ハイパフォーマンス
コンピューティング 55－6
（1995．3．9）

非経験的分子軌道法プログラム(AMOSS)の並列化

平原幸男† 望月祐志‡ 高田俊和‡
中田宏† 土谷正人†

NEC情報システムズ†
NEC基礎研究所‡

非経験的分子軌道法は、分子系の電子状態、化学反応性を理論的に調べる為の有力な手段であるが、その基本近似であるハートリー-フォック(HF)法においてさえ、展開基底関数の4乗のオーダーの積分を扱う必要がある。従って、生体分子など軌道数が千以上に達するような大型の系をルーチン的に解くためには分散・並列化による高速処理が不可欠である。私たちは、こうしたことから自主開発のソフトAMOSSを軸にして分子軌道計算の並列化に取り組んでいる。この報告では、WSクラスター及びMPP(Cenju-3)上での並列化HF計算の結果について述べるが、共に16台構成時に約15倍の加速を得ている。

Parallelized *ab initio* Molecular Orbital Method Program (AMOSS)

Yukio HIRAHARA† Yuji MOCHIZUKI‡ Toshikazu TAKADA‡
Hiroshi NAKADA† Masato TSUCHIYA†

NEC Informatec Systems†
NEC Fundamental Research Laboratories‡

Ab initio molecular orbital (MO) methods are promising to investigate the variety of electronic structures and/or chemical reactivities for the molecular systems in the balanced fashion. However, their computational efforts are considerably demanding. Even the standard Hartree-Fock procedure requires the N^4 order processings related with two-electron integrals, where N is the number of basis functions. To solve the systems having more than thousand functions routinely, the acceleration through parallelizations is crucial. Thus we have been developing the parallelized MO calculations based on the AMOSS package. In this report, we discuss the HF parallelizations on the clustered WS's and the MPP (Cenju-3). In the case of 16 nodes, the acceleration is observed to be 15 for both types of parallel environment, indicating a satisfactory scalability of our coarse-grained algorithm.

1 はじめに

非経験的分子軌道 (MO:molecular orbital) 法は、分子系の電子状態、化学反応性を理論的に調べる有力な手段の1つであり、100 %自主開発のプログラムパッケージ AMOSS を通じて、ハートリーフォック (HF:Hartree-Fock)、その解析微分 (HF-gradient) による分子構造最適化、及び電子相関計算として2次 (MP2:second order Møller-Plesset) 摂動論の機能を提供しており、配置間相互作用 (CI:configuration interaction) や多配置 HF も現在開発中である。

ここで、演算量の観点から分子軌道計算を捉えてみると、小量の入力データ (分子構造や基底関数) から、膨大な量の中間作業データ (基本的に保存の必要はない) の処理を経て再び小量の出力データ (エネルギーや1電子密度) を得るという特徴がある。ここで中間データとは、基底関数の総数を N として、HF では N^4オーダ、電子相関計算では N^5オーダ以上もの演算量を要求する2電子積分と言い換えられる。実際、積分の処理は1ショット計算の全 CPU 時間の 90 %以上を占めており、高速化の対象そのものである。

さて、単一ノード処理速度の限界が (ベクトル化を持ってしても) 認識された昨今の高速大規模計算 (HPC:high performance computing) の流れにおいては、並列処理・分散処理が基幹要素を成していることは言うまでもなく、非経験的分子軌道計算の高速化もこれからは並列処理の分脈の中で考えていかなければならないのは明らかである。こうしたことから、私たちは AMOSS を軸にした並列化分子軌道計算に取り組んでいる。プラットフォームとしては、ベクトル CPU をノードとする HPP、RISC ベースの MPP、及び WS クラスター、さらにこれらを複合したメタシステム (metasystem) までを幅広く想定している。アルゴリズム的には、上記の分子軌道計算の特徴を活かした2電子積分を並列化のキーにする粗粒度の積分駆動 (integral-driven) 型に分類され、その制御はメッセージ通信によるマスター／スレーブ (あるいは、クライアント／サーバ) 式に属する。現在想定している目標は、HF レベルで数千基底以上の生体分子、半導体結晶モデルなど、電子相関レベルで相関軌道総数が～五百程度の大規模問題のルーチン化 ("実時間処理"化) である。

