensionale Gitter vor. Die Algorithmen benutzen nur lokale Information und lassen sich gut für kontinuierlich arbeitende dynamische Lastverteilungsprobleme verwenden. Diese Einführung fährt mit einer Präzisierung der Modellbildung fort, formuliert dann den Basisalgorithmus und ordnet ihn schließlich in das Spektrum existierender Arbeiten ein.

Im zweiten Abschnitt wird dann der Basisalgorithmus im eindimensionalen Fall untersucht. Hier läßt sich rigoros zeigen, daß der Algorithmus in diesem Fall sehr effizient ist. Im dritten Abschnitt beschäftigen wir uns mit Lastausgleichsverfahren für höhere Dimensionen d, insbesondere für $d = 2$. Hier muß der Basisalgorithmus modifiziert werden, um eine vollständige Balancierung zu ermöglichen. Auch die Analyse erweist sich als deutlich schwieriger und wird deshalb durch Simulationen ergänzt. Am Ende steht ein Ausblick auf Fragestellungen, die derzeit noch untersucht werden.

1.1 Modellbildung

Gegeben seien n^d durch ein d-dimensionales Gitter (d konstant) verbundene synchron arbeitende Prozessoren (PEs).[1] Die Last wird durch gleichgroße Lastpakete modelliert. Während einer Zeiteinheit kann ein PE genau ein Lastpaket bewegen, d.h. entweder zu einem benachbarten PE senden oder von ihm empfangen. Sei $l_i(t)$ die Last (gemessen in Anzahl Pakete) von PE i zum Zeitpunkt t ($i \in \mathcal{I} := \{1, \ldots, n\}^d$). Wann immer die beabsichtigte Zeit aus dem Kontext ersichtlich ist, wird der Parameter t weggelassen. Es sei $l_{\max} = \max_{i\in\mathcal{I}} l_i$, $l_{\min} = \min_{i\in\mathcal{I}} l_i$ und $\bar{l} = \sum_{i\in\mathcal{I}} l_i / n^d$. Gesucht ist ein Algorithmus, der die Last möglichst schnell und soweit wie möglich ausgleicht. Wegen der Diskretheit der Last wird diese Bedingung durch

$$\forall i \in \mathcal{I} : \lfloor \bar{l} \rfloor \leq l_i \leq \lceil \bar{l} \rceil \tag{1}$$

definiert. Wir sprechen in diesem Fall auch von *vollständigem Lastausgleich*. Ziel ist es außerdem, ein einfaches, nur auf lokaler Information beruhendes Verfahren zu haben, das kontinuierlich abläuft, so daß es auch für dynamische Lastverteilung einsetzbar ist. Weiterhin gibt es viele Anwendungen, bei denen nur *Load sharing* – d.h. $l_{\max} \leq 1 \vee \forall i \in \mathcal{I} : l_i > 0$ – interessiert, da alle nicht arbeitslosen PEs effektiv arbeiten können.

Dieses Modell läßt sich im wesentlichen als d-dimensionaler Zellularautomat mit von Neumann Nachbarschaft auffassen, bei dem die Systemlast Teil der Zustandsinformation ist. (Im strengen Sinne dann, wenn sich $l_{\max}(0)$ durch eine Konstante nach oben beschränken läßt.) Wir entlehnen deshalb im Zellularautomatenbereich übliche Terminolgie wie *Konfiguration* und „Bewegung“ von *Signalen* [20].

1.2 Der Basisalgorithmus

Das angewandte Verfahren ist im Grunde eines der einfachsten denkbaren: Kommuniziere zyklisch mit allen Nachbarn und tausche ein Lastelement aus, wenn lokal ein Ungleichgewicht besteht. Algorithmus 1 gibt eine präzise Beschreibung. Eine gewisse Komplikation besteht darin, die PEs so zu koordinieren, daß immer Einigkeit darüber besteht, über welche Verbindung jeweils zu kommunizieren ist. Deshalb wird in den PEs für jede Dimension eine Flagge verwaltet, die sowohl räumlich als auch zeitlich abwechselnd wahr und falsch ist.[2] Dadurch, daß im-

[1] Eine Verallgemeinerung auf ungleiche Seitenlängen ist auch möglich, würde die Diskussion aber unnötig komplizieren. Ähnliches gilt für die Synchronität, die leicht durch lokale Synchronisation zwischen asynchron laufenden PEs ersetzt werden kann.

[2] Bei einem strengen Zellularautomatenmodell, muß diese Färbung in einer n Schritte dauendern Vorberechnungsphase ermittelt werden, da der PE-Index nicht zur Verfügung steht.

