Lösungen für anspruchsvolle Probleme bei massiver Parallelisierung eines quantenchemischen Simulationssystems

Zheng-Yu Jiang
Institute of parallel and Distributed High Performance Systems(IPVR),
University of Stuttgart,
Breitwiessenstraße 20-22, 70565 Stuttgart, Germany.
Tel: 0049-711-7816445, fax: 0049-711-7816424,
Email: zhengyu@informatik.uni-stuttgart.de

1. Einleitung

Die quantenchemischen Rechnungsmethoden erfordern nicht nur sehr viele Rechenzeit, sondern stellen auch an die Haupt- und Plattenspeicher extreme Anforderungen. Aufgrund des Umfangs und der komplexität des MOLPRO (etwa. 420000 Programmzeilen) ist die Parallelisierung sehr arbeitsaufwendige Aufgabe. Da in den verwandten Methoden sehr große Datenmenge bei arithmetischen Operationen in parallei weise kombiniert werden müssen, stellt extreme Aufforderungen an die Datenkommunikation, Lastbalancierung und Ein-/Ausgabe. Wir werden zuerst die quantenchemischenrechnungen kurz vorstellen, Daher werden die anspruchsvolle Probleme sowie die Lastungleichkeit, der Komunikationsaufwand und parallele Ein-/Ausgabe in diesen Anwendungssystemen erklärt.

2.1 Quantenchemische Rechnungen

Die meisten Ziele der Quantenchemie sind zur Berechnung von der elektrischen Wellenfunktionen und Reaktionsenergie durch angenäherte Lösungen der zeitunabhängigen Schrödingergleichungen $\hat{H}\Psi = E\Psi$. Die unbekannten Eigenfunktionen, Ψ, dieser Gleichung werden in Form eines Satzes von N-Partikel Basisfunktionen (configuration state functions, CSF's) ausgedrückt, die ihrerseits als eine Linearkombination von Atomorbitalen (AO's) angenähert werden.

2.2 Anspruchsvolle Probleme

Die herkömmliche Probleme bei quantenchemischen Rechnungen sind den Rechenaufwand für verschiedenen Elektronenpaar-Typen extrem unterschiedlich, so daß bei der Parallelisierung die Lastbalancierungsproblematik besonders beachtet werden muß. Engpässe durch den Kommunikationsaufwand und begrenzten I/O-, Platten- und Hauptspeicherkapazitäten können ein Anwendungsprogramm in eine schlechte gesamte Leistung führen.

2.2.1 Lastungleichkeit

Die wichtige Frage hierbei sind, wie man die „*Meta-Loop*" nach den unabhängigen Elektronpaaren optimal partitionieren kann, damit die Berechnungsaufgaben möglichst gleichmäßig auf den Parallelrechner verteilt werden kann, und wie man die Kommunikationszeit, die Synchronisationszeit und der Aufwand für die gleichmäßige Verteilung der Rechenlast bei den massiv-paralleien Berechnungen minimieren. Wir können daraus sehen, daß die Lastbalancierung eine wesentliche Rolle in der parallelen und verteilten Berechnung spielt. Da das Potential von parallelen und verteilten Systemen soweit wie möglich nur ausgenutzt werden kann, wenn die gesamte Systemlast auf alle verfügbaren Prozessoren gleichmäßig verteilt werden kann. Die Grundidee besteht darin, daß man ein optimales Partitionierungsverfahren in der iterativen Lösungsverfahren quantenchemischer Rechnung verwenden kann. Aus dieser Grundidee wurde ein iterativer Lastbalancierungalgorithmus entwickelt, mit dem solche Anwendungsprogramme effizient auf parallelen und verteilten Rechnern ausgeführt werden können.

2.2.2 Kommunikationsaufwand

Bei der parallelen Berechnung eines Operators benötigt eine globale Summierungs- und Verket-

tungs-Operation über allen Prozessoren durchgeführt, um die Beiträge von allen Elektronpaaren von jedem Prozessor auf berechneten Operatoren sich zu belaufen.

Das Verfahren solcher globalen Message-Operationen ist, jeder Prozessor sendet eine Nachricht und am Ende der globalen Message-Operationen bekommt jeder Prozessor die Nachrichten von allen anderen Prozessoren. Daher stellt sich die wichtige Frage, wie solche globale Message-Operationen effizient auf verschiedenen Parallelrechnern ausgeführt werden können, insbesondere angesichts der Randbedingung, daß *MOLPRO* Paket nicht nur auf *distributed Memory* sondern auch auf *shared Memory* Parallelrechnern unter Verwendung derselben Parallelverabeitungskonzepte effizient ablaufen soll. Die Abhängigkeit der parallelen Effizienz von der Kommunikationstopologie des verwendeten Rechnersystems nimmt in den letzten Jahren glücklicherweise stark ab, darf jedoch bei der Auswahl des Kommuinkationsalgorithmus noch nicht völlig ignoriert werden.

