Kapitel 3

Parallelisierung

Da diese Dissertation neue methodische Verfahren einer theoretisch-chemische Arbeitsgruppe vorstellen soll, wird aus Gründen der Selbstkonsistenz für die Parallelisierung der Programme hier zunächst ein kurzer Überblick über die Arbeitsweise eines Parallelrechners gegeben und anhand eines einfachen Beispiels die Parallelisierung eines Programmes vorgestellt werden. Dabei ist es für die Parallelisierung wichtig, sich auf ein bestimmtes Programmiermodell festzulegen.

3.1 Parallele Rechnerarchitektur und Programmiermodell

Im folgenden soll das verwendete Programmiermodell vorgestellt werden. Hierzu ist es notwendig, sich zunächst ein Bild von einer parallelen Rechnerarchitektur zu machen. Dabei besteht ein Parallelrechner aus P verschiedenen Prozessoren p, $0 \leq p < P$ mit jeweils lokalem Speicher M. Da die Prozessoren im Laufe einer Rechnung Daten austauschen müssen, sind sie über ein Kommunikationsnetzwerk miteinander verbunden, wodurch die in Abb.3.1 dargestellte Konfiguration entsteht. Für die parallele Nutzung einer solchen Architektur existieren zwei grundsätzlich verschiedene Konzepte *MIMD* und *SIMD*.

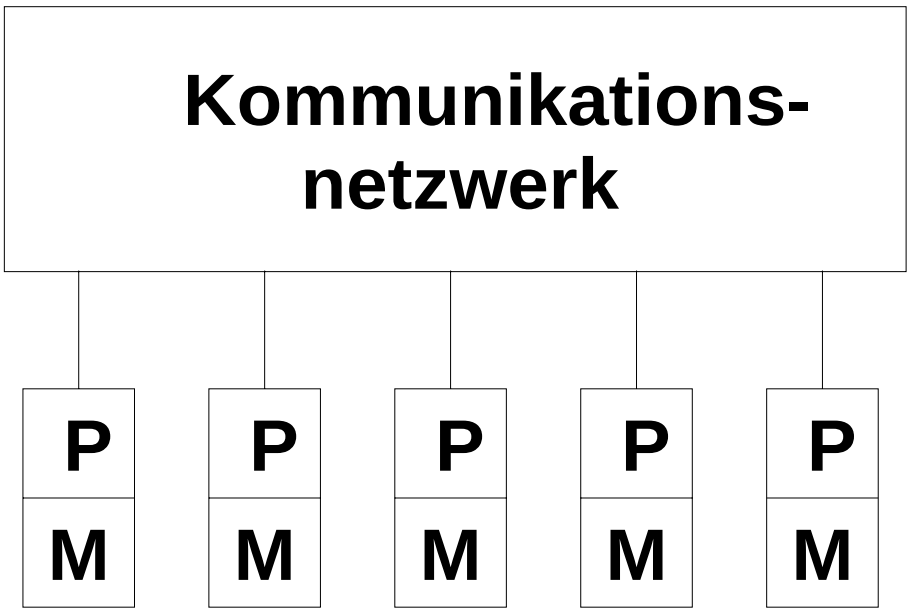

Abbildung 3.1: *Parallele Architektur bestehend aus 5 Prozessoren P, die jeweils mit lokalem Speicher M ausgestattet und über ein Netzwerk verbunden sind.*

MIMD, steht für *Multiple Instruction Multiple Data*. Dabei handelt es sich um ein Programmiermodell, bei dem die verschiedenen Prozessoren unterschiedliche Programme (Instruktionen) auf unterschiedlichen Daten operieren lassen. Als eine mögliche Anwendung für molekulare Dynamikrechnungen kann man sich vorstellen, daß nicht in konventioneller Weise zuerst die Potentialflächen an allen Punkten des Raumes berechnet werden und anschließend auf ihnen die Dynamik des Systems bestimmt wird, sondern daß ein Prozessor die Dynamik berechnet, während die anderen Prozessoren mit der Berechnung der Potentialfläche in den Regionen beschäftigt sind, in denen sich das System gerade befindet. Ein solcher Ansatz, die Potentialflächen sozusagen "on the fly" zu berechnen, wird von M. Parinello [53] verfolgt.

SIMD, bedeutet *Single Instruction Multiple Data* und verfolgt einen grundsätzlich anderen Ansatz. Hierbei ist man bemüht, die zu verarbeitenden Daten in einer so geschickten Weise auf die Prozessoren zu verteilen, daß sie anschließend mit ein und demselben Code verarbeitet werden können. Dieses Verfahren wird besonders

häufig für Probleme der numerischen Mathematik eingesetzt, wie z.B. Matrixinversion und Lösung großer Differentialgleichungen, wie unter anderem auch zur Lösung der Schrödingergleichung.

Zur Verdeutlichung des SIMD-Konzepts zeigt Abb 3.2 ein Beispiel, bei dem mit Hilfe von drei Prozessoren die Summe der Koordinaten eines 9-komponentigen Vektors $\vec{x} \in \mathbb{R}^9$ berechnet wird.

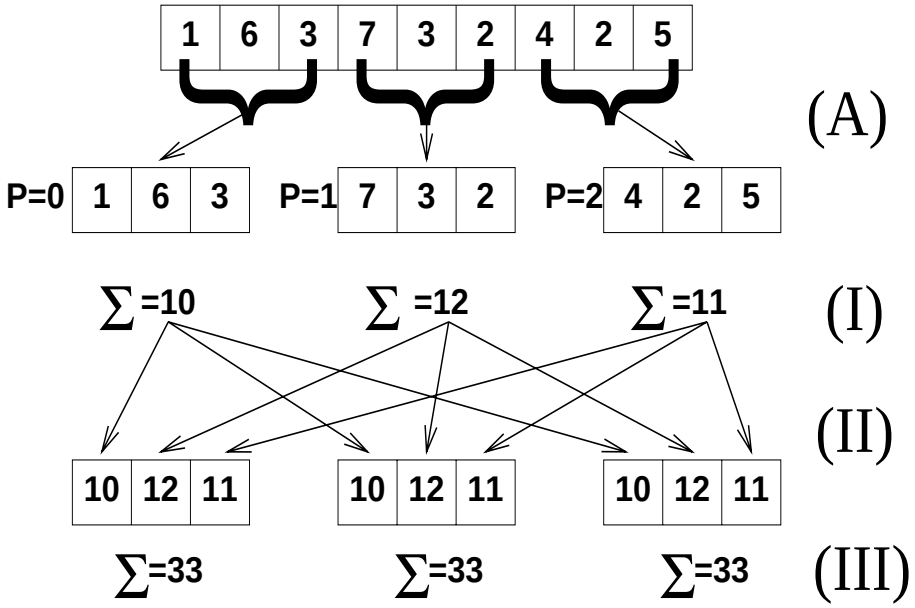

Abbildung 3.2: *Parallele Berechnung der Summe der Koordinaten eines 9-komponentigen Vektors, mit drei Prozessoren.*

Dabei erhält jeder der drei Prozessoren zunächst eine Identität zugewiesen, im Falle von $P = 3$ Prozessoren ist dies eine ganze Zahl $0 \leq p < P$, also 0,1 oder 2. Danach werden die anfänglichen Daten auf die Prozessoren verteilt [1](in Abb. 3.2 Schritt A), typischerweise so, daß jeder Prozessor nur diejenigen Daten erhält, die er tatsächlich benötigt, hier also ein Drittel des Vektors. Diese Datenverteilung wird in