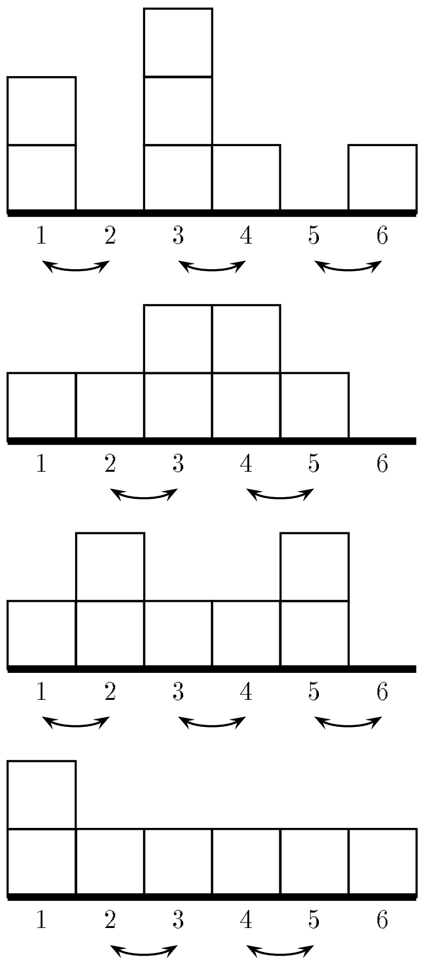
mer die PEs mit einer „wahren“ Flagge die Kontrolle übernehmen, läßt sich die Kommunikationsreihenfolge leicht regeln. Die Lastausgleichszyklen werden wiederholt, bis eine (hier offengelassene) Terminierungsbedingung erfüllt ist. Bei einer Anwendung für dynamische Lastverteilung würde z.B. dann abgebrochen, wenn die gesamte Last abgearbeitet ist.

Algorithmus 1 (Basisalgorithmus). Im folgenden bezeichnen $i = (i_1, \ldots, i_d)$ einen PE-Index und $e_1, \ldots, e_d$ die d-dimensionalen Einheitsvektoren.

FOR all PEs do in parallel and synchronously
  FOR j := 1 **TO** d **DO** F_j := odd(i_j)
  WHILE ¬finished
    FOR j := 1 **TO** d **DO**
      F_j := $\neg F_j$
      IF $F_j \wedge i + e_j \leq n \wedge l_{i+e_j} \neq l_i$ **THEN**
        IF $l_{i+e_j} > l_i$
        THEN get 1 unit from PE $i + e_j$
        ELSE put 1 unit to PE $i + e_j$

Abbildung 2 verdeutlicht dieses Prinzip für den eindimensionalen Fall. Lasteinheiten sind durch Quadrate dargestellt, und Doppelpfeile geben die PEs an, zwischen denen im nächsten Schritt Lastausgleich stattfindet.

Abbildung 2. Drei Lastverteilungsschritte für $d = 1$ und $n = 6$.



1.3 Andere Ansätze

Der in der Literatur bisher am breitesten behandelte Ansatz zur lokalen Lastverteilung beruht auf Mittelung der Last benachbarter PEs. Je nach Variante wird dies als *Diffusion* [3, 2, 23, 17, 18, 19], *Neighborhood balancing* oder *Dimension exchange* [21, 22] bezeichnet. Obwohl dort die oft schwer erfüllbare Annahme gemacht wird, daß Last beliebig unterteilbar ist, und in einem Schritt beliebig viel Last bewegt wird, ist dieses Verfahren relativ ineffizient. Es werden $\Omega(n^2)$ Schritte benötigt um auch nur ungefähre Konvergenz oder Load sharing zu erreichen. Dennoch sind Diffusionsansätze interessant, da sie durch lineare Differenzengleichungen beschrieben werden können, die zum Teil geschlossen gelöst werden können.

Unser Lastmodell läßt sich als „online“-Variante des *Tokenverteilungsproblems* [16, 1] auffassen. Algorithmen, die einzelne Lastpakete austauschen, sind auch in der Praxis weit verbreitet [13, 14, 15, 10]. Bei Analysen solcher Algorithmen werden aber meist vollständige Verknüpfung [11] oder beliebige Netzwerke [4] betrachtet, so daß die Ergebnisse für Gitter wenig aussagekräftig sind.

Das Prinzip, die Kommunikation in Gittern durch sich periodisch ändernde Flaggen zu regeln, ist wohlbekannt. Es wird zum Beispiel für *Odd-even transposition sort*, *2D-Bubble sort* und für parallele Gauss-Seidel Iteration eingesetzt [12, 7]. Auf dem gleichen Prinzip beruht auch die *Dimension exchange*-Variante des Diffusionsansatzes [22]. Allerdings wird meistens eine schachbrettartige Einfärbung verwendet, die in unserem Fall eine ausreichende „Durchmischung“ der Last verhindern würde (siehe auch Abschnitt 3.1). Eine Verallgemeinerung für beliebige Graphen findet sich in [4], wo *Random matchings* verwendet werden.

Nah verwandt zu unserem Ansatz ist das in [5, 6] beschriebene *Flüssigkeitsmodell*. Es gibt aber einige Unterschiede: Dort wird eine torusförmige Verbindung vorausgesetzt. PEs müssen gleichzeitig Last empfangen und senden können. Last wandert immer in eine Richtung und es wird selbst dann Last übertragen, wenn beide betrachteten PEs gleich belastet sind. Außerdem wird der Begriff der Balanciertheit dort weniger streng definiert.

Sehr ähnlich ist auch das in [9] beschriebene Verfahren für eindimensionale Gitter. Allerdings wird dort innerhalb einer einzigen Iteration mit beiden Nachbarn kommuniziert. Dieser scheinbar kleine Unterschied hat die fatale Folge, daß das Verfahren in z.T. extrem unbalancierten Situationen stecken bleiben kann.