2.2.3 Parallele Ein-/Ausgaben

Die quantenchemischen Rechnungen erfordern sehr viel Ein- und Ausgabe. Bei der konventionellen Quantenchemischen Rechnungen werden häufig Matrix-Matrix- oder Matrix-Vektor-Multiplikationen in Dateien zwischengespeicherten Größen wie z. B. den Zweielektorenenintegralmatrizen und den Koeffizientenmatrizen ausgeführt. Aus N Basisfunktionen werden $O(N^4)$ Zweielektronenintegrale erzeugt. Bereits für mittelgroße Molekülsysteme kann dies sehr leicht zu einer 2-Elektronenintegraldatei in der Größe von mehreren Giga-Byte führen. In den letzten 20 Jahren wurden massiv-parallele Rechner entwickelt, die bei niedrigen Kosten große Verarbeitungskapazitäten zur Verfügung stellen. Da eine parallele Ein-/Ausgabe in der Regel nicht verfügbar ist, reicht die I/O-Bandbreite bei weitem nicht aus.

3. Lösungsansätze zur Probleme

Wir werden uns in folgenden Abschnitten mit der Lösungen für oben erwährte Probleme beschäftigen. Dabei werden ein neuer iterativer selbstadaptiver Lastbalancierungsalgorithmus, Shared Array-Algorithmus für globale Message-Operation und einen Global-Array-I/O-Mechanismus präsentiert.

3.1 Lastbalancierung

Die Lastverteilung oder Lastbalancierung spielt eine wichtige Rolle bei parallelen quantenchemischen Rechnungen. Die meistens Rechenzeit z.B. im *MOLPRO* wird für die Multiplikation der Integralmatrix mit der Koeffizientenmatrix benötigt. Aber eine wichtige Eigenschaft ist, daß die zu einem *Elektronenpaar oder Symmetry-Unique-Shell* gehörenden Matrizen unabhängig von anderen *Elektronenpaaren oder Symmetry-Unique-Shells* sind. Deshalb kann man die folgende drei Lastverteilungsschemen verwenden.

3.1.1 CTSS (*Cyclical Task Scheduling Scheme*)

In diesem Schema wird eine statische Lastverteilung durch die Aufgabeverteilung realisiert. Mit diesem Schema kann die Last nach der Elektronpaarliste auf die verfügbaren Prozessoren verteilt werden. Bild 1 zeigt diesen statischen Last-Verteilungsalgorithmus.

```
P = number_of_nodes()
my_id=node_id()
Do task=1, ... , N
    Do (global work)
        node = task mod P
        if (my_id = node) then
            Do OwnWork
        endif
    EndDo(global work)
EndDo(Task)
```

Bild 1:Das zyklische Lastverteilungsschema (CTSS).

3.1.2 GRR (*Global Round-Robin*) dynamisches Last-Verteilungsschema

Im GRR dynamischen Lastverteilungsschema steht ein globaler Zähler zur Verfügung. Wenn ein Prozessor seine gegenwärtige Aufgabe fertig bearbeitet hat, greift er die nächste Aufgabe durch Aufruf der Funktion nxtask() (oder nxtval()) wie ein globaler Zähler. Diese gelesene globale Zahl bezeichnet eine entsprechende Task-Nummer einer Taskliste. Der globale Zähler wird nach jedem Aufruf eins erhöht. Allerdings darf gleichzeitig nur ein Prozessor den globalen Zähler lesen.

3.1.3 Selbstadaptiver Lastbalansierungsalgorithmus (SALBA)

Dieser Algorithmus ist passend nur an Anwendungsklassen, die iterative Methoden zur Lösung der Systemgleichungen verwenden. Die selbstadaptive Lastverteilung verteilt auf iterative Weise die Systemlast gleichmäßig auf alle verfügbaren Prozessoren. Offensichtlich wird die Performance des von diesem Lastverteilungsschema eingesetzten Anwendungsprogramms auch auf iterative Weise verbessert. Dieses Schema besteht aus zwei Teilen [Jiang97] [Jiang98/3]:

- Iterative, selbstadaptive, eindimensionale, mehr-stufige und geradlinige Partitionierung

Die Last im Anwendungssystem wird entweder durch aufwendige Tasks (Aufgaben) oder durch umfangreiche Daten verursacht. In *MOLPRO* werden die überwiegenden Lasten durch die zeitaufwendige Matrixmultiplikation erzeugt, die in verschachtelten Schleifen (nested loops) ausgeführt werden. Durch eine mehrstufige Partitionierung dieser Matrizen wird eine effektive Bearbeitung solcher Berechnungsaufgaben auf Parallelrechnern ermöglicht. Dafür wird die adaptive 1D geradlinige Last-Partitionierung in der weiteren iterativen Berechnungen fortgesetzt. Da die Berechnung in iterative Weise ausgeführt wird, benutzten wir die Lastinformation und die Partitionierungsinformation der letzten iterativen Berechnungen bei der 1D geradlinigen Partitionierung, um die optimale Lastverteilung erreichen oder annähern zu können.