[1]Wenn im folgenden von einer Verteilung der Daten auf die Prozessoren gesprochen wird, so ist damit gemeint, daß Daten in die zu den Prozessoren gehörigen lokalen Speicher geschrieben werden.

der Regel beim Programmstart vom Prozessor mit der Identität $p = 0$ vorgenommen. Ein darauffolgender Ablauf von SIMD-Programmen besteht dann aus einer Sequenz von Blöcken, die stets die folgende Struktur aufweisen:

- lokale Operationen
- Datenaustausch
- Synchronisation

Die erste lokale Operation in diesem Beispiel besteht in der Berechnung der Teilsummen aus den lokalen Daten, in Abb. 3.2 geschieht dies in Schritt (I). Zur Bildung der Gesamtsumme benötigt ein jeder Prozessor alle Teilsummen der anderen Prozessoren, so daß sich ein Datenaustausch anschließen muß, in dem jeder Prozessor seine noch lokale Teilsumme den anderen Prozessoren bekannt gibt. Dieser Datenaustausch wird durch eine Synchronisation abgeschlossen, die sicherstellt, daß nach ihr alle "verschickten" Daten auch "angekommen" sind. Diese beiden Schritte, Datenaustausch und Synchronisation, sind in Abb. 3.2 als Schritt (II) zusammengefaßt, in dem jeder der drei Prozessoren die Teilsummen aller anderen erhält und diese zunächst zwischenspeichert. Im abschließenden dritten Schritt (III) folgt wieder eine Operation auf rein lokalen Daten, wobei jeder Prozessor die Teilsummen zum Endergebnis aufaddiert. Ein "C"-Programm für dieses Beispiel ist in Anhang A zu finden.

Auch wenn die Parallelisierung des hier vorgestellten Beispiels aufgrund seiner geringen Gößе wenig sinvoll ist, so zeigt es doch, welche Funktionen in parallelisierten Programmen zusätzlich benötigt werden, nämlich Funktionen zum Datenaustausch zwischen den Prozessoren und zu deren Synchronisation. Diese Funktionen werden durch Programmbibliotheken, wie z.B. der hier verwendeten *Oxford BSP-Library* [54, 55, 56], bereitgestellt. Die BSP-Bibliothek gestattet eine hardwareunabhängige Implementierung der Programme und ist für annähernd alle Plattformen erhältlich. Darüberhinaus erlaubt sie die Formulierung eines abstrakten Modells für einen Parallelcomputer mit einer Theorie für die Abschätzung der Rechenzeit des parallelisierten Codes, was im folgenden Abschnitt vorgestellt wird.

3.2 Das BSP-Modell und Rechenzeitabschätzung

Das Konzept des BSP-Programmiermodells (Bulk-Synchronous-Parallel) geht zurück auf Valiant [57], der die in Abb. 3.1 dargestellte Rechnerarchitekur als abstraktes Modell einer parallelen Programmierung zugrunde legte. Als Ziele beim Entwurf des Modells wurde zum einen verfolgt, eine portable Programmbibliothek zu erhalten, zum anderen eine Vorhersage der Rechenzeit von Programmen innerhalb dieses Modells treffen zu können. Dieses auf R.H. Bisseling [55] zurückgehende Verfahren wird zusammen mit den notwendigen Definitionen für das BSP-Modell wird hier kurz vorgestellt, um später die erzielte Performance der entwickelten Programme mit den theoretischen Vorhersagen vergleichen zu können.

Ein **BSP-Computer** besteht aus P Prozessoren mit jeweils lokalem Speicher, die über ein Kommunikationsnetzwerk untereinander verbunden sind. Jeder Prozessor kann alle Speicherzellen des gesamten Rechners ansprechen, wobei eine Lese- bzw. Schreiboperation im lokalen Speicher relativ schnell ist. Wird jedoch eine nicht-lokale Speicherzelle angesprochen, so muß die Information über das Netzwerk verschickt bzw. angefordert werden, woraus wesentlich längere Zugriffszeiten resultieren. Vereinfachend wird die Modellannahme getroffen, daß die Zugriffszeit auf nicht-lokalen Speicher immer die gleiche ist, unabhängig davon, wo sich der Speicher befindet.

Ein **BSP-Algorithmus** besteht aus einer Folge von Supersteps[2], wobei ein Superstep entweder ein Computation-Superstep ist, in dem die Prozessoren ausschließlich Operationen auf lokalen Daten ausführen, oder ein Communication-Superstep, der nur dem Datenaustausch unter den Prozessoren dient. Nach jedem Superstep muß sichergestellt werden, daß jeder Prozessor seine anstehenden Aufgaben erledigt hat, also bei einem Computation-Superstep alle Transformationen auf den lokalen Daten ausgeführt wurden bzw. in einem Communication-Superstep alle Daten, die zum Datenaustausch vorgesehen waren, verschickt und empfangen wurden. Um dies zu gewährleisten, wird programmgesteuert eine Synchronisation eingeleitet. Im Unterschied zum Messagepassing geschieht dies ausschließlich unter Beteiligung *aller*

[2]Die Bezeichnungen Superstep und im weiteren Computation-Superstep, Communication-Superstep, Bulk-Synchronisation und h-Relation wurden aus Ref. [55] übernommen.

Prozessoren und wird daher im Rahmen des BSP-Modells als Bulk-Synchronisation bezeichnet.

Die Abschätzung der Rechenzeit eines Programmes [55] geschieht durch eine getrennte Betrachtung von Computation-Supersteps und Communication-Supersteps. Für beide treten verschiedene die Geschwindigkeit bestimmende Parameter auf. Während der Zeitaufwand für einen Computation-Superstep maßgeblich durch die Rechenleistung s der CPU, meist angegeben in (Mflop/s), und der Anzahl w der auszuführenden Gleitkommaoperationen bestimmt ist, so wird der Aufwand für einen Communication-Superstep bestimmt durch die Zeit g, die aufgewandt werden muß für nicht lokale Speicherzugriffe und deren Anzahl h, zuzüglich der Zeit L für die anschließende Bulk-Synchronisation. Sowohl g als auch L werden in *flop*-Äquivalenten angegeben, also Zeiten, die dem Aufwand einer Gleitkommaoperation entsprechen. Zusammen mit der Prozessorenanzahl P kann die Leistung eines BSP-Computers durch vier Größen spezifiziert werden:

$$\begin{aligned} s &= \text{die Geschwindigkeit eines Prozessors in } [\textit{flop}/\text{s}] \\ L &= \text{der Aufwand für eine Bulk-Synchronisation in } [\textit{flop}] \\ g &= \text{der Aufwand für einen nicht-lokalen Datenzugriff in } [\textit{flop}] \\ P &= \text{Anzahl der Prozessoren} \end{aligned}$$

Der Aufwand eines Communication-Supersteps wird über den Begriff der *h-Relation* definiert als die maximale Anzahl von Dateneinheiten, typischerweise in der Größe einer Gleitkommazahl, die innerhalb eines Communication-Supersteps von einem Prozessor p verschickt h_s^p oder empfangen h_r^p werden:

$$h = \max_{0 \le p < P} \{h_s^p, h_r^p\}. \tag{3.1}$$

Damit ist der Aufwand A_{comm} eines Communication-Supersteps in *flop*-Äquivalenten durch

$$A_{comm} = hg + L \quad [\textit{flop}] \tag{3.2}$$

beschrieben. Finden in dem Superstep noch zusätzlich w lokale Gleitkommaoperationen pro Prozessor statt, so ist aus Sicht eines Prozessors der gesamte Auf-

wand des somit aus w Gleitkommaoperationen, einer h-Relation und einer Bulk-Synchronisation bestehenden Supersteps gegeben durch

$$A = A_{comp} + A_{comm} = w + hg + L \quad [flop] \tag{3.3}$$

und damit die Rechenzeit durch

$$T = A/s \quad \text{sec.} \tag{3.4}$$

Nachdem nun ein Modell sowie das Verfahren zur Rechenzeitabschätzung vorgestellt wurden, sollen die wichtigsten Funktionen der Programm-Bibliothek besprochen werden.