2 Der eindimensionale Fall

Wir führen nun eine detaillierte Analyse vor, die zeigt, daß der Basisalgorithmus 1 für 1-dimensiomale Gitter asymptotisch optimal ist.

Satz 3. *Für $d = 1$, und eine beliebige Anfangskonfiguration ist nach höchstens $n(l_{\max} - l_{\min})$ Schritten eine balancierte Konfiguration erreicht.*

Wir beweisen zunächst das folgende Lemma über das Verhalten von Maximum- und Minimumstellen.

Lemma 4. *Eine Maximumstelle (Minimumstelle) führt eine periodische Pendelbewegung zwischen linkem und rechtem Rand mit Periode $2n$ aus, bis sie abgebaut wird, indem mit einem PE Last ausgetauscht wird, das mindestens eine um zwei kleinere (größere) Last hat.*

Beweis. Wir führen nur den Fall einer Maximumstelle vor. Der Beweis für Minumustellen verläuft analog. Es sei $l_x(t_0) = l_{\max}$. Weiterhin sei F_1 zum Zeitpunkt t_0 auf PE x wahr und $x < n$. Dann läßt sich folgende Fallunterscheidung machen:

$l_{x+1}(t_0) = l_{\max}(t)$ Es passiert zwar nichts, aber wir können das so auffassen, daß die Maximumstellen ausgetauscht wurden.
Insbesondere gilt $l_{x+1}(t_0 + 1) = l_{\max}(t)$.

$l_{x+1}(t_0) = l_{\max}(t) - 1$ Der Zustand der beiden Zellen wird ebenfalls ausgetauscht.

$l_{x+1}(t_0) < l_{\max}(t) - 1$ Die Maximumstelle wird abgebaut.

Im nächsten Schritt ist F_1 auf PE $x + 1$ ebenfalls wahr, usw., so daß die Maximumstelle in $n - x$ Schritten zum rechten Rand läuft. Dort passiert einen Schritt lang gar nichts und dann ist F_1 auf der PE der Maximumstelle falsch. Auf Grund einer ähnlichen Betrachtung wie oben läuft die Maximumstelle anschließend in $n - 1$ Schritten zum linken Rand, setzt wieder einen Schritt aus und läuft dann in $x - 1$ Schritten zu PE x und ist wieder auf einem PE mit wahrem F_1. Dieser Prozeß wiederholt sich damit beliebig oft, solange die Maximumstelle auf kein PE mit Last kleiner $l_{\max} - 1$ trifft. Ist F_1 auf dem PE der Maximumstelle anfangs nicht wahr oder befindet diese Stelle sich an einem der Ränder, läuft der gleiche Prozeß in anderer Reihenfolge ab. □

Beweis. (von Satz 3). Es genügt zu zeigen, daß die Differenz zwischen $l_{\max}$ und $l_{\min}$ in jeweils n Schritten um mindestens eins abgebaut wird, wenn die Last nicht balanciert ist. Genauer:

$$l_{\max}(t_0) > \lceil \bar{l} \rceil \vee l_{\min}(t_0) < \lfloor \bar{l} \rfloor \Rightarrow$$
$$l_{\max}(t_0 + n) < l_{\max}(t_0) \vee l_{\min}(t_0 + n) > l_{\min}(t_0).$$

Nehmen wir das Gegenteil an. Dann gibt es eine Maximumstelle x und eine Minimumstelle y mit $l_x(t_0) - l_y(t_0) > 1$, die mindestens n Schritte lang die in Lemma 4 beschriebene Pendelbewegung durchführen ohne abgebaut zu werden. Dies steht aber im Widerspruch zu einer einfachen geometrischen Betrachtung[3], die zeigt, daß Minimum- und Maximumstelle sich irgendwann innerhalb der n Schritte begegnet sein müssen, und dadurch abgebaut worden sein müssen. □

Dieses Ergebnis ist zumindest asymptotisch optimal, da es auf Grund der beschränkten Bisektionsbreite eines 1-dimensionalen Gitters Ausgangskonfigurationen gibt, bei denen jeder Lastausgleichsalgorithmus, bei dem die Anzahl der von einem PE in einem Schritt abzugebenden oder zu empfangenden Lasteinheiten durch eine Konstante beschränkt ist, $\Omega(n(l_{\max} - l_{\min}))$ Schritte benötigt. Zum Beispiel werden im von uns betrachteten Modell mindestens $(n-1)\bar{l}$ Schritte benötigt, um die Ausgangskonfiguration $l_1 = \cdots = l_{n/2} = 2\bar{l}$ und $l_{n/2+1} = \cdots = l_n = 0$ zu balancieren: PE $n/2 + 1$ muß mindestens $\bar{l}n/2$ Pakete empfangen und $\bar{l}n/2 - \bar{l}$ Pakete wieder verschicken. Unser Algorithmus benötigt für diese Ausgangskonfiguration höchstens $n\bar{l}$ Schritte also nur wenig mehr.