- Gruppen-Partitionierung des Prozessors

Im SALBA wird eine Gruppen-Partitionierung des Prozessors durchgeführt, damit jede Prozessorgruppe einen Teil der gesamten Aufgabe behandeln kann. Normalerweise erfordert am Ende einer parallelen Bearbeitung, um eine oder mehre globale Message-Operation sowie die globale Verkettung und die globale Summierung durchzuführen, damit die Beiträge jedes Prozessors zur resultierenden Matrix kombiniert bzw. addiert werden können. Die Lastverteilung erfordert den Austausch von Lastinformation zwischen Prozessoren, außerdem hängt der Aufwand der globalen Message-Operationen von der Anzahl der beteiligten Prozessoren ab. Wenn alle verfügbaren Prozessoren in Gruppen partitioniert werden, kann dieser Kommunikationsaufwand deutlich reduziert werden.
Dieses Lastbalancierungsschema verteilt sowohl Daten als auch Aufgabe auf die verfügbaren Prozessoren, deshalb zeigt es ein gutes Laufzeitverhalten. Die ausführliche Beschreibung dieses Schemas und Laufzeitsergebnisse siehe [Jiang97].

3.1.4 Evaluierung des Balancierungsalgorithmus anhand quantenchemischer Simulation

Zur Bewertung wird die oben erwähnten Lastbalancierungsalgorithmen ins MRCI-Programm des MOLPRO integriert. Die Implementierungsplattforme waren CRAY T3D bzw. T3E und SGI Power-Challenge in Stuttgart. Die Rechenzeit des externen Austauschoperators mit CCSD- oder MRCI-Methoden liegen in Höhe von 83-96% der gesamten CPU-Zeit [Jiang98/3]. Außerdem skalieren die Rechenkosten der 2-Elektronenintegrale mit $O(N^4)$ und bei der Transformation in der molekularen Orbital-Basis mit $O(mN^4)$, wobei N die Zahl der Basis-Funktionen, und m die Anzahl besetzter Orbitale bezeichnet. Die Rechenkosten des externen Austauschoperators betragen $O(m^2) \bullet O(N^4)$ [Jiang98/2].

Die Berechnungsform des externen Austauschoperators lautet:

$$K(C^{ij})_{\mu\nu} = \sum_{\rho\sigma} C^{ij}_{\rho\sigma} \langle \mu\rho|\sigma\nu\rangle \qquad (1)$$

wobei C^{ij} die Matrix der Koppelungskoeffizienten ist, und $\langle \mu\rho|\sigma\upsilon\rangle$ bezeichnet die Zweielektronenintegrale. (i,j) stellt ein Elektronenpaar dar.

Für dieses Berechnungsmodell können wir für die parallele Ausführung eine zweistufige Partitionierung verwenden. Die erste Stufe ist die Gruppenpartitionierung von C^{ij}, in welcher jede Prozessorgruppe einen Block der C^{ij} Matrix erhält. Diese Partitionierung hängt davon ab, wie viele zu behandelnde Einträge die C^{ij} Matrix enthält, und wie viele Prozessoren sich an dieser Berechnung beteiligen. Die zweite Stufe ist die iterative selbstadaptive 1D geradlinige Partitionierung der 2-Elektronenintegrale in der parallelen Matrix-Matrix-Multiplikation zur parallelen Berechnung des externen Austauschoperators.

Wir werden die Ergebnisse einer Benchmark-Berechnung auf der CRAY T3E mit 512 PEs darstellen.

Als ein Benchmark wurde das Hexatrien-Mole-

kül mit MRCI-Methode berechnet. Dabei keine räumliche Symmetrie wurde ausgewählt. Für den Vergleich wurde die dynamische Lastbalancierungsalgorithmen, nämlich SALBA, GRR und CTSS, in das parallele quantenchemische Rechnungsprogramm integriert.

Häufig erreicht ein paralleler Algorithmus eine akzeptable Performance, wenn nur bis zu einer beschränkten Anzahl von Prozessoren (z.B. bis zu hundert) bei Ausführung eines hinreichend großen Anwendungsprogramms verwendet wird. Um eine theoretische Skalierbarkeit von Anwendungen unter *SALBA* real zu demonstrieren, wurde *MOLPRO* mit *SALBA* auf einer CRAY T3E mit 512 Prozessoren ablaufen lassen. Abbildung 2 belegt, daß *SALBA* den besten *Speedup* im Vergleich zu den anderen zwei Algorithmen aufweist. Bei der hier verwendete Berechnung des Hexatriene-Moleküls mit 100 kontraktierten Basisfunktionen und 256 Elektronpaaren kann mithilfe von *SALBA* ein Speedup von 400 auf einer T3E mit 512 Prozessoren erzielt werden, während die anderen Lastbalancierungsalgorithmen bzw. GRR und CTSS lediglich einen Speedup ca. von 317 erreichen. Dies ist heute für parallelen quantenchemischen Rechnungen, hier sogar mit relativen kleinen Basisfunktionen (100 Basisfunktionen), ein sehr gutes Ergebnis. Wird das Berechnungsproblem mit mehr als 100 Basisfunktionen berechnet, ist eine noch bessere parallele Performance zu erwarten. Aus Speicherplatzgründen werden auf einer CRAY-T3E mindestens 32 Prozessoren benötigt, um dieses Berechnungsproblem mit 100 Basisfunktionen durch *MOLPRO* lösen zu können.