3.3 Die BSP-Funktionen

An dieser Stelle soll keine vollständige Übersicht über sämtliche BSP-Funktionen gegeben werden, sondern es sollen nur diejenigen angesprochen werden, die hauptsächlich bei einer Parallelisierung im Rahmen des SIMD-Modells benötigt werden. Die hier verwendete Notation lehnt sich an die der Programmiersprache "C" an, in der eine Funktion durch ihren Namen, ihre Parameter mit zugehörigen Datentypen, sowie den Wertetyp, den sie zurückliefert, deklariert wird:

$$rückgabetyp \quad funktionsname \quad (typ1 \quad parameter1, \ldots, typN \quad parameterN);$$

Dabei gliedern sich die Funktionen der BSP-Bibliothek in drei Hauptgruppen: Initialisierungsfunktionen, Abfragefunktionen und Funktionen zum Datenaustausch.

3.3.1 Initialisierungsfunktionen

Das Starten und Beenden eines Parallelprogrammes wird explizit durch zwei Funktionen vorgenommen. Durch

void bsp_begin(int P);

wird ein Programm auf P Prozessoren gestartet, während es durch

void bsp_end();

beendet wird. Der Datentyp "void" bezeichnet den "leeren" Datentyp, d.h. die Funktion liefert keinen Wert zurück.

3.3.2 Abfragefunktionen

Während eines Programmablaufs ist es fast immer notwendig, die Anzahl der beteiligten Prozessoren sowie die Identität des lokalen Prozessors zu bestimmen. Hierzu dienen die beiden Funktionen

int bsp_nprocs(); und *int bsp_pid();*

die erste liefert die Anzahl der beteiligten Prozessoren als Ganzzahl, die zweite liefert die Identifikationsnummer des lokalen Prozessors.

3.3.3 Funktionen zum Datenaustausch

Speicherbereiche, also Variablen, auf die nicht nur der lokale Prozessor zugreifen darf, sondern auch die nicht-lokalen Prozessoren, müssen vor den ersten Speicherzugriffen bekannt gegeben werden. Hierzu teilt ein jeder Prozessor den anderen mit, wo bei ihm ein solcher Speicherbereich beginnt und anhand der Bereichsgröße, wo er endet. Die explizite Bekanntgabe des Bereichs geschieht mit Hilfe der Funktion

*void bsp_pushreg (void *ident, int size);*

die als Parameter zum ersten einen Zeiger **ident* auf den Anfang des bekannt zu gebenden Speicherbereichs erhält und zum zweiten dessen Größe *size* in Bytes. Wird ein einmal eingetragener Speicherbereich nicht mehr benötigt, so kann die Bekanntgabe mit

*bsp_popregister(const void *ident);*

widerrufen werden. Der eigentlichen Datenaustausch zwischen nicht-lokalen Speichern geschieht entweder schreibend oder lesend, durch die Funktionen

*void bsp_put(int pid, const void *src, void *dst, int offset, int nbytes);*

die ab der Stelle **src* vom lokalen Speicher *nbytes* an die Stelle **dst + offset* des zum Prozessor *pid* gehörigen, nicht lokalen Speicher kopiert. Entsprechendes leistet die Funktion zum Lesen *void bsp_get(int pid, const void *src, int offset, void *dst, int nbytes);*

die aus dem Speicher eines anderen, durch *pid* spezifizierten Prozessors ab der Speicherzelle **src + offset nbytes* in den lokalen Speicher ab der Adresse **dst* einliest.

Für diese Funktionen zum Datenaustausch existieren noch Varianten *bsp_hpput* und *bsp_hpget*, die eine höhere Performance erwarten lassen (daher "hp" für high performance), auf die hier aber nicht weiter eingegangen wird, da zum einen die Funktionsaufrufe vollkommen analog zu den obigen sind, so daß sie sich im Aufruf lediglich durch ihren Funktionsnamen unterscheiden und zum anderen deckt sich die Semantik der "hp"-Funktionen mit der der nicht optimierten Funktionen.

3.4 Parallelisierung der Propagationsprogramme

In diesem Abschnitt soll besprochen werden, wie Programme zur Lösung der zeitabhängigen Schrödinger-Gleichung parallelisiert werden können. Die größte Schwierigkeit bei der Propagation von Wellenpaketen besteht dabei wieder in der Berechnung von nicht-lokalen Differentialoperatoren, wie den des Impulses oder der kinetischen Energie, siehe Kapitel 2 Gleichung (2.42). Den Kern dieser Operationen bildet eine FFT-Routine, deren Parallelisierung hier vorgestellt wird. Die Hauptaufgabe stellt dabei die Verteilung der Daten auf die Prozessoren und der Datenaustausch unter diesen dar. Für zweidimensionale Problemstellungen, haben sich Standardverfahren für die Datenverteilung etabliert, wie die zyklische Verteilung der Spalten einer

Matrix auf die Prozessoren [58]. Für dreidimensionale Problemstellungen, wie sie in dieser Arbeit auftreten, existiert bisher noch keine Standardlösung, so daß hier eine der Intuition nahekommmende Verteilung entwickelt wurde. Hierbei müssen einige einschränkende Annahmen bezüglich der Anzahl der Datenpunkte in jeder Raumrichtung (N_x, N_y, N_z) sowie der Anzahl P der Prozessoren gemacht werden. Für die vorzustellende Datenverteilung werden drei Forderungen erhoben:

1. die Anzahl der Punkte in jeder Dimension (N_x, N_y, N_z) sei jeweils eine Zweierpotenz [3]

2. die Anzahl P der Prozessoren teile die Anzahl N_z der Gitterpunkte in z-Richtung und ist somit selbst wieder eine Zweierpotenz

3. es existiert eine Faktorisierung von P in P_x und P_y, d.h. $P = P_x \cdot P_y$, wobei P_x und P_y wiederum Teiler der Punktzahlen N_x in x-Richtung bzw. N_y in y-Richtung sind.