Tatsächlich ist die in Satz 3 bewiesene Schranke oft nur eine grobe Abschätzung der tatsächlichen Effizienz des Algorithmus. Zum Beispiel läßt sich der folgende Satz zeigen:

Satz 5. *Für jede Ausgangskonfiguration gilt nach höchstens $2n$ Schritten:*

$$\forall i \in \mathcal{I} : l_i > 0 \vee \forall i \in \mathcal{I} : l_i \leq 1 \ .$$

Beweis. (Skizze) Stellen mit Last Null führen während der gesamten Dauer ihrer Existenz die in Lemma 4 beschriebene Pendelbewegung durch. Gibt es eine Stelle mit Last größer eins, die länger als $2n$ Schritte existiert, so müßte diese irgendwann mit jeder nicht anderweitig beseitigten Nullstelle zusammengetroffen sein, wodurch diese beseitigt werden. Demnach gibt es nach $2n$ Schritten entweder keine 0-Stellen oder keine Stellen mit Last größer eins. □

Insbesondere wird also in höchstens $2n$ Schritten Load sharing erreicht. Außerdem wird für alle Ausgangskonfigurationen mit $\bar{l} \leq 1$ die Last in höchstens $2n$ Schritten balanciert. Für diese Klasse von Konfigurationen kann kein Algorithmus systematisch mehr als einen Schritt schneller sein, weil für $l_1 = n, l_x = 0$ für $x > 1$ mindestens $2n - 1$ Schritte benötigt werden.

3 Der mehrdimensionale Fall

Wir wenden uns nun der Lastverteilung in höherdimensionalen Gittern zu. Die Formulierung von Algorithmus 1 im vorangegangenen Abschnitt war bereits auf diesen Fall ausgelegt.

Wie auch bei anderer Gelegenheit[4] ergeben sich beim Übergang vom Ein- zum Zweidimensionalen zusätzliche Schwierigkeiten. Es gibt Situationen, in denen der Basisalgorithmus im Mehrdimensionalen nicht mehr vollständig lastausgeglichene Konfigurationen im Sinne von Ungleichung (1) erreicht.

Im ersten Unterabschnitt wird daher zunächst diskutiert, welche Eigenschaften des Basisalgorithmus allgemein gelten. In einem zweiten Unterschnitt stellen wir

[3]Ein detaillierterer Beweis würde sich einer Fallunterscheidung nach relativer Lage von x und y sowie dem Zustand der Flaggen bedienen.

[4]Zum Beispiel ist die Überprüfung der globalen Überführungsfunktion auf Surjektivität anhand der lokalen Überführungsfunktion für eindimensionale Zellularautomaten algorithmisch durchführbar, für zwei- und höherdimensionale aber nicht mehr [8].

dann einen verfeinerten Algorithmus für zweidimensionale Gitter vor, der stets eine ausgeglichene Lastverteilung herstellt. Seine Verallgemeinerung auf Dimensionen $d \geq 3$ ist dann ebenfalls offensichtlich.

3.1 Der Basisalgorithmus im Mehrdimensionalen

Die Konvergenz von Algorithmus 1 im Eindimensionalen beruht darauf, daß Lastminima und Lastmaxima irgendwann zusammenstoßen müssen und dadurch abgebaut werden. Leider ist dies im Mehrdimensionalen nicht so einfach. Zwar läßt sich Lemma 4 verallgemeinern, weil die Projektionen der Maximum- und Minimumstellen auf jede einzelne Dimension eine Pendelbewegung mit Periode $2dn$ durchführen. (Dies ist der Grund warum für jede Dimension eine eigene Flagge verwaltet wird.) Dies bedeuted jedoch nicht, daß die Minima und Maxima kollidieren müssen. Der folgende Satz zeigt, daß Konvergenz in unserem strengen Sinne in der Tat nicht mehr gegeben ist.

Satz 6. *Es gibt Ausgangskonfigurationen mit $l_{\max} - l_{\min} = d$ bei denen das Lastungleichgewicht von Algorithmus 2 nicht abgebaut werden kann.*

Beweis. (Skizze) Betrachte den Fall $n = 2$ und $l_i := \sum_{j=1}^{d} i_j - 1$ (i_j bezeichnet wieder die j-te Komponente von i). Benachbarte PEs unterscheiden sich in ihrer Last jeweils nur um eins, so daß ein Lastausgleichsschritt zum Austausch der Last benachbarter PEs führt. Da der Austausch synchron stattfindet, ergibt sich daraus eine Spiegelung der Lastkonfiguration entlang Dimension j. Dies ändert aber nichts an der Invariante, daß jeweils nur PEs mit Lastunterschied eins benachbart sind. □

Offensichtlich ist Satz 6 auf allgemeinere Situationen übertragbar.[5] Ausgenützt wird im Beweis nur, daß die Abbildung, die eine Lastkonfiguration in die nächste überführt, mit einer Symmetrie der Hyperwürfel kommutiert. Dagegen geht die Art der lokalen Vorgehensweise bei der Lastverteilung nicht mit ein. Über weitere Untersuchungen zu diesem Thema wird an anderer Stelle berichtet werden.