本稿では、第1段階として AMOSS の HF モジュールを並列化し、代表的なメッセージ通信ライブラリ PVM でリンクした WS クラスター、及び MPP 型並列専用機 Cenju-3 上で試行した結果を報告する。構成は以下の様である。2章では HF 法の概説と処理の流れを、3章では並列化 HF のスキームを詳細に論じる。4章では実行結果を示すが、スケーラビリティは 16 ノード時に 15 倍の加速が得られている。

2 HF 法

HF は分子軌道法の中では"標準"の近似法であり、定式化の基本は「分子系の電子波動関数を単一の Slater 行列式で記述する」ことである。行列式は反対称化された分子軌道とスピンの積

$$\Psi_{\mathrm{HF}} = |\psi_1\bar{\psi}_1\psi_2\bar{\psi}_2\psi_3\bar{\psi}_3.... > \qquad (1)$$

であり、個々の分子軌道は基底関数によって展開

$$\psi_i = \sum_p c_{pi}\chi_p \qquad (2)$$

されているので、手続きとしては分子軌道の正規直交条件を付して式 (1) の行列式の与えるエネルギーが停留になるように展開係数行列 c を決めてやればよく、代数的には一般化固有値の対角化問題

$$\mathbf{fc} = \mathbf{sce} \qquad (3)$$

に帰着される。この式中で、f がいわゆる Fock 行列 (系のエネルギーの表現行列)、s が基底関数間の重なり、及び e が軌道エネルギーを含む対角行列である。さて、f は1電子積分 (運動エネルギーと核引力) と2電子積分 (電子間の反発) から成るが、並列実行による高速化の対象とすべきは N^4

オーダ量である2電子積分項の処理であるのは1章で既に述べたとおりである。演算としては、個々のFock行列要素 f_{pq} に電子密度行列要素 d_{pq} と2電子積分要素 g_{pqrs} を掛けて足し込む操作

$$f_{pq} \leftarrow f_{pq} + d_{rs} * g_{pqrs} \tag{4}$$

である。ここで断らねばならないのは、密度が式(3)を解いて決まるべき展開係数と依存関係

$$d_{pq} = 2\sum_{i} c_{pi}c_{qi} \tag{5}$$

を持つために。式(3)の固有値方程式は実は非線形であるという点である。つまり、HFの実際の数値解法では、適当な初期見積りから始めて、係数行列cがもはや(閾値内で)変化しなくなりエネルギーが一定値に収束するまで、2電子積分と密度行列からFock行列を構成して繰り返し解かねばならない。この反復過程中では電子密度のみが変化し、積分値は不変であることに注意されたいが、HFのこの性格を用いて積分を「毎回生成する」か「適当に使い回す」ことに応じた幾つかの積分ハンドリング手法を適用することが出来る。

3 並列化HFスキーム

3.1 レシピ

既に述べているように、並列化のターゲットは繰り返し行われる式(4)の2電子積分のFock行列要素への足し込み操作であるが、個々の操作は全く独立に実行可能なので、積分を指定するキー *pqrs* を並列化の変数とし、スレーブタスク側で"部分"Fock行列を作り、マスタータスク側で"完全"Fock行列を

$$\mathbf{f} = \sum_{partial} \mathbf{f}_{partial} \tag{6}$$

で作るレシピが単純率直である。この式中で、スレーブに対する記法 *partial* は積分キーのリストに対応し、これが積分駆動並列化の由来となっている。なお、1電子積分要素はマスタータスク側で"完全"行列に最初(もしくは最後)に加算しておけば良い。

3.2 データの配置

各反復中で、式(6)の並列化実行は、メッセージ通信に基づいて進められる。ここで、通信の対象となるデータが密度行列と"部分"Fock行列であるのは自明である。この他に、積分値の実際の計算など準備のための作業用行列群も必要であり、これら一連の行列領域は、分散メモリ環境ではそのまま各ノード毎に、また共有メモリ環境でもノード毎に分割してバッファリングして確保する必要がある。後者の共有メモリ環境で、参照のみの密度行列に対してもバッファリングを行っているのは、メモリアクセス競合による影響を少しでも少なくする為であるが、これは同時に「利用可能ノード数分だけのメモリ領域が必要となる」という負の面も合わせ持ってはいる。

3.3 並列動作の詳細

さて、分散メモリ環境の代表例であるWSクラスターでの実行を例にして並列化HFの動作を述べる。図1はメッセージ通信による動作流れのブロック図である。まず、マスタータスクがスレーブタスク群を生成(spawn)し、次いで積分生成のキーパラメータ *LID* などを個別に送信する。ここからがHFの反復ループとなり、詳しく挙げると