3.2 Die globale Verkettung-Operation

Für die Kommunikationsprobleme stellen wir hier einen Shared-Array-Algorithmus für die globale Verkettung-Operation als einen veranschaulichenden Beispielen zur Lösung der Kommunikationsprobleme in internen Verbindungsnetzwerken vor. Für die erganzende Untersuchung wird auf [Jiang98/1,Jiang98/3] verwiesen.

3.2.1 Definition

Zu Anfang der globalen Verkettungs-Operation erhält jeder Prozessor einen Vektor von N Elementen, z.B. besitzt Prozessor j einen Vektor $x_j(i)$, i=1,2,...,N. Es existieren $P_1, P_2, ... P_P$, von P Prozessoren. Der resultierende Vektor A hat die

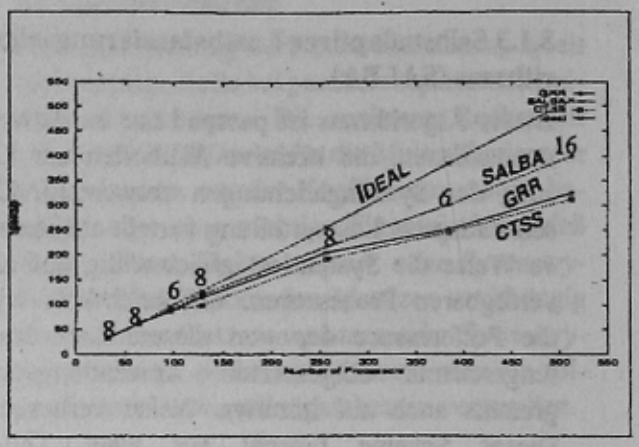

Bild 2: Vergleich zum Speedup der drei Lastbalancierungalgorithmen auf der CRAY T3E/ 512 für verschiedene Anzahl von Prozessorgruppen (6,8,16).

Größe von NP Elementen und soll ein so geordneter Vektor sein:

$$((A = \{x_1[i] | x_2[i] | x_3[i] | ... | x_P[i]\}) , \forall i, i = 1, 2, ..., N); \quad (2)$$

wobei „|" die Verkettung anzeigt.

Jeder Prozessor schickt eine eigene Message an alle anderen Prozessoren und der empfangende Prozessor wird eine Message nach Gleichung 2 ordnen. Zum Ende der globalen Verkettung-Operation wird jeder Prozessor den resultierten Vektor A erhalten haben. Deshalb verhält sich die gesamte zu kombinierende Messagegröße zu der selbst erhaltenen Messagegröße und der Anzahl der Prozessoren, welche an dieser globalen Verkettungsoperation beteiligt sind, wie $Size(A) = P \bullet N$.

3.2.2 Shared-Array-Algorithmus

Ziel des Shared-Array-Algorithmus ist, daß er sowohl für Rechner mit verteiltem Speicher als auch für Rechnern mit gemeinsamem Speicher geeignet sein soll, um portierbar zu gewährleisten. Alle Knoten werden in einem virtuellen Feld (*shared-array*) organisiert, jeder Knoten erhält N Elemente der zu kombinierenden Message. Vor allem wird das *Shared-Array* durch die Benutzung des *Global-Array Toolkit* [NieHar94], über den Aufruf der Subroutine ga_create(type, dim1, dim2, array_name, chunk1, chunk2,g_a), erzeugt. Danach wird an jedem Knoten die eigene Message im lokalen

Shared-Array durch die ga_put() Subroutine abgelegt. Wenn alle Knoten diese Operation zum Ende ausgeführt haben, wird die Kombination-Operation automatisch im *Shared-Array* fertiggestellt. Somit werden die Kombinationskosten gespart. Anschließend kann jeder Knoten die kombinierten NP Byte-Message durch Aufruf einer fernen Get-Operationen mit der ga_get() Subroutine erhalten. Um den Konflikt während der Get-Operation bei gemeinsamen Daten zu vermeiden, organisiert ein sogenanntes Pipelining-Get-Verfahren den Zugriff auf das globale Array. In jedem Get-Schritt wird jeder Prozessor aus den gemeinsamen Daten seinen jeweils gewünschten Teil eines globalen-Arrays abholen. Falls dieser Teil gleich groß und die Get-Geschwindigkeit auf jedem Prozessor gleich ist, kann der Netzwerk-Konflikt absolut vermieden werden.

Dieser Algorithmus bringt den Vorteil, daß die Kombinationsoperation lokal automatisch durchgeführt werden kann, so daß die Kombinationskosten nicht benötigt werden. Aber es steht der Nachteil dagegen, daß jeder Knoten eine Fernzugriffsoperation durchzuführen hat, um die kombinierte Message zu erhalten. Wenn in derselben Zeit mehr als einen Knoten eine Shared-Daten abholen möchte, wird ein Konflikt auftreten. Es gibt auch zusätzlichen Aufwand beim Erzeugen und Zerstörungen eines *Shared-Arrays*. Die Performance der globale Verkettungs-Operation kann deshalb einen Verlust erleiden, vorallem wenn die Kosten beim Erzeugen des *Shared-Arrays* relativ groß sind, z.B. in dem Fall, wenn die zu kombinierende Message auf der *Power-Challenge* Maschine bei der All-to-all Kommunikation die zu kombinierende Messag relativ klein ist (siehe das Ergebnis in folgendem Abschnitt).