Für viele Anwendungen ist ein Lastungleichgewicht bis zu d aber durchaus akzeptabel. Und tatsächlich gibt es gute Gründe anzunehmen, daß Algorithmus 1 in Zeit $\Theta(n(l_{\max} - l_{\min}))$ eine Situation mit maximalem Ungleichgewicht d herstellen kann. Dies ist asymptotisch optimal. Denn es gibt Konfigurationen, die von keinem Algorithmus schneller balanciert werden können, weil Gitter nur über eine beschränkte Bisektionsbreite verfügen. Ein detaillierter Beweis ist hier aus Platzmangel leider nicht möglich, da eine Vielzahl von Fallunterscheidungen nötig erscheint. Die Grundidee ist ähnlich wie im Beweis von Satz 3. Aus $l_{\max}(t_0) - l_{\min}(t_0) > d$ folgt die Existenz von Stellen $x_0, \ldots, x_d$ mit $l_{x_0}(t_0) = l_{\min}(t_0)$ und $l_{x_d}(t_0) = l_{\max}(t_0) > l_{\min}(t_0) + d$, die so wandern, daß innerhalb von jeweils $\mathrm{O}(n)$ Schritten x_j auf x_{j+1} trifft. Dies führt in $\mathrm{O}(n)$ Schritten zum Abbau des Maximums oder des Minimums.

Eine mehrdimensionale Verallgemeinerung von Satz 5, die zeigen würde, daß Load sharing in Zeit $\mathrm{O}(n)$ erreicht wird, gilt leider nicht. Unabhängig vom Algorithmus entsteht nämlich für $d > 1$ ein Bandbreitenproblem wenn die gesamte Last in einem kleinen Teil des Gitters konzentriert ist.[6]

3.2 Ein Algorithmus für vollständigen Lastausgleich in zweidimensionalen Gittern

Das Problem des zweidimensionalen Basisalgorithmus wird im nachfolgenden Bild deutlich. Es zeigt eine Folge von Konfigurationen, die der Basisalgorithmus zyklisch durchläuft, ohne vollständigen Lastausgleich zu erreichen:

```
1211 2111 1101 1011 1011 0111 1111 1111
1110 1101 2111 2111 1111 1111 0111 0111
1111 1111 1111 1111 2111 1211 1111 1111
1111 1111 1111 1111 1111 1111 1211 1121

1111 1111 1111 1111 1111 1111 1121 1211
1111 1111 1111 1111 1112 1121 1111 1111
0111 1011 1112 1112 1111 1111 1110 1110
1121 1112 1011 1101 1101 1110 1111 1111
```

Im Unterschied zum Eindimensionalen ist es nun nicht mehr erzwungen, daß sich eine Minimum- und eine Maximumstelle jemals „treffen“, um den notwendigen Lastausgleich durchzuführen.

Deshalb wird der Basisalgorithmus dahingehend geändert, daß diese Eigenschaft doch wieder erfüllt ist. Absolute Lastmaxima und -minima, die nicht schon vorher abgebaut werden, werden „gezwungen“, sich innerhalb von $\mathrm{O}(n)$ Schritten zu unmittelbar benachbarten PEs zu „begeben“ und sich spätestens an diesem Treffpunkt anzugleichen.

Als einen solchen Treffpunkt ist im folgenden zum Beispiel die Mitte des linken Randes des Gitters gewählt. Dazu werden zwei „Trichter“ vorgesehen, die dafür sorgen, daß sich Lastmaxima entgegen dem und Lastminima im Uhrzeigersinn dieser Stelle nähern.

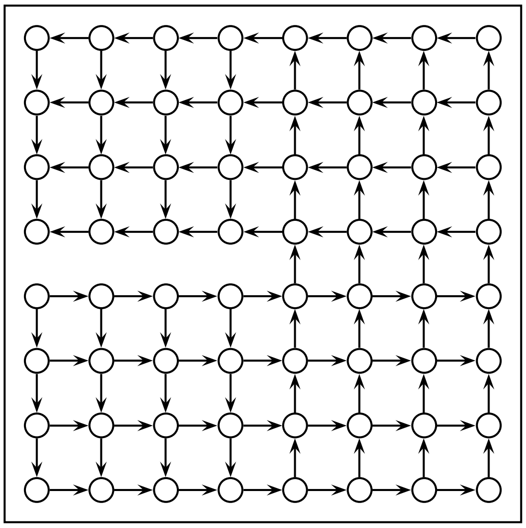
Um das gewünschte Verhalten zu erreichen, wird das Verhalten zweier benachbarter PEs, die in einem Schritt gegebenenfalls Last austauschen sollen, gegenüber dem Basisalgorithmus wie folgt geändert:

- Ist die Lastdifferenz größer oder gleich 2 wird wie früher auf jeden Fall eine Lastangleichung vorgenommen.
- Ist die Lastdifferenz genau 1, so wird ein Lastpaket nur dann von einem PE zu seinem Nachbarn transportiert, wenn es sich dabei entlang der *Vorzugsrichtung* bewegt. Die jeweiligen Vorzugsrichtungen sind in Abbildung 7 durch Pfeile dargestellt. Insbesondere findet also zwischen PEs mit Koordinaten

[5] Zum Beispiel gilt dies für den Algorithmus aus [5] (siehe auch [6]).
[6] Das gleiche gilt für Tori und damit für die Algorithmen in [6].

der Form $(x, n/2)$ und $(x, n/2+1)$ für ein x mit $1 \leq x \leq n/2$ bei Lastdifferenz 1 *kein* Lasttransport statt.