Mit Hilfe dieser Annahmen ist es möglich, die im Ortsraum diskretisierte Wellenfunktion in Schichten entlang der z-Achse auf die Prozessoren zu verteilen (siehe Abb. 3.3), wonach jedem Prozessor p nur noch ein Teil Ψ^p der Wellenfunktion zur Verfügung steht. In dieser Verteilung ist es jedem Prozessor unabhängig von den anderen Prozessoren möglich, auf seinem Teil der Wellenfunktion für alle z-Werte lokal eine zweidimensionale Fouriertransformation der x-y-Ebene durchzuführen (siehe Abb. 3.4). Damit ist die Transformation in den Impulsraum jedoch noch nicht vollständig, denn es fehlt noch die Transformierte entlang der z-Achse. Damit nun die Transformation entlang der z-Achse ausgeführt werden kann, müssen sich alle zu einem (k_x, k_y)-Wert in z-Richtung gehörende Daten lokal auf einem Prozessor befinden. Um dies zu erreichen, erfolgt ein Datenaustauch, der alle zu einem (k_x, k_y)-Wert zugehörigen z-Werte sammelt und lokal auf einem Prozessor speichert, so daß jeder Prozessor Säulen der Wellenfunktion enlang der z-Achse erhält. Exemplarisch ist dies für eine Faktorisierung von P in $P = P_x \cdot P_y$ mit $P_x = P_y = 2$, wodurch je zwei Säulen in k_x- und k_y-Richtung entstehen, in Abbildung 3.5 gezeigt. Jetzt kann ein

[3] Diese Forderung ergibt sich aus dem Algorithmus der FFT nach Cooley und Tukey [44], die nur Sequenzen der Länge 2^n behandeln kann.

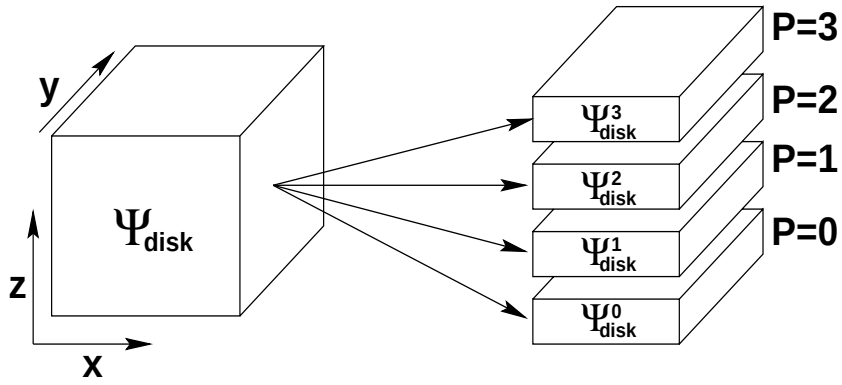

Abbildung 3.3: *Verteilung einer Ortsraum-Wellenfunktion auf 4 Prozessoren*

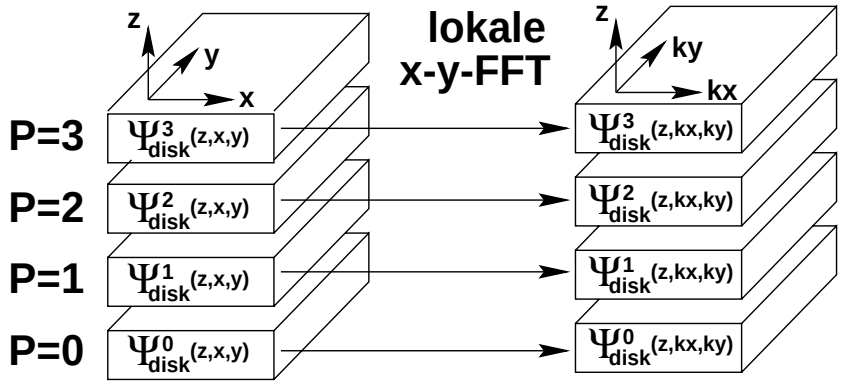

Abbildung 3.4: *Lokale 2D-FFT der Wellenfunktion in der x-y-Ebene.*

jeder Prozessor lokal eine Fourier-Transformation entlang der z-Achse durchführen, um die Wellenfunktion vollständig in den k-Raum zu überführen. Um den bisher mehr intuitiv gefaßten Sachverhalt in ein Programm umsetzen zu können, müssen die schicht- bzw. säulenweise Verteilung der Daten sowie der Wechsel zwischen diesen formal beschrieben werden.

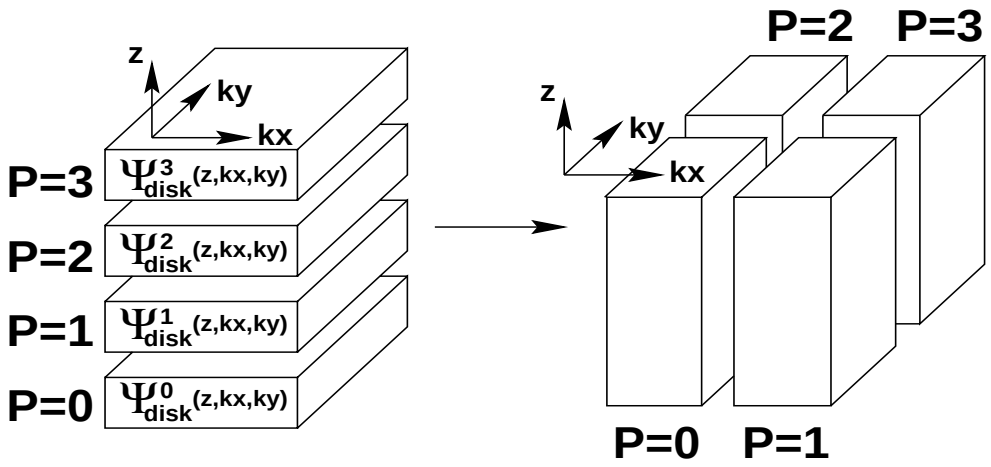

Abbildung 3.5: *Umordnung der Wellenfunktion auf die Prozessoren, wodurch es ermöglicht wird, die Fourier-Transformation entlang der z-Achse durchzuführen.*

3.5 Formale Fassung der Datenverteilung

Für die formale Fassung der Datenverteilung wird ausgegangen von einer diskretisierten Wellenfunktion (wobei an dieser Stelle noch bewußt ausgelassen wird, ob sie im Ortsraum oder im k-Raum diskretisiert ist)

$$\Psi \begin{bmatrix} j \\ k \\ l \end{bmatrix} \quad \text{mit} \quad \begin{array}{l} 0 \le j < N_x \\ 0 \le k < N_y \\ 0 \le l < N_z \end{array} , \tag{3.5}$$

die schicht- bzw. säulenweise auf die Prozessoren verteilt werden soll. Dabei soll die schichtweise Verteilung im folgenden als *D-Verteilung* und die säulenweise als *K-Verteilung* bezeichnet werden, da eine im Ortsraum dargestellte Wellenfunktion in D-Verteilung gebracht wird ("D" aus dem Englischen direkt-space) und eine im Impulsraum dargestellte in K-Verteilung ("K" von k-space). Um die Art der Verteilung anzugeben, erhalten die diskretisierten Wellenfunktionen noch einen linken

oberen Index D oder K. Damit wird eine Verteilung angesehen als eine Abbildung

$$\Psi \begin{bmatrix} j \\ k \\ l \end{bmatrix} \mapsto^{\mathrm{D/K}} \Psi \begin{bmatrix} p/q \\ s \\ t \\ u \end{bmatrix} \tag{3.6}$$

der diskreten Wellenfunktion auf die Prozessoren. Hierzu wird der vorher dreidimensionale Indexraum auf einen vierdimensionalen erweitert, wobei mit den Bezeichnungen aus der Zuordnung 3.6 p bzw. q den Prozessor bezeichnet, der den Wert $\Psi \begin{bmatrix} j \\ k \\ l \end{bmatrix}$ erhält, und diesen mit der Indizierung (s, t, u) in seinem lokalen Segment der Wellenfunktion speichert. Die Verteilung der Wellenfunktion auf die Prozessoren kommt somit einer Indextransformation gleich, wobei der Indexbereich um einen Index p erweitert wird, der den Zielprozessor angibt.