(a) マスターが密度行列をスレーブ側に一括送信(multicast)

(b) 各スレーブ側が密度行列を受信、そして"部分"Fock行列領域をクリア

(c) 続いてリストに応じて積分寄与を加算

(d) 全加算処理終了後に"部分"Fock行列をマスターに送信

(e) マスターは"完全"Fock行列を復元

が進められる。これ以下(d)と(e)をまとめてリダクション(reduction)処理として記している。この後、式(3)の一般化固有値問題を解いて新しいcとエネルギーを得て収束を判定し、もし収束していれば反復をブレークし、そうでなければ再び(a)に戻って繰り返す。

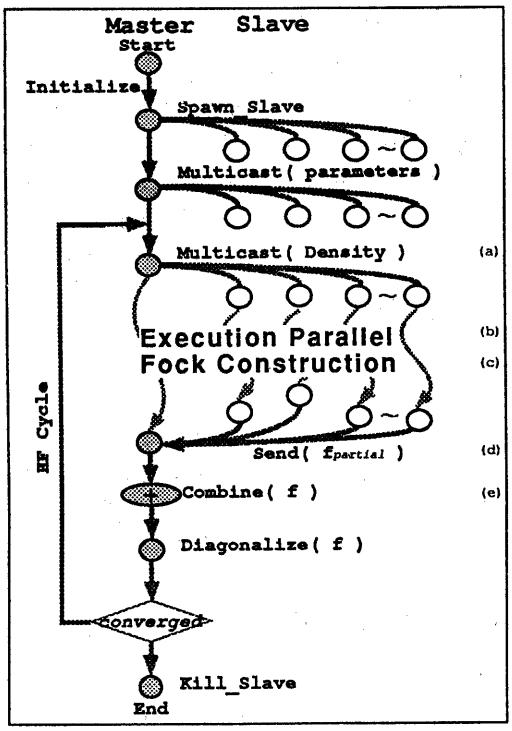

図 1: 並列化動作のブロック図

図1の並列実行中、"部分"生成の間は各スレーブタスクは全く独立であること、最初と最後に送受信すべき密度行列と Fock 行列が共に N^2オーダの量であること (例えば、基底数を千とすればちょうど8MB が通信すべき行列の大きさになる)、また並列化変数である積分の全リストが N^4オーダであることを強調しておきたい。HF(に限らず定性的には各種分子軌道計算法全般) の特徴でもあるこれらの要因が、結果的に種々のプラットフォームを通じて良好なスケーラビリティを実現する上で本質的な粗粒度性を与える。また、そのリストの長さからノード数が千以上のかなり大規模な MPP でもノード数を活かしきるポテンシャルを"生来的"に持っていることも特記出来るだろう。

次に、並列化の核心部分である上記の (c) のループ構造に話を移す。図2がそのループ部分であり、各スレーブタスク毎に予め送信された LID により

$$cnt = NPROC - LID \tag{7}$$

によって初期設定 (ここで、$NPROC$は全ノードの数) された変数 cnt を、積分添字 $pqrs$ の4重グランドループの内側で順次更新し、mod 判断により各々の処理を割り当てるラウンドロビン式の制御となっている。

```
C Quadruply-nested grand loops
      do p
        do q
          do r
            do s
              cnt = cnt + 1
              if(mod(cnt,NPROC).eq.0) then
                ( Calculate Integrals )
                ( Accumulate Contributions )
              end if
            end do
          end do
        end do
      end do
```

図 2:並列化構造

スレーブタスクは割り当てられた積分を計算し (もしくは後述のように"読み出し")、その寄与を式 (4) に従い"部分"Fock 行列の中に加算する。積分の計算は、基底関数の spd タイプの組み合わせによって (演算量の異なる) 個別のルーチンによって実行されることからスレーブタスク間では MIMD 実行の形になっており、短縮長にもバラツキがあることから本質的に全く均一の粒度とはなり得ないことを明記しておく。なお、各積分値計算そのものはベクトル実行可能であり、ノードがベクトル CPU の HPP の場合、特に優れたパフォーマンスが得られるはずである。