Abbildung 7. Vorzugsrichtungen für Lastpakete bei Lastunterschied 1. Fehlende Pfeile zwischen benachbarten PEs bedeuten, daß bei Lastunterschied 1 *kein* Lasttransport stattfindet.



Die Vorzugsrichtungen stellen sicher, daß sich Lastmaxima entgegen dem Uhrzeigersinn zum PE $(n/2, 1)$ bewegen und Lastminima im Uhrzeigersinn zum PE $(n/2+1, 1)$. Sofern $l_{\max} - l_{\min} \geq 2$ ist, findet dort eine Lastangleichung statt. Wir sprechen im folgenden gelegentlich auch von der *Mauer* zwischen den PEs $(x, n/2)$ einerseits und den PEs $(x, n/2+1)$ andererseits ($1 \leq x \leq n/2$).

Die obige Überlegung macht auch schon klar, daß der Algorithmus in jeden Fall nach endlich vielen Schritten eine Konfiguration erreicht, die im Sinne von Ungleichung 1 vollständig balanciert ist.

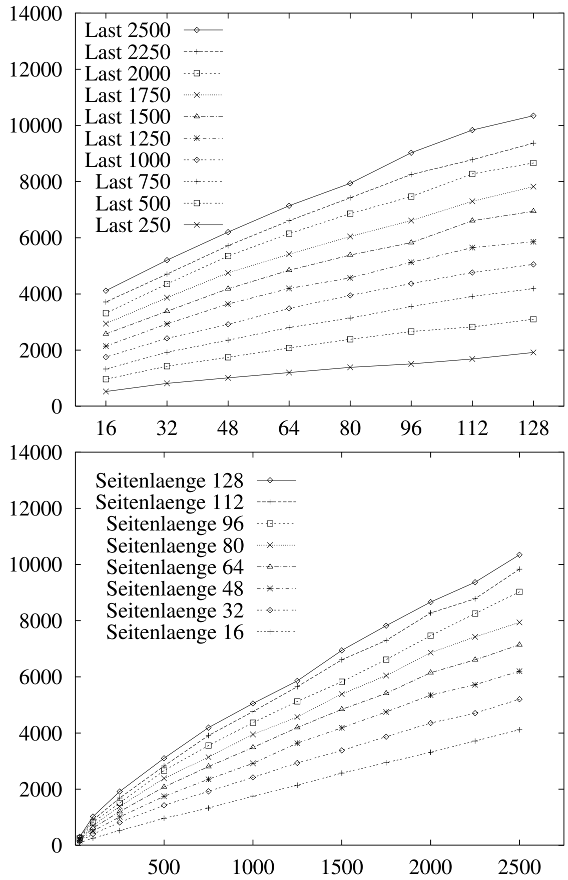
Eine diesem Vorgehen ähnliche Idee wird in [19] untersucht. Da hierbei aber Lastbalancierung durch Diffusion und auf einer größeren Familie von Graphen als nur Gittern untersucht wird, sind die sich ergebenden Schranken für den Zeitbedarf bis zum vollständigen Lastausgleich verhältnismäßig groß (in $\Omega(n^d)$).

Hingegen wird man für den eben vorgestellten Algorithmus ein schnelleres Erreichen ausgeglichener Konfigurationen erwarten können. Die beiden Viertel des Gitters oberhalb bzw. unterhalb der Mauer wirken als eine Art „Trichter", in die Stellen mit großer bzw. kleiner Last hineinfließen. Handelt es sich dabei auf beiden Seiten um „sehr viele", so wirkt aber die Stelle, an der die beiden Spitzen der Trichter zusammenstoßen nicht etwa als Flaschenhals. Vielmehr wird dann entlang der gesamten Mauer Lastausgleich vorgenommen. Da die Mauer eine Länge in Θn hat, die also größenordnungsmäßig gleich der Bisektionsbreite ist, wird hierdurch der Lastausgleich nicht künstlich verlangsamt.

Leider stellt sich die mathematisch präzise Analyse der für den Lastausgleich benötigten Zeit als recht schwierig heraus. Stattdessen wollen wir daher im folgenden einige Meßergebnisse einer Reihe von Simulationsläufen (auf einer MasPar MP-1 mit 2^{14} Prozessoren) referieren.

Sofern nicht ausdrücklich etwas anderes gesagt wird, sind die für die folgenden Abbildungen verwendeten „Laufzeiten" arithmetische Mittelwerte von Schrittzahlen, die bei Simulationen für Anfangskonfigurationen benötigt wurden, bei denen mit einem Pseudozufallszahlengenerator die Anfangslasten der PEs zufällig, gleichverteilt und unabhängig zwischen 0 und einer vorgegebenen Maximallast L gewählt wurden.

Abbildung 8. Absolute Laufzeiten t in Abhängigkeit von n für verschiedene L (oben) und in Abhängigkeit von L für verschiedene n (unten).