3.2.3 Evaluierung globaler Verkettungs-Algorithmen

Die Evaluierung der globalen Verkettungs-Algorithmen wurde durch Vergleich mit anderen Algorithmen bzw. den Torus-Algorithmus und Baum-Algorithmus durchgeführt. Weil diese Torus-Algorithmus und Baum-Algorithmus sind in vieler Anwendungsprogramme verwendet worden. Abbildung 3 zeigt zwei logische Netz-

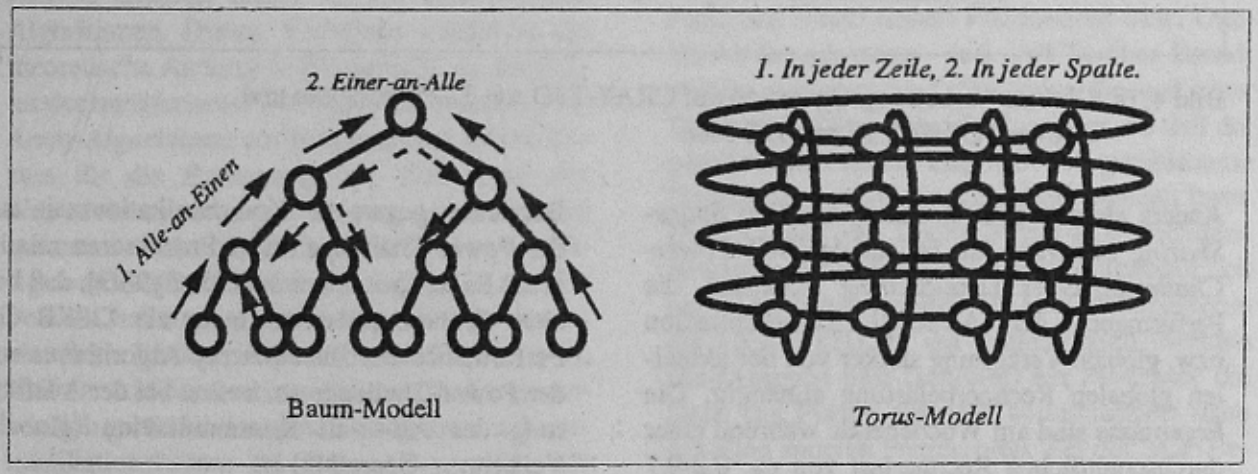

Bild 3:Die Topologie der Torus- und Baum Modelle.

werk-Modelle der Baum- und Torus-Algorithmus. Für die ausführliche Beschreibung dieser zwei Algorithmen siehe [Jiang98/3, Jiang98/1]. Die drei Algorithmen wurden auf der CRAY-T3D/E und SGI-Power-Challenge implementiert. Die Ergebnisse sind in den Abbildungen 4 (a) und 4 (b) dargestellt, und sind mit der von links nach rechts ansteigenden Messagegröße skaliert werden. Alle Implementierungen wurden mit TCGMSG, einem portablen *Message-Passing-Packet*, aufgebaut, um die Performance der verschiedenen Algorithmen fair vergleichen zu können. In den Abbildungen wird mit *Tree* der Baum-Algorithmus gekennzeichnet. Aus Abbildung 4(a) sehen wir, daß auf der T3D der *Shared-Array-Algorithmus* die beste Performance hat. Die offensichtliche Ursache ist, daß die effektive durchschnittliche Bandbreite der *Remote-Get*-Operation von 35 MB/s und *der Lokal-Put-Operation von* 140 MB/s mit *Shared-Array-Algorithmus* erreicht worden sind. Diese Bandbreite ist mehr als 45 MB/s, die von ande-

ren Algorithmen erschließt wurde. Anderseits bietet die T3D eine schnellere Verbindungsarchitektur für Informationsteuerung und einen Datendurchsatz mit einer Spitzentransferrate von 300 MB/s (auf der T3E von 500 MB/s) an jedem Knoten, so daß der Aufwand für die Erzeugung des *Shared-Arrays* auf der T3D vernachlässigt werden kann. Außerdem bietet die CRAY T3D eine relativ kleine Latenzzeit von ca. 17 ns beim Zugriff auf den gemeinsamen Speichern während der Ausführung der *Put- und Get-Operationen*. Wir erkennen aus der theoretischen Analyse [Jiang981], daß der *Shared-Array-Algorithmus* die kleinste Amplitude der transferierten Message von $O((P-1)N)$ hat. Das ist einen wichtigen Faktor, der die *Overall-Performance* einer globalen Verkettung-Operation beeinflußt.