3.4 メッセージ通信の実装

WS クラスター等の分散プラットフォーム環境に対しては現在最も普及している PVM の 3.3.5

版をメッセージ通信のライブラリとして採用した。理由は、マルチベンダ (ヘテロ機種) 運用の容易さ、また近い将来に標準化が予想されるMPIへの互換が容易であるためであり、PVMの関数としてもスレーブタスク発生など前処理・後処理を除けばpvmfrecv、pvmfsend、及びpvmfmcastの基本的なものしか用いていない。一方、弊社で開発したMPP型専用並列機Cenju-3では固有の通信ライブラリPARALIB/CJを用いている。両者は共に並列化AMOSSのソース中に組み込まれており、PVMとPARALIB/CJのcall切り換えは入力パラメータで実行機環境の違いを指定するだけ出来るようになっている。

3.5 積分のハンドリング

2章で述べた反復の際の2電子積分値の不変性から、AMOSSでは以下の3種のハンドリング法がサポートされている。

(1) Conventional 方式

一旦積分を生成してファイルに貯め、HFの反復中に繰り返し読み出してFock行列を構成する"伝統的"方式。⇒ ベクトル機など高速CPUを積むマシンの登場以前は主流であったが、そのI/O効率の悪さから経過時間的に不利であり、また利用可能ファイル容量が計算規模上限を決めてしまう欠点もあるため、最近では次に挙げるDirect方式が用いられることが多い。

(2) Direct 方式

反復計算中に毎回積分を計算して、一切のファイル依存を止める方式(言うまでもなく、前提は高速のCPUの利用である)で、CPU時間消費は多いがConventionalに比べると経過時間的には有利である。ここで、AMOSSでは2種類のDirectモードがあり、In-coreでは利用可能な空きコアメモリ領域に可能な限り積分を貯めて収容しきれない部分のみを反復中に計算し、Pureではメモリに依らず全量を毎回計算する。⇒ 仮に潤沢なメモリ領域が確保出来て、全てIn-core処理できれば(一度だけ積分を計算すればよいので)最速なのは自明であろう。

(3) Semi-direct 方式

上記ConventionalとDirectの組み合わせ。例えば、In-Coreに入らない部分はファイルに貯める、などのハンドリングが可。

並列化は上記3つの方式に全て対応しており、並列環境下の資源を有効に使いきることが可能である。恩恵としては、

(i) In-coreのDirectによる並列化時には、積分を収容すべきコアメモリ領域がスレーブ側ノード上に分散されるため、単一ノードよりも"楽に"領域を確保出来るのでノード数が多くなるほど、全In-coreのチャンスがかなり大型の分子にも開ける。

(ii) Conventional等のファイル使用も、WSクラスターの場合、並列I/Oの扱いとなること、また利用可能量も大きくなるために、上述のデメリットが緩和される。

が挙げられる。もちろん、スレーブタスクの利用可能メモリが十分ではない場合、あるいはファイルI/Oを避けたい場合はPure-directとして実行すればよい。

4 並列化の評価試験の結果と考察

4.1 評価のシステム環境

評価試験を実行するWSクラスターとしては、弊社筑波研究所に設置されたECCS[1]を用いた。ECCSの実体は、図3のように16台の弊社EWS4800/320VXをEthernet-MPTで接続し、PVMデーモンとライブラリを実装したものである。また、MPP型専用並列機Cenju-3は64Mメモリの16ノードのものを使った(中央研究所に設置)。ECCSとCenju共に、独占使用により時

[1]Ews Cluster for Computational Science:ベットネーム

間、加速(スケーラビリティ)を測定した。なお、比較のために単一ベクトル機 SX-3/12R(中央研究所内) でも時間測定を行っている。

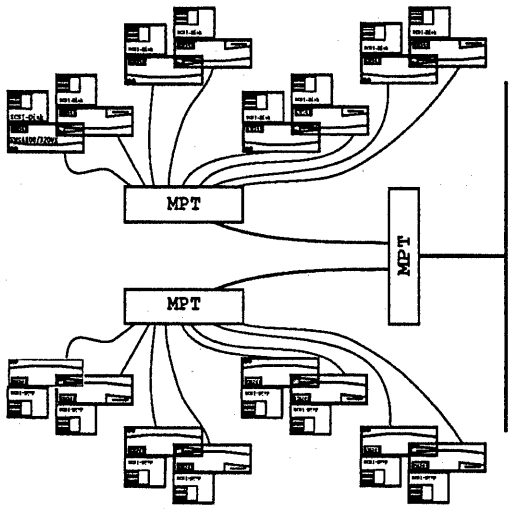

図 3: ECCS システム図

4.2 評価の対象データ

評価データとしては表1の5種類の中大型有機分子を選び、2種類の基底関数 (MIDI-4、MINI-4) を使用した。基底の数は 122～986 で、計算された積分の個数は 1800 万～43 億個にも達している (元来は N^4 オーダだが、実際には添字の対称性と適当な閾値判断によって有効な積分しか計算されない)。