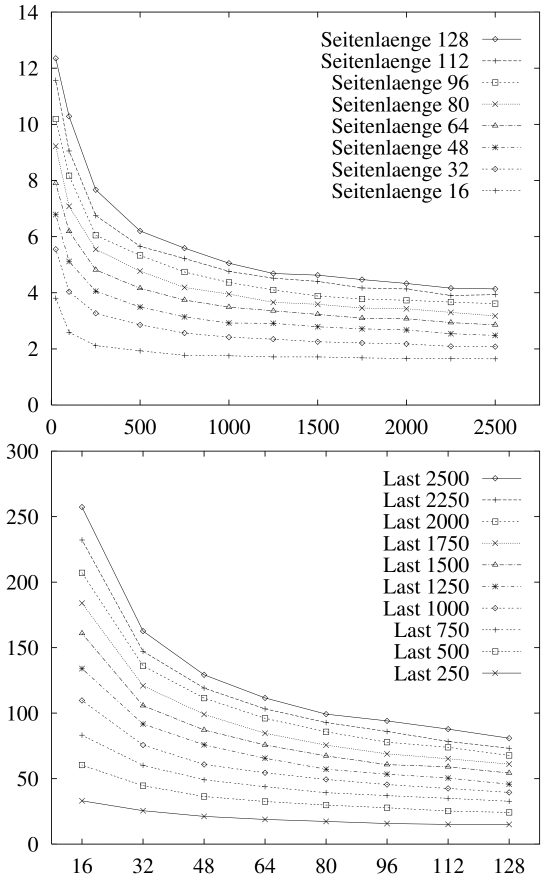
Im oberen Diagramm von Abbildung 8 sind für einige L die Laufzeiten für Werte n zwischen 16 und 128 aufgetragen. Darunter sind die Laufzeiten über der anfänglichen Maximallast aufgetragen. Die im Durchschnitt benötigte Zeit steigt sowohl mit der anfänglichen Maximallast als auch mit der Gittergröße. Dies erscheint auch qualitativ plausibel.

In beiden Fällen ist kein Gesetz für den Zusammenhang zwischen Laufzeit, Maximallast und Gittergröße offensichtlich. Zwar gibt die Theorie Hinweise darauf, daß man einen Einfluß von L und von $\sqrt{\log n}$ erwarten kann. Leider führten die entsprechende Untersuchungen der Meßdaten bislang nur zur bruchstückhaften Ergebnissen.

Man betrachte zum Beispiel das linke Diagramm in Abbildung 9. Dort ist auf der Ordinate jeweils der Quotient aus Laufzeit und anfänglicher Maximallast aufgetragen. Die Abbildung suggeriert, daß dieser Quotient für große

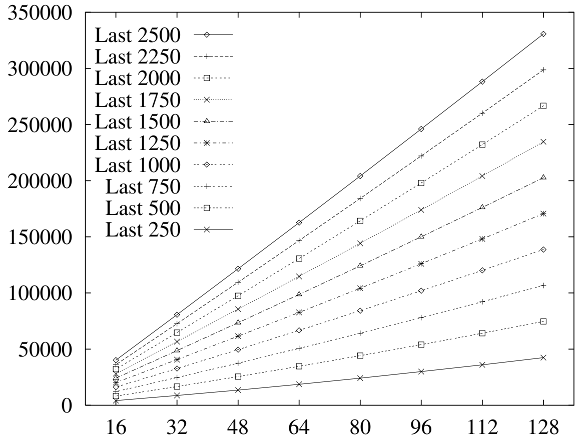
Lasten durch Konstanten nach oben und unten beschränkt ist, daß also mit anderen Worten die Laufzeit asymptotisch linear mit der Maximallast wächst.

Abbildung 9. t/L in Abhängigkeit von L für verschiedene n (oben) und t/n in Abhängigkeit von n für verschiedene L (unten).



Ein durch die bisherige Diskussion gar nicht erfaßter Aspekt ist der des „schlimmsten Falles“. Hierauf sei abschließend kurz eingegangen. Es ist nicht nur so, daß solche Fälle durch die Mittelwertbildung für die obigen Diagramme nicht mehr zur Geltung kommen würden, sondern tatsächlich war unter den während der Simulationen pseudozufällig gewählten Anfangskonfigurationen keine einzige besonders aufwendige. Die einzelnen Laufzeiten streuten zum Beispiel bei $n = 128$ und $L = 2500$ „nur“ um maximal 50 Prozent um den Mittelwert von 10346. Ähnlich wie in Abschnitt 2 lassen sich aber gezielt Anfangskonfigurationen konstruieren, für die der vorgestellte Algorithmus weitaus länger für den Lastausgleich braucht (im genannten Beispielfall 330680 Schritte). Einen Überblick über die Laufzeiten gibt Abbildung 10. Beim Vergleich mit Abbildung 8 achte man auf die unterschiedliche Skalierung!

Abbildung 10. Laufzeiten für ein „schlimmen“ Typ von Anfangskonfigurationen.