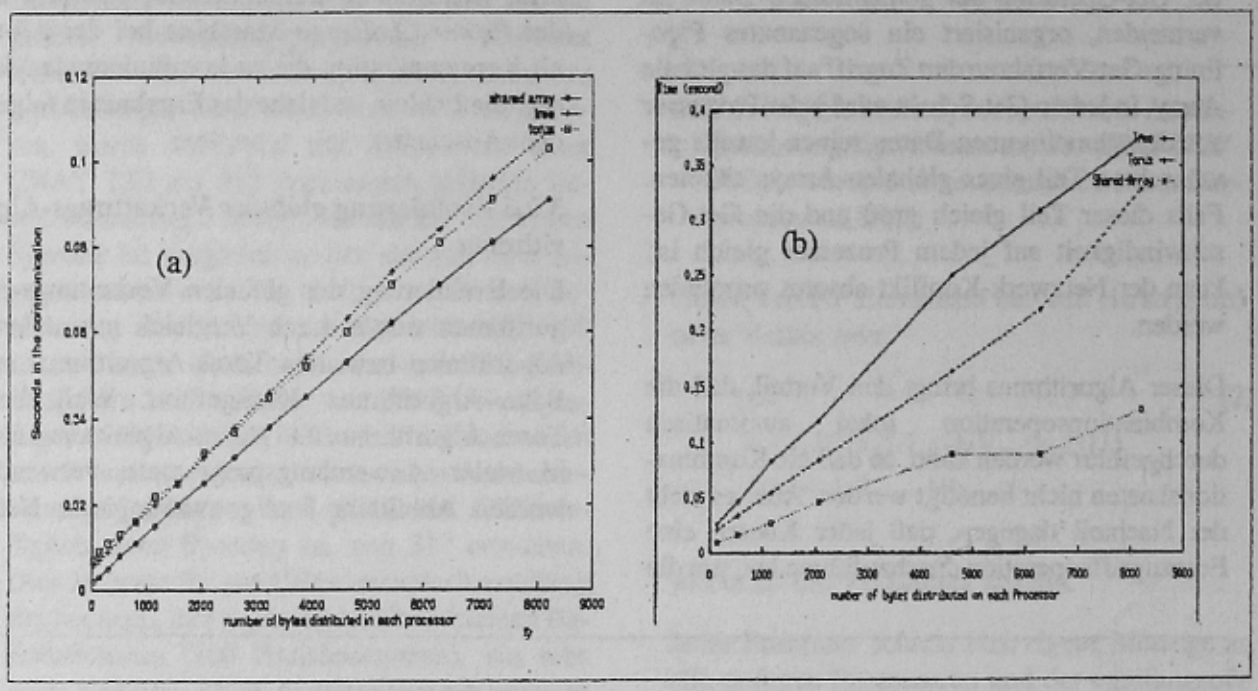

Bild 4: (a) Globale Verkettung-Operation auf CRAY-T3D mit 32 Prozessoren und
(b) auf CRAY-T3E mit 128 PEs.

Anders als der T3D Rechner, der eine *Space-Sharing* Maschine ist, ist auf der SGI *Power-Challenge*, einer *Time-Sharing* Maschine, die Performance der All-to-all Kommunikation bzw. globale Verkettung stärker von der aktuellen globalen Rechnerbelastung abhängig. Die Ergebnisse sind am Wochenende während einer durchschnittlichen Rechnerlast von ca. 0.2-0.7 erzielt worden. Die dabei festgestellte Zeitabweichung ist für kleine Messagegrößen von weniger als 2 Kbyte nicht sehr deutlich. Dagegen ist die Zeitabweichung bei größerer Message ganz signifikant. Der Torus-Algorithmus zeigt eine beste Performance für die All-to-all Kommunikation im Problembereich vom größer als 2 KByte auf der SGI Power-Challenge, wie in Abbildung 6 gezeigt wird. Beim Shared-Array-Algorithmus kann man eine beste Performance nur in einem kleinen Problembereich der All-to-all Kommunikation feststellen. Der Aufwand für das Erzeugen eines Shared-Arrays ist in anderen Bereichen gegen zur Kommunikationszeit auf der Power-Challenge mit 8 Prozessoren relativ groß. Es ist aber interessant [Jiang98/3], daß bei einer Problemgröße von mehr als 128KB die Performance des Shared-Array-Algorithmus auf der Power-Challenge am besten bei der Ausführung der All-to-all Kommunikation (globale Verkettung-Operation) ist.

Da die Kommunikationsperformance im internen Verbindungsnetzwerk nicht nur von der Größe der übertragenen Message sondern auch stark von der Anzahl der Prozessoren, welche an der Kommunikation teilnehmen, abhängt, führen wir das Kommunikationsprogramm auf der T3E mit 128 PEs aus, um daran feststellen zu können, wie die verschiedene Algorithmen der globalen Verkettung-Operationen sich bei einem Parallelrechnern mit der mittelgroßen Anzahl des Prozessors verhalten. Aus Abbildung 4 (b) ersehen wir, daß der *Shared-Array-Algorithmus* auf der