3.5.1 Vorbemerkung zur verwendeten Ganzzahlarithmetik

In der Formulierung der Verteilungen werden die Division mit Rest sowie die Restbildung über den natürlichen Zahlen $\mathbb{N}$ benötigt. Dabei gelten folgende Bezeichungen:

- Division mit Rest: $\div$
- Restbildung: $\mod$

Diese beiden Operationen $\div$ und $\mod$ sind für $a, b \in \mathbb{N}$ definiert durch

$$a \div b := \max_{d \in \mathbb{N}} \{d | d \cdot b \leq a\}, \tag{3.7}$$

und

$$a \mod b := a - (a \div b) \cdot b. \tag{3.8}$$

Damit gilt die oft verwendete Identität

$$a = b \cdot (a \div b) + (a \mod b). \tag{3.9}$$

3.5.2 Die D-Verteilung

Für die schichtweise D-Verteilung einer Wellenfunktion wird die Dicke einer Scheibe in z-Richtung als N_{pz} bezeichnet, sodaß gilt

$$N_{pz} = N_z/P \tag{3.10}$$

Damit stellt die D-Verteilung die folgende Indextransformation D dar:

$$D : \begin{bmatrix} j \\ k \\ l \end{bmatrix} \mapsto \begin{bmatrix} q \\ r \\ j \\ k \end{bmatrix} = \begin{bmatrix} l \div N_{pz} \\ l \mod N_{pz} \\ j \\ k \end{bmatrix} \tag{3.11}$$

mit

$$\begin{array}{l} 0 \le j < N_x \\ 0 \le k < N_y \\ 0 \le l < N_z \end{array} \quad \text{und} \quad \begin{array}{l} 0 \le q < P \\ 0 \le r < N_{pz} \\ 0 \le j < N_x \\ 0 \le k < N_y \end{array} \tag{3.12}$$

Die schichtweise Verteilung der diskretisierten Wellenfunktion auf die Prozessoren wird also durch $D : \Psi[j][k][l] \mapsto^D \Psi[q][r][j][k]$ beschrieben, wobei die Indizes q, r, j, k folgendes bezeichnen:

- $0 \leq q < P$: der Prozessor, der diesen Datenpunkt erhält,
- $0 \leq r < N_{pz}$ der z-Index innerhalb der lokalen Scheibe der Wellenfunktion,
- $0 \leq j < N_x$ der Index in der x-Koordinate,
- $0 \leq k < N_y$ der Index in der y-Koordinate.

Die simple Umordnung der Indizes j, k von den ersten beiden auf die letzten beiden Stellen wurde lediglich vorgenommen, da in der Programmiersprache "C", die zur Formulierung der Programme diente, die Daten so abgelegt werden, daß zuerst über die letzten Indizes variiert wird (in Fortran geschieht dies gerade in der umgekehrten Weise). So liegen in dieser Darstellung die Daten einer zu einem z-Wert gehörigen x, y-Ebene hintereinander im Speicher, wodurch auf diesen effizient eine zweidimensionale FFT durchgeführt wird. Man benötigt zusätzlich zu D auch deren Inverse D^{-1}, mit deren Hilfe später die Transformation zwischen D- und K-Darstellung definiert werden wird.

$$D^{-1} : \begin{bmatrix} q \\ r \\ j \\ k \end{bmatrix} \mapsto \begin{bmatrix} j \\ k \\ l \end{bmatrix} = \begin{bmatrix} j \\ k \\ q \cdot N_{pz} + r \end{bmatrix} \tag{3.13}$$

Einfaches Nachrechnen beweist, daß es sich hierbei tatsächlich um die inverse Ab-

bildung zu D handelt:

$$D^{-1} \circ D \left(\begin{bmatrix} j \\ k \\ l \end{bmatrix} \right) = D^{-1} \left(\begin{bmatrix} l \div N_{pz} \\ l \mod N_{pz} \\ j \\ k \end{bmatrix} \right) \tag{3.14}$$

$$= \begin{bmatrix} j \\ k \\ (l \div N_{pz}) \cdot N_{pz} + (l \mod N_{pz}) \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix} \tag{3.15}$$

3.5.3 Die K-Verteilung

Für die säulenweise Verteilung wird ausgegangen von einer Faktorisierung der Prozessorenanzahl $P = P_x \cdot P_y$, wodurch P_x Säulen entlang der x-Achse und P_y Säulen entlang der y-Achse entstehen. Dabei kommen zu der Bezeichnung N_{pz} aus der D-Verteilung die folgenden hinzu:

- $N_{px} = N_x / P_x$: Die Anzahl der Punkte einer Säule in x-Richtung,
- $N_{py} = N_y / P_y$: Die Anzahl der Punkte einer Säule in y-Richtung.

Die K-Verteilung wird dann über die Indextransformation K definiert:

$$K : \begin{bmatrix} j \\ k \\ l \end{bmatrix} \mapsto \begin{bmatrix} p \\ m \\ n \\ l \end{bmatrix} = \begin{bmatrix} P_x \cdot (k \div N_{py}) + (j \div N_{px}) \\ j \mod N_{px} \\ k \mod N_{py} \\ l \end{bmatrix} \tag{3.16}$$

mit

$$\begin{array}{c} 0 \le j < N_x \\ 0 \le k < N_y \\ 0 \le l < N_z \end{array} \quad \text{und} \quad \begin{array}{c} 0 \le p < P \\ 0 \le m < N_{px} \\ 0 \le n < N_{py} \\ 0 \le l < N_z \end{array} \tag{3.17}$$

in der die Indizes p, m, n, l folgende Zuordnung erfahren:

- p : der Prozessor, der den Datenpunkt erhält,
- m : x-Koordinate innerhalb der lokalen Säule,
- n : y-Koordinate innerhalb der lokalen Säule,
- l : z-Koordinate.