4 Zusammenfassung und Ausblick

Der verblüffend einfache Ansatz, einzelne Lastpakete zwischen benachbarten PEs auszutauschen wann immer ein lokaler Lastunterschied besteht, ist schon in einer der einfachsten denkbaren Realisierungen ein sehr effektives Lastausgleichsverfahren. Im Eindimensionalen werden nahezu optimale und im im mehrdimensionalen Fall asymptotisch optimale Ergebnisse erzielt.

Durch Einführung einer gewissen Inhomogenität in das Verhalten der PEs kann sogar im Mehrdimensionalen ein vollständiger Lastausgleich erzielt werden. Da der kurzperiodische und lastunabhängige Charakter des Algorithmus dadurch nicht verändert wird, läßt er sich weiterhin als kontinuierlich arbeitender dynamischer Lastverteiler einsetzen. Simulationen zeigen, daß unsere Algorithmen für pseudozufällige Ausgangssituationen deutlich effizienter sind, als es die Betrachtungen für den schlimmsten Fall erwarten lassen mögen.

Die wahrscheinlichkeitstheoretische Begründung für diese Beobachtung wird einer der Gegenstände nachfolgender Arbeiten auf diesem Gebiet sein. Andere Fragestellungen sind die detailliertere Ausarbeitung der Analyse des mehrdimensionalen Basisalgorithmus, asymptotische Aussagen über die Geschwindigkeit des modifizierten Algorithmus, sowie die Untersuchung von Algorithmenvarianten. Zum Beispiel gibt es eine relativ einfache Möglichkeit, den „Trichter“, in dem sich Lastminima und Lastmaxima treffen, in das Zentrum des Gitters zu rücken und zu einem symmetrischeren und wahrscheinlich etwas schnelleren Algorithmus zu gelangen.

Eine interessante Fragestellung ist auch eine möglichst realistische Modellierung von dynamischen Lastverteilungsproblemen und deren Interaktion mit dem Algorithmus. Vorläufige Ergebnisse zu Branch-and-bound Algorithmen liegen bereits vor.

Literatur

[1] F. M. auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. In *ICALP*, pages 398–409, 1993.

[2] J. E. Boillat. Load balancing and Poisson equation in a graph. *Concurrency: Practice and Experience*, 2(4):289–311, 1990.

[3] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[4] B. Ghosh, F. T. Leighton, B. M. Maggs, S. Muthukrishnan, C. G. Plaxton, R. Rajaraman, A. W. Richa, T. E. Tarjan, and D. Zuckerman. Tight analyses of two local load balancing algorithms. In *ACM Symposium on the Theory of Computing*, 1995.

[5] D. Henrich. *Lastverteilung für Branch-and-bound auf eng-gekoppelten Parallelrechnern*. Dissertation, Universität Karlsruhe, 1994.

[6] D. Henrich. Local load balancing according to a simple liquid model. In *PARS Workshop*, PARS Mitteilungen, Stuttgart, 1995.

[7] D. Ierardi. 2d-bubblesorting in average time $O(\sqrt{N \lg N})$. In *ACM Symposium on Parallel Architectures and Algorithms*, 1994.

[8] J. Kari. Reversibility of 2D CA is undecidable. *Physica D*, 45, 1990.

[9] G. A. Kohring. Dynamic load balancing for parallelized particle simulations on MIMD computers. *Parallel Computing*, 21:683–693, 1995.

[10] N. Kuck, M. Middendorf, and H. Schmeck. Generic branch-and-bound on a network of transputers. In R. Grebe et al., editors, *Transputer Applications and Systems*, pages 521–535. IOS Press, 1993.

[11] T. Lauer. *Adaptive dynamische Lastbalancierung*. PhD thesis, Max Planck Institut für Informatik Saarbrücken, 1995.

[12] T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann, 1992.

[13] F. C. Lin and R. M. Keller. The gradient model load balancing method. *IEEE Transactions on Software Engineering*, 13(1):32–38, 1987.

[14] R. Lüling, B. Monien, and F. Ramme. Load balancing in large networks: A comparative case study. In *3th IEEE Symposium on Parallel and Distributed Processing*. IEEE, 1991.

[15] G. P. McKeown, V. J. Rayward-Smith, and S. A. Rush. Parallel branch-and-bound. In *Advances in Parallel Algorithms*, pages 349–362. Blackwell, 1992.

[16] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal of Computing*, 18:229–243, 1989.

[17] X. Qian and Q. Yang. An analytical model for load balancing on symmetric multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 20:198–211, 1994.

[18] J. Song. A partially asynchronous and iterative algorithm for distributed load balancing. *Parallel Computing*, 20:853–868, 1994.

[19] R. Subramanian and I. D. Scherson. An analysis of diffusive load-balancing. In *ACM Symposium on Parallel Architectures and Algorithms*, pages 220–225, 1994.

[20] R. Vollmar. *Algorithmen in Zellularautomaten*. Teubner, 1979.

[21] M. H. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9), 1993.

[22] C. Xu and F. C. Lau. Decentralized remapping of data parallel computations with the generalized dimension exchange method. In *Scalable High-Performance Computing Conference*, pages 414–421, 1994.

[23] C. Xu and F. C. Lau. Optimal parameters for load balancing with the diffusion method in mesh networks. *Parallel Processing Letters*, 4(1):139–147, 1994.