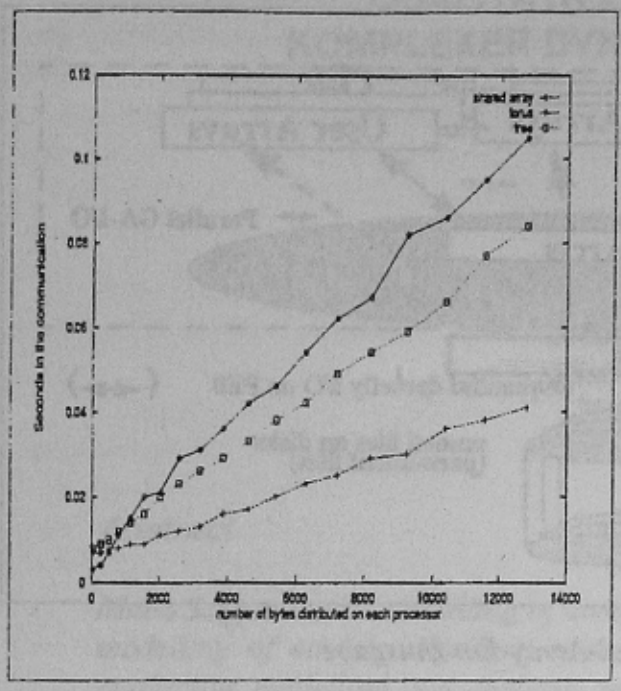

Bild 5: Globale Verkettungs-Operation auf SGI Power-Challenge mit 8 Prozessoren.

T3E sogar mit 128 PEs die beste Performance liefert. Die Abbildung 4(b) zeigt besonders, daß bei zunehmender Größe der übertragenen Message die Performance des *Shared-Array-Algorithmus* deutlich besser ist bei allen anderen Algorithmen. Dieses Verhältnis entspricht der theoretische Analyse in [Jiang98/3], da der Faktor der transferierten Messaggröße beim *Shared-Array-Algorithmus* am kleinsten ist. Und die Kosten für die Erzeugung und Zerstörung des *Shared-Array* auf der T3E kann gegenüber den gesamten Kommunikationskosten vernachlässigt werden (Details siehe [Jiang98/1]). Auf der Power-Challenge weist der Torus-Algorithmus eine beste Performance auf (siehe Abbildung 5), weil der Aufwand auf der *Power-Challenge* wesentlich große im Vergleich zur gesamten Kommunikationskosten ist. Besonders wollen wir darauf aufmerksam machen, daß in den Anwendungsprogrammen die Erzeugung und die Zerstörung eines virtuellen *Shared-Arrays* nur einmal benötigt wird, jedoch mehr als zehn- sogar hundertmal die globale Message-Operation im Shared-Array bei einem Anwendungsprogramm durchgeführt werden kann. Deshalb braucht dieser Aufwand auf der CRAY T3D/E nicht besonders berücksichtigen zu werden.

3.3 Parallel-Global-Array-I/O

Heutzutage stehen viele parallele Dateisysteme (z. B. PIOFS von IBM, HFS von der Universität Toronto, PPFS von der Universität Illinois) und parallele Ein-/Ausgabe-Bibliotheken (z. B. Panda von der Universität Illinois, MPI-IO vom MPI-IO Committee und MPI-2 vom MPI Forum) zur Verfügung [Stock98][Kotz97]. Diese sind jedoch noch nicht standardisiert und ihre Leistungsfähigkeit erscheint noch nicht ausreichend, um die „Grand-Challenge Probleme" effizient zu lösen [WomGre97]. Der Ein-/Ausgabe-Flaschenhals beschränkt die gesamte Performance der Anwendungsprogramme.

Bei der Parallelisierung des quantenchemischen Simulationssystems MOLPRO wurde ein paralleler Ein-/Ausgabe-Mechanismus, der sogenannte „Parallel-Global-Array-I/O" entwickelt. Er basiert auf dem *Global-Array-Toolkit*. Die zahlreichen 2-Elektronenintegrale werden damit bei der Berechnung nicht auf Platte sondern im Hauptspeicher, z B. im sogenannten *Global Array* abgespeichert.

Bild 6 auf Seite 8 zeigt diesen Ein-/Ausgabe-Mechanismus. Durch die Benutzung dieses Mechanismus wird der Zugriff auf eine temporäre Datei bei den Global-Arrays parallel durchgeführt, während der Zugriff auf eine permanente Datei bei Platten durch die Kommunikation zwischen Prozessor 0 und andere Prozessoren bzw. One-To-All-Broadcasting und All-To-One-Broadcasting realisiert wird. Für jeden Record einer Datei wird ein globales Feld erzeugt, so daß der parallele I/O durch Zugriffe auf gemeinsame Hauptspeicherbereiche ausgeführt werden kann. Die I/O Zeit des Global-Array-I/O-Mechanismus ist ca. 22% kleiner als die vergleichbare Zeit bei einem konventionellen I/O auf 4 Prozessoren der CRAY-T3D. In der sequentiellen direkten Ein-/Ausgabe liest (schreibt) ein Prozessor die Datei aus Platten ein, und kommuniziert dann mit allen anderen Prozessoren. Bei der SCF-Berechnung kann dadurch durchschnittlich 36% der Laufzeit auf der CRAY-T3D eingespart werden.

4. Konklusionen

Die Lösungen für anspruchsvolle Probleme bei massiv-parallelisierung eines quantenchemischen Simulationssystems wie die unzufriedene Leistungsfähig der parallelen Ein-/Ausgabe, Lastungleichkeit, Komunikationsaufwand wurden diskutiert. Dabei wurden die entsprechen Algorithmen bzw. der Mechanismus vorgestellt, und einige untersuchte Ergebnisse präsentiert.