Auch zu K benötigt man die inverse Abbildung K^{-1}:

$$K^{-1} : \begin{bmatrix} p \\ m \\ n \\ l \end{bmatrix} \mapsto \begin{bmatrix} j \\ k \\ l \end{bmatrix} = \begin{bmatrix} N_{px} \cdot (p \mod P_x) + m \\ N_{py} \cdot (p \div P_x) + n \\ l \end{bmatrix}. \tag{3.18}$$

Wiederum erhält man den Beweis, daß K^{-1} wirklich die Inverse zu K ist durch simples Nachrechnen:

$$K^{-1} \circ K \begin{bmatrix} j \\ k \\ l \end{bmatrix} = K^{-1} \begin{bmatrix} P_x \cdot (k \div N_{py}) + (j \div N_{px}) \\ j \mod N_{px} \\ k \mod N_{py} \\ l \end{bmatrix} \tag{3.19}$$

$$= \begin{bmatrix} N_{px} \cdot [(P_x \cdot (k \div N_{py}) + (j \div N_{px})) \mod P_x] + (j \mod N_{px}) \\ N_{py} \cdot [(P_x \cdot (k \div N_{py}) + (j \div N_{px})) \div P_x] + (k \mod N_{py}) \\ l \end{bmatrix} \tag{3.20}$$

wegen $[P_x \cdot (k \div N_{py})] \mod P_x = 0$
und $[P_x \cdot (k \div N_{py})] \div P_x = k \div N_{py}$ gilt

$$= \begin{bmatrix} N_{px} \cdot [(j \div N_{px}) \mod P_x] + (j \mod N_{px}) \\ N_{py} \cdot [(k \div N_{py}) + ((j \div N_{px}) \div P_x)] + (k \mod N_{py}) \\ l \end{bmatrix} \tag{3.21}$$

und unter Berücksichtigung von $(j \div N_{px}) \mod P_x = (j \div N_{px})$ (wegen $j \div N_{px} < P_x$) sowie $(j \div N_{px}) \div P_x = 0$ erhält man die gesuchte Identität:

$$K^{-1} \circ K \begin{bmatrix} j \\ k \\ l \end{bmatrix} = \begin{bmatrix} N_{px}(j \div N_{px}) + (j \mod N_{px}) \\ N_{py}(k \div N_{py}) + (k \mod N_{py}) \\ l \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix}. \tag{3.22}$$

Durch die Transformationen D und K kann nun eine diskretisierte Wellenfunktion, wie in Abb. 3.6 gezeigt, in die entsprechenden Verteilungen gebracht werden. Der bislang noch nicht bestimmte Wechsel zwischen den Verteilungen wird im folgenden betrachtet.

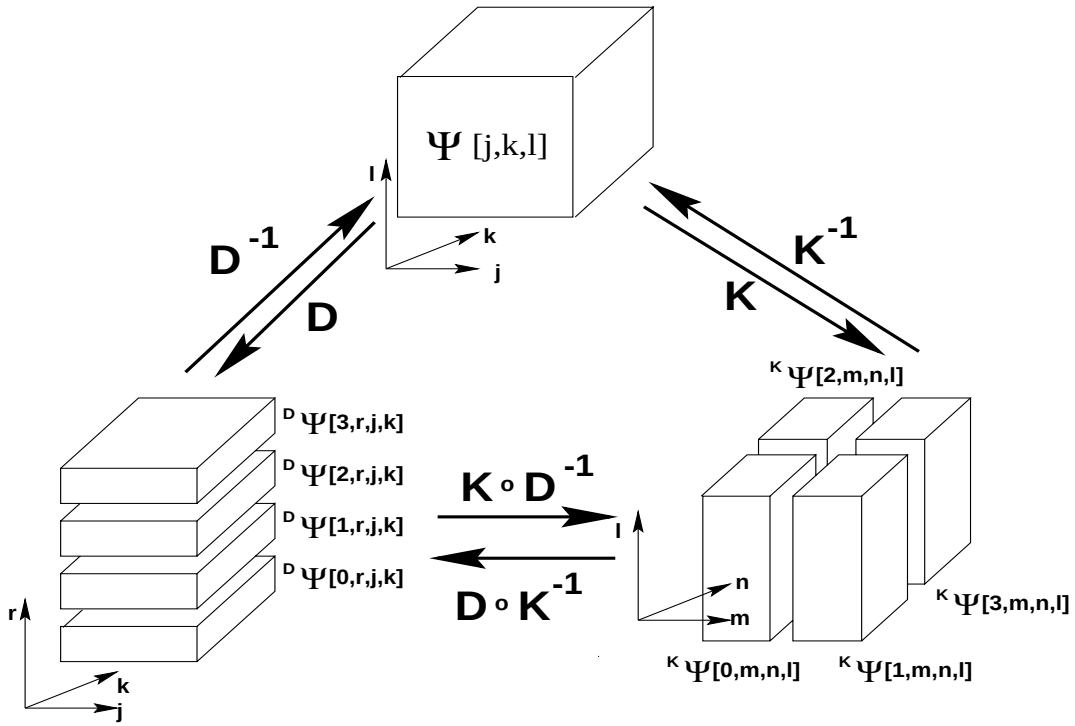

Abbildung 3.6: *Verteilung einer Wellenfunktion in K- und D-Darstellung mit den entsprechenden Transformationen. In diesem Beispiel sind $P = 4$ Prozessoren beteiligt, wobei eine Faktorisierung von $P = P_x \cdot P_y$ mit $P_x = P_y = 2$ verwendet wurde.*

3.5.4 Wechsel zwischen den Darstellungen

Wie in Abschnitt 3.4 besprochen, muß man zur parallelen Berechnung der FFT zwischen K- und D-Verteilung transformieren zu können. Da man die Abbildungen D und K mit ihren Inversen kennt, kann die Transformation T_{DK} von der D-Verteilung in die K-Verteilung durch die Hintereinanderausführung von zuerst D^{-1} gefolgt von K berechnet werden und es gilt: $T_{DK} = K \circ D^{-1}$. Entsprechend geschieht dies für die Transformation in umgekehrter Richtung von der K- in die D-Verteilung durch

$T_{KD} = D \circ K^{-1}$. Durch Ausmultiplikation von $T_{DK} = K \circ D^{-1}$ erhält man:

$$T_{DK} : \begin{bmatrix} q \\ r \\ j \\ k \end{bmatrix} \mapsto \begin{bmatrix} p \\ m \\ n \\ l \end{bmatrix} = \begin{bmatrix} P_x(k \div N_{py}) + j \div N_{px} \\ j \mod N_{px} \\ k \mod N_{py} \\ q \cdot N_{pz} + r \end{bmatrix} \tag{3.23}$$

mit

$$\begin{array}{ll} 0 \leq q < P & 0 \leq p < P \\ 0 \leq r < N_{pz} & 0 \leq m < N_{px} \\ 0 \leq j < N_x & 0 \leq n < N_{py} \\ 0 \leq k < N_y & 0 \leq l < N_z \end{array} \quad \text{und} \quad . \tag{3.24}$$

Entsprechend ergibt sich die Rücktransformation $T_{KD} = D \circ K^{-1}$ als

$$T_{KD} : \begin{bmatrix} p \\ m \\ n \\ l \end{bmatrix} \mapsto \begin{bmatrix} q \\ r \\ k \\ j \end{bmatrix} = \begin{bmatrix} l \div N_{pz} \\ l \mod N_{pz} \\ N_{px} \cdot (p \mod P_x) + m \\ N_{py} \cdot (p \div P_x) + n \end{bmatrix} \tag{3.25}$$

Nachdem nun sowohl die verschiedenen Verteilungen der Daten (D und K) als auch die Transformationen (T_{DK} und T_{KD}) zwischen diesen formal gefaßt sind, soll nun der parallele Programmablauf von der anfänglichen Datenverteilung über die FFT bis hin zur Propagation von Wellenpaketen besprochen werden. Für die weiteren Ausführungen ist es zweckmäßig, sowohl die Wellenfunktionen sowie die Operatoren in Anlehnung an eine Programmiersprache zu beschreiben. So werden für die Wellenfunktionen, die auf einem Gitter $N_x \cdot N_y \cdot N_z$ diskretisiert sind, und deren Verteilung auf $P = P_x \cdot P_y$ Prozessoren folgende Konstrukte eingeführt:

- *complex PSI[N_x][N_y][N_z]*

- *complex DPSI[N_{pz}][N_y][N_x]*
- *complex KPSI[N_x/P_x][N_y/P_y][N_z]*

Dabei bezeichnet *complex* den Wertebereich des folgenden Feldes, dessen Größe durch den darauf folgenden Indexbereich angegeben ist. Desweiteren soll PSI die gesamte Wellenfunktion darstellen, die jedoch nie als ganzes abgespeichert wird, sondern entweder von einer Datei gelesen oder punktweise berechnet wird. Somit wird PSI hier nur für die folgenden Ausführungen eingeführt und liegt während des Programmablauf ausschließlich als Verteilung $DPSI$ bzw. $KPSI$ auf den Prozessoren vor. $DPSI$ soll die schichtweise Verteilung von PSI auf die Prozessoren bezeichnen, was schon daran zu erkennen ist, daß der erste Index in der z-Koordinate nicht mehr N_z sondern bei P Prozessoren nur noch die Scheibendicke N_z/P ist. Durch $KPSI$ wird in der säulenweisen Verteilung ein Segment der Wellenfunktion beschrieben. Auch hier ist die Art der Verteilung an der Indizierung zu erkennen, an der Säulenbreite $N_{px} = N_x/P_x$ in x-Richtung und $N_{py} = N_y/P_y$ in y-Richtung. Für beide Verteilungen gilt, daß ein Segment stets die Größe $N_x \cdot N_y \cdot N_z/P$ besitzt, wodurch alle P Segmente zusammen wieder die ursprüngliche Größe der Wellenfunktion erreichen.

3.5.5 Verteilung der Wellenfunktion auf die Prozessoren

Die Verteilung der Wellenfunktion auf die Prozessoren soll nur vom Prozessor mit der Identität $p = 0$ vorgenommen werden. Dabei werden die Werte $PSI[j][k][l]$ der Wellenfunktion für $0 \leq j < N_x$, $0 \leq k < N_x$ und $0 \leq l < N_x$ vom Prozessor $p = 0$ berechnet und mit Hilfe der Transformation D auf die anderen Prozessoren verteilt, indem für jeden Wert (j, k, l) der Zielprozessor q und die lokale Indizierung (r, j, k) innerhalb dessen Segments $DPSI$ der Wellenfunktion berechnet werden. Damit geschieht die Datenverteilung in folgender Weise:

- Jeder Prozessor erfragt bei Programmstart seine Identität p mit Hilfe der Funktion *bsp_pid*

- Jeder Prozessor gibt mit Hilfe der Funktion *bsp_pushregister* sein Segment $DPSI$ der Wellenfunktion zum Lesen bzw. Schreiben frei.
- Nur der Prozessor $p = 0$ führt für alle (i, j, k) folgendes aus:
  - Berechne (oder lese von Datei) für gegebenes (i, j, k) den Wert der diskretisierten Wellenfunktion.
  - Bestimme durch die Transformation D (Gleichung (3.11)) den Prozessor q, der diesen Wert erhalten soll, sowie die Indizierung (r, j, k) in dessen Segment $DPSI$.
  - Schreibe mit Hilfe der Funktion *bsp_put* diesen Wert in den Speicher des Prozessors q an die Stelle $DPSI[r][j][k]$
- Alle Prozessoren führen zum Schluß durch Aufruf der Funktion *bsp_sync*() eine Synchronisation durch.

Die Verteilung der Daten funktioniert vollkommen analog für die K-Verteilung.

3.5.6 Berechnung der 3D-FFT

Um nun die 3D-FFT zu berechnen betrachtet man nur noch die Segmente $DPSI$ und $KPSI$ der Wellenfunktion, die den Prozessoren vorliegen. Die Berechnung der 3D-FFT nach Bekanntgabe der Speicherbereiche $DPSI$ und $KPSI$ gestaltet sich nun folgendermaßen:

- Jeder Prozessor führt lokal auf seinem Segment $DPSI[r][j][k]$ eine zweidimensionale FFT über den Indizes j, k durch.
- Die Daten werden umgeordnet in die säulenweise Verteilung, indem jeder Prozessor für alle Werte (r, j, k) innerhalb seines Segments der Wellenfunktion und mit Hilfe seiner Identität q durch die Transformation T_{DK} (Gleichung (3.5.4)) den Zielprozessor p berechnet, der diesen Datenpunkt an der Stelle $KPSI[m][n][l]$ seines lokalen Segments speichert.

- Die Transformation der Daten wird durch eine Synchronisation aller Prozessoren abgeschlossen
- Jeder Prozessor führt die noch fehlende Fourier-Transformation über dem Index l seines Segments $PSI[m][n][l]$ durch.

Mit diesen Ausführungen ist die Berechnung der inversen 3D-FFT auch klar, es muß in obiger Vorgehensweise stets nur rückwärts transformiert werden und T_{DK} durch T_{KD} ersetzt werden.

3.5.7 Propagation von Wellenpaketen

Für die Propagation der Wellenpakete soll hier exemplarisch von dem einfacheren Propagationsschema (2.47) ausgegangen werden. An dieser Stelle werden noch die Zeitentwicklungsoperatoren U_d^{pot} und U_k^{kin} (siehe Gleichungen (2.50) und (2.50)) benötigt, für die jeweils wieder entsprechende Variablen angelegt werden:

- *complex UPOT[N_z/P][N_x][N_y]*
- *complex UKIN[N_x/P_x][N_y/P_y][N_z]*

Wie aus diesen Variablendeklarationen erkennbar ist, werden die diskretisierten Operatoren in unterschiedlicher Weise auf die Prozessoren verteilt, nämlich der Zeitentwicklungsoperator U_d^{pot} der potentiellen Energie wird als *UPOT[N_z/P][N_x][N_y]* in D-Verteilung gespeichert, während der Zeitentwicklungsoperator U_k^{kin} der kinetischen Energie in K-Verteilung als *UKIN[N_x/P_x][N_y/P_y][N_z]* abgelegt wird. Die Verteilung der beiden Zeitentwicklungsoperatoren auf die Prozessoren erfolgt analog der in Abschnitt 3.5.5 beschriebenen Vorgehensweise für die Wellenfunktionen. Sind die Verteilungen der Wellenfunktion und der Zeitentwicklungsoperatoren erst einmal erledigt, so stellt sich eine parallel ablaufende Propagation folgendermaßen dar:

- lokale punktweise Multiplikation von $DPSI[r][k][j]$ mit $UPOT[r][k][j]$

- parallele 3D-FFT wie unter 3.5.6 beschrieben
- lokale punktweise Multiplikation von $KPSI[m][n][l]$ mit $UKIN[m][n][l]$
- parallele Rücktransformation der 3D-FFT

Damit ist die Hauptaufgabe der parallelen Propagation gelöst. Bemerkenswert ist hierbei, daß sich die unterschiedliche Darstellung von Wellenfunktionen und Operatoren im Orts- oder Impulsraum auf die Datenverteilung überträgt. Orts- und Impulsraum entsprechen der D- bzw. K-Verteilung. Die Anwendung von Operatoren ist für Impuls-Operatoren nur in der K-Verteilung diagonal und damit in der parallelisierten Version lokal, während Operatoren der potentiellen Energie nur in der D-Verteilung von den Prozessoren lokal berechnet werden können. In anderen Worten: In der kanonischen Verteilung eines Operators läuft dessen Anwendung auf eine Wellenfunktion sowohl punktweise (wegen der Diagonalität) als auch lokal (im Hinblick auf die Parallelverarbeitung) ab.