表 1: 評価データ

データ名	分野	化学式
Artemisin	医薬品	$C_{15}H_{22}O_5$
Dobambc	液晶分子	$C_{31}H_{43}NO_3$
Maltpentaose	生体内塘	$C_{30}H_{52}O_{26}$
Graphite	ナノチューブ	$C_{86}H_{26}$
Chlorophyll Dimer	光合成	$C_{12}H_{72}N_8O_{12}Mg_2$

4.3 Pure-direct 実行の評価結果

まず最初に、ノード数の増加とスケーラビリティの関係を見るために、反復1回目の並列化 Fock 行列構成部分について Pure-direct モードでの評価を行った。ECCS では5種の分子全て、CenJu-3 では、Artemisin、Dobambc、及び Maltpentaose の3種である。図4にこれらの加速結果を示すが、16 ノード時においても、11～15 倍の加速が得られ、基底数の増加により問題規模が大きくなるほど率が上がる傾向が見られるなど、十分なスケーラビリティが出ており、積分駆動型の並列化が WS クラスター、MPP 共に実際に有望であることが実証された。

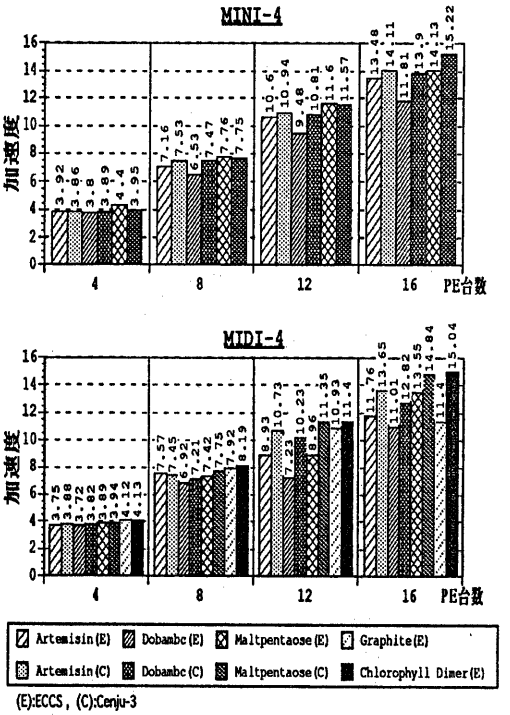

図 4: Cenju-3、ECCS 加速度

さて、ECCS と Cenju 各々の場合について分析を進めてく。ECCS では、理論加速の上限を越

えるものも見られるが、その理由の1つは、各スレーブによる式(4)の積分加算部分(図2を参照)に随伴するランダムアクセスによるキャッシュミスヒットが、分散によってアクセス領域が実効的に小さくなることにより減少したためと思われるが、これは同時にキャッシュ(2次キャッシュ容量が大きいことが望ましい)の存在とその効率が計算性能に与える影響の大きさを示唆している。一方、Cenju-3はキャッシュについてアクセス速度など特に入念に設計されている。実は、ECCSを構成するEWSとCenjuのノードCPUは共にMIPSのR4400系であり、内部クロックは順に200MHzと150MHzでむしろ前者の方が高いが、時間的には両者共にほとんど同じ時間で並列化HF計算をこなす。これは、Cenju-3の上記キャッシュの効率の良さと共にもう1つ、ネットワーク性能の良さも効いている。ECCSとCenjuのリダクション処理の時間を、16ノード時に前者を1として相対化すると僅か0.06であり、多段接続のクロスバーネットで"武装"した並列専用機の強みが活きており、ノード数がさらに増えて通信量が応じて増加していくとネットワーク性能の良さがさらに強みとなり、高いスケーラビリティが保持されると期待出来る。