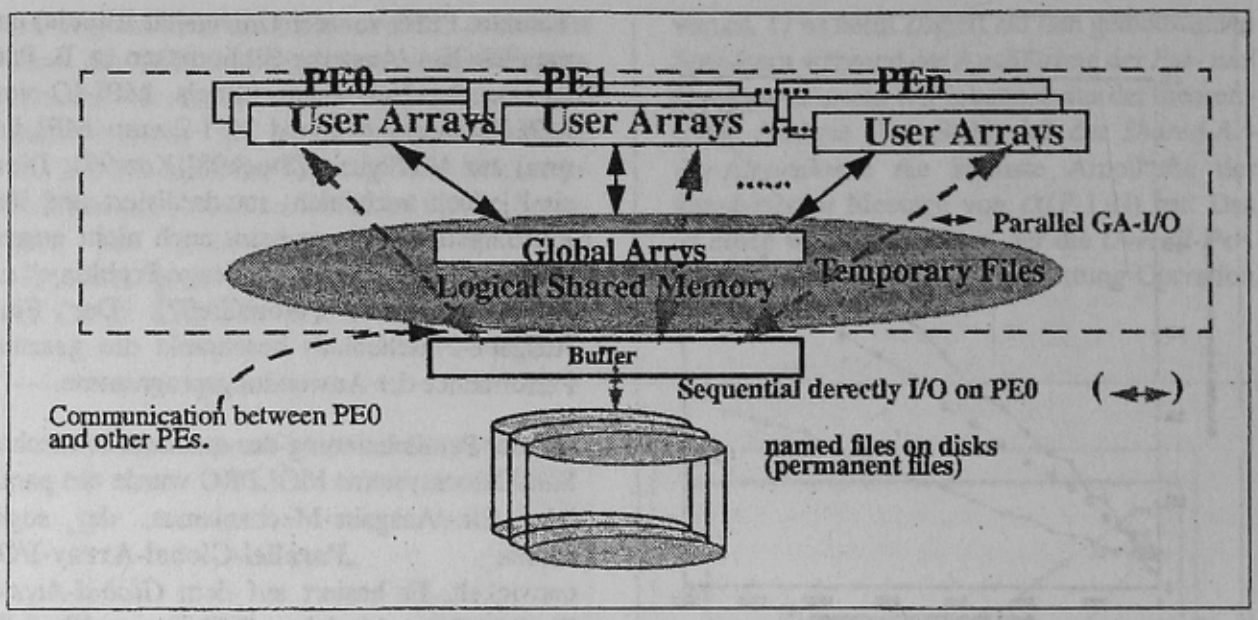

Bild 6: Parallele Global-Array-Ein-/Ausgabe

Literaturverzeichnissen

[Jiang97] Zhen-Yu Jiang, A Self-Adaptive 1D Multi-Degree Rectilinear Partitioning for Adaptive Load Balancing in Massiv Parallel and Distributed Computing. Proceedings of International Conference of Parallel and Distributed Processing Techniques and Applications (**PDPTA'97**), Las Vegas, USA, 1997, Vol. 1, P422-431.

[Jiang98/1] Zheng-Yu Jiang, An Efficient Global Concatenation Algorithm on Distributed and Shared Memory Supercomputers. Proceedings of International Conference of Parallel and Distributed Processing Techniques and Applications (**PDPTA'98**), Las Vegas, USA, 1998, Vol. 2, P835-842.

[Jiang98/2] Zheng-Yu Jiang, Wolfgang Bekker, H-J Werner, and Andreas Reuter, A Grouping Scheme for Parallel Evaluation of External Exchange Operators in Correlated ab initio Methods, Proceedings of International Conference of Parallel and Distributed Processing Techniques and Applications (**PDPTA'98**), Las Vegas, USA, 1998, Vol. 4, P1806-1809.

[Jiang98/3] Zheng-Yu Jiang, Parallelism, Performance Optimization and Load Balancierung in Quantum Chemistry Simulation on Distributed and Shared memory supercomputers, Dissertation, 1998, university of Stuttgart, Germany.

[MolPro98] MOLPRO is a package of ab initio programs written by H.-J. Werner and P. J. Knowles, with contributions from R. D. Amos, A. Berning, D. L. Cooper, etc.

[DobKno97] A. Dobbyn, P.J. Knowles and R. J. Harrison, Parallel Internally Contracted Multireference Configuration Interaction, J. Comp. Che., in press.

[GreMc92] David S. Greenberg and K.S. McCurley. All-to-all broadcasting on Mesh Connected Parallel Computers. Technical Report at Sandia National Laboratories. 1992.

[NieHar94] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, Proceedings of Superconputing 1994 (**IEEE** Computer Sociaty Press, Washington, DC, 1994.

[Kotz97] D. Kotz, Parallel I/O bibliography, http://www.cs.dartmounth.edu/pario/bib.html

[WomGre97] David E. Womble and David S. Greenberg, Parallel I/O: An introduction, Parallel Computing 23 (1997) 403-417.

[Stock98] Heiz Stockinger, Classification of Parallel Input/Output Products, Proceedings of International Conference of Parallel and Distributed Processing Techniques and Applications (**PDPTA'98**), Las Vegas, USA, 1998, Vol. 1, P469-476.