Die Parallesierung ist damit im wesentlichen auf die Berechnung der Abbildungen D, K, T_{DK} sowie deren Inversen zurückgeführt.

3.6 Rechenzeitbedarf

Es bleibt noch die Frage nach dem Rechenzeitbedarf der hier vorgestellten Algorithmen. Um den Rechenzeitbedarf abschätzen zu könne, benötigt man zuerst die Anzahl w von Gleitkommaoperationen, die innerhalb einer Rechnung durchgeführt werden. Dazu wird ein Propagationsschritt betrachtet, in dem die Wellenfunktion punktweise mit dem Zeitentwicklungsoperator der potentiellen Energie U_d^{pot} multipliziert wird, gefolgt von einer 3D-FFT, an die sich die punktweise Multiplikation mit U_k^{kin} anschließt. Geschlossen wird der Propagationsschritt durch eine Rücktransformation mit Hilfe der inversen 3D-FFT. Die Anzahl der Gleitkommaoperationen ergibt sich somit als die Summe der Gleitkommaoperationen innerhalb der FFT zuzüglich der Anzahl von Gleitkommaoperationen, die für die punktweise Multiplikation der Wellenfunktionen mit den Operatoren benötigt werden. Für eine

FFT der Länge $N = N_x \cdot N_y \cdot N_z$ nach dem Radix-2 Algorithmus [44, 59] werden genau $5N \cdot \mathrm{ld}(N)$ Multiplikationen benötigt, wobei $\mathrm{ld}(N)$ den Logarithmus dualis, also zur Basis 2, bezeichnet. Für die punktweise Multiplikation der Wellenfunktionen mit den Zeitentwicklungsoperatoren benötigt man jeweils genau N komplexwertige Multiplikationen, was 6 Gleitkommaoperationen (4 Multiplikationen und 2 Additionen) entspricht. Es werden also benötigt für

- Eine FFT: $5N \cdot \mathrm{ld}(N)$ Gleitkommaoperationen
- Die lokale Multiplikation mit einem Operator: $6N$ Gleitkommaoperationen

und somit für einen aus 2 FFTs sowie zweimaliger Anwendung eines Operators bestehenden Propagationsschritt:

- Ein Propagationsschritt: $10N \cdot \mathrm{ld}(N) + 12N$ Gleitkommaoperationen

Das in Abschnitt 3.2 vorgestellte Modell zur Aufwandsbestimmung benötigt die Anzahl der pro Prozessor ausgeführten Gleitkommaoperationen. Da diese vollkommen gleichmäßig auf die P Prozessoren verteilt sind, muß jeder Prozessor für einen Propagationsschritt genau $w = (10N \cdot \mathrm{ld}(N) + 12N)/P$ Gleitkommaoperationen ausführen. Desweiteren muß der Datenaustausch berücksichtigt werden. Hierfür muß jeder Prozessor die Wellenfunktion zweimal umverteilen, wobei sämtliche Daten ausgetauscht werden müssen. Da das lokale Segment einer Wellenfunktion aus N/P komplexwertigen Punkten besteht, also aus $2N/P$ Gleitkommazahlen, verschickt auch jeder Prozessor genau diese Anzahl $h = 2N/P$ in einem Communication-Superstep. Da für einen Propagationsschritt die Datenumverteilung genau zweimal stattfindet, verschickt jeder Prozessor $2 \cdot h$ Gleitkommazahlen, wobei der Aufwand für das Versenden einer Gleitkommazahl dem Aufwand von g Gleitkommaoperationen gleichkommt. Nach jeder Umverteilung der Daten muß eine Synchronistation stattfinden, die zeitlich äquivalent zu L Gleitkommaoporationen ist. Damit läßt sich der Aufwand eines

für P Prozessoren parallelisierten Propagationsschrittes beziffern auf

$$A_{propa} = w + 2 * h * g + 2 * L = (10N \cdot \mathrm{ld}(N) + 12N)/P + g \cdot 4N/P + 2 \cdot L \tag{3.26}$$

Für eine Cray T3D wurde mit einem von Prof. R. Bisseling entworfenen Analyseprogramm die folgenden Geschwindigkeitsparameter ermittelt:

1. $s = 6.5 Mflop/s$ pro Prozessor
2. $g = 20 flop$
3. $L = 100 flop$

Das Programm wurde an Reproduktionsrechnungen des Pump-Probe-Spektrums von Na_3 [52] getestet, wobei ein Gitter der Größe $N = 64 \cdot 64 \cdot 32 = 131072$ verwendet wurde. Man erkennt, daß bei dieser Gittergröße und bei obigen Werten für g und L der Aufwand maßgeblich durch den Teil $(10N \cdot \mathrm{ld}(N) + 12N)/P$ bestimmt ist und man somit einen fast linearen Anstieg der Rechenleistung mit wachsender Prozessorenanzahl erwarten kann. Wie Abb. 3.7 zeigt wurde dies auch erreicht. Die dabei ermittelte Rechenleistung liegt mit 7.4 $Mflop/s$ pro Prozessor über der, die mit Hilfe eines Performance-Tools von R. Bisseling ermittelt wurde[4]. Diese könnte jedoch noch gesteigert werden, falls eine für die Cray-T3D optimierte eindimensionale FFT-Routine eingesetzt würde, auf deren Einsatz hier jedoch aus Portabilitätsgründen verzichtet wurde. Eine weitere Möglichkeit zur Performance-Verbesserung, wäre auf Cray-spezifische Routinen für den Datenaustausch zurückzugreifen, worauf aber ebenfalls aus Gründen der Portabilität verzichtet wurde. Insgesamt läßt sich jedoch festhalten, daß ein annähernd linearer Anstieg der Rechenleistung erhalten wurde, wobei die Rechenleistung über der Vorhersage des Performance-Tools liegt.

[4]Ein Performance-Tool kann natürlich nie genaue Werte vorhersagen, da die Rechenleistung immer von der Problemstellung und der Struktur des Codes abhängt. Das heißt, diese Abweichung bewegt sich im normalen Bereich und ist weder auf einen besonders optimalen Code noch auf falsche Werte des Performance-Tools zurückzuführen

Abbildung 3.7: *Anstieg der Rechenleisung in Abhängigkeit von der Anzahl der beteiligten Prozessoren*

Der in diesem Kapitel vorgestellte Algorithmus zur Wellenpaketpropagation wird im weiteren immer verwendet, sobald mehrdimensionale Wellenpaketdynamik berechnet werden soll. Die durch die Parallelisierung erzielte Zeitersparnis macht sich besonders bemerkbar, wenn man algorithmisch optimale Laserfelder zur Kontrolle der Molekulardynamik bestimmen möchte, wozu in einem iterativen Schema vielfache Wellenpaketpropagationen notwendig sind. Wie nun optimale Laserfelder zur Quantenkontrolle bestimmt werden können, soll Thema des folgenden Kapitels sein.