4.4 In-core-direct 実行の評価結果

次ぎに、ECCS(16ノード時)の完全In-core-direct実行とSX-3のPure-direct実行を相対比較してみる(基底はMINI-4で、収束までの全HF計算が測定対象)。表2の結果を見ると、なんとSX-3よりも3倍ほどクラスターの方が速い。計算資源の確保の観点から見ると、SX-3のような単一ノードの共同利用機上で0.7GBものメモリ領域を1ユーザジョブが利用するのはかなり難しく、Pure-directでの実行形態の方がむしろ多い。一方、16ノード構成のECCSの場合、同じ0.7GBの空間を得るのに1ノード当たりでは僅か44MBしか必要としていないので、完全In-core実行のチャンスは多い。WSでは専用機に比べてメモリの拡張、ノード数を増やすことが共に容易である点も正因子として挙げられる。極論すれば、メモリ資源の分散確保条件が整えば、並列化HFにおけるWSクラスターはSX-3に匹敵(あるいはそれ以上の)パフォーマンスを発揮することが可能と言える。

表2: In-core結果比較

データ	メモリ領域	SX-3	ECCS
Artemisin	0.5GB	1	0.3
Dobambc	0.7GB	1	0.4

4.5 Conventional 実行の評価結果

今度は、従来は経過時間的には不利であると言われてきた、積分をファイルに蓄積するConventional実行についても、WSクラスターの分散処理により"旨味"が出てくる例を示したい。つまり、積分ファイルは各スレーブWS毎に随伴する局所ディスク上に置かれることで、読み書き操作が自動的に効率の良い並列I/Oとなる点に注目する。表3に、SX-3のPure-directとの相対比較を示す(MIDI-4基底で、やはり全HF処理時間が対象)。こちらも、SX-3に遜色の無い時間で計算出来ることが分かる。今回評価したデータの中で最大規模のMIDI-4のChlorophyll Dimerでも、全体で128GBのファイル領域を確保出来れば、43億個の積分(添字のラベル分も合わせて)全てを格納出来るが、これはWSクラスターのような分散環境では現実的な話であり、RAIDなどがWSに普及してくれば、しばらく廃れていたConventional手法が見直される可能性が高い(一般的に言って、メモリよりはディスクの方が安価であることに注意)。

表3: Conventional結果比較

データ	ファイル領域	SX-3	ECCS
Artemisin	4.4GB	1	1.35

4.6 評価結果まとめ

ここでは、評価結果から得られた分散並列化の意味を総括的に考えてみたい。まず言えるのは、分散化によってメモリやファイルなど計算資源を「ユーザが簡単・手軽にかつ柔軟に使える」ようになることで、その"威力"は既に見てきたとおりであるが、その大前提が分散並列環境下で運用されるソフトの基本アルゴリズムの"質の良さ"に掛かっていることは認識されるべきである。つまり、参照の局所性に留意し、さらに通信を最小化した粗粒度性をソフトがアルゴリズムレベルから生来的に有していることが本質的な要求なのである。幸い、私たちが今回試行した AMOSS の並列化 HF はこの要求を十分満足しているようである。

試行として次に進むべきは、今回個別に評価した ECCS と Cenju-3 を複合した"初歩的な"メタシステム上での評価である。さらには、SX-3 の後継である SX-4 をも複合するシナリオも有り得る。いずれにせよ、メタシステムによる実行は、今後の TFLOPS→PFLOPS の HPC の指向方向の中で必然であり、まず ECCS-Cenju 系できちんとパフォーマンスが出せることが試金石として重要な意味を持っている。

5 まとめ

今回の報告では、AMOSS を軸にした非経験的分子軌道計算の並列処理に対する私たちの取り組みを HF を例に紹介した。アルゴリズムは、粗粒度の積分駆動型でメッセージ通信モデルに基づく。WS クラスター (ECCS) と MPP(Cenju-3) での実行共に、良好な加速スケーラビリティが得られた。また、メモリやファイルの分散による高速化の可能性も論じた。

参考文献

[1] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck and Vaidy Sunderam 、"PVM3.3 User's Guide and Reference Manual"、(1994)

[2] Stefan Brode, Hans Horn, Michael Ehrig, Diane Moldrup, Julia E.Rice, and Reinhart Ahlrichs、"Parallel Direct SCF and Gradient Program for Workstation Cluster"、*J.Comp.Chem.* 14(1993)1142

[3] Hans P.Lüthi, John E.Mertz, Martin W.Feyereisen and Jan E.Almlöf、"A Corse-Grain Parallel Implementation of the Direct SCF Method"、*J.Comp.Chem.* 14(1